Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

A hybrid artificial bee colony algorithm with an iterated local search mechanism for distributed no-wait flowshop problems with preventive maintenance

Chuan-Chong Li[a], Yuan-Zhen Li[a*] , Lei-Lei Meng[a] and Biao Zhang[a]

[a]*School of Computer Science, Liaocheng University, 25200, Liaocheng, China*

C H R O N I C L E

Article history:
Received November 16 2024
Received in Revised Format December 4 2024
Accepted February 24 2025
Available online February 24 2025

Keywords:
Distributed permutation flowshop scheduling
Makespan
No-wait
Preventive maintenance
Artificial bee colony algorim

A B S T R A C T

In this paper, a distributed no-wait permutation flowshop scheduling problem with a preventive maintenance operation (PM/DNWPFSP) is investigated. A mixed-integer linear programming model for the PM/DNWPFSP is established. The problem characteristics and preventive maintenance characteristics of the PM/DNWPFSP are analyzed, and an accelerated calculation method of the completion time is proposed. A hybrid artificial bee colony (HABC) algorithm with an iterated local search mechanism for neighborhood search is proposed. To improve the quality of the solution, the shift, the swap and the hybrid operators are conducted in the critical factory. A local search operator based on the shift, the swap and the hybrid operators is proposed to jump out of local optima. A large number of experiments are conducted to evaluate the performance of the proposed HABC. The experimental results show that the proposed HABC algorithm has many promising advantages in solving the PM/DNWPFSP.

1. Introduction

In the context of the globalized economic era, cooperation models between companies have become diversified (Naderi and Ruiz, 2010). The production mode of enterprises is changing. Many factories have been established around the world to improve competitiveness and the ability to cope with various risks. The production mode has changed from traditional centralized production to distributed production (Allahverdi, 2015). The distributed manufacturing strategy is a new manufacturing model with scalability, flexibility, and high reliability that can effectively improve the utilization rate of enterprises and factories distributed in various places. With the rapid development of computer information technology and internet communication technology, distributed production methods can more reasonably allocate production resources, optimize the production structure of products, and share production resources (Lu et al., 2022). Through the integrated production of production resources, a win–win effect is achieved, thereby improving the efficiency and competitiveness of enterprises.

To fully utilize production resources, reducing production costs, improving production efficiency and reducing the construction period have become important issues for modern enterprises. Scheduling plays a very important role in many industries, such as manufacturing, communication networks and transportation. Production scheduling technology has become a key research problem in related industries. As a very important category, the distributed permutation flowshop scheduling problem (DPFSP) is broadly related to the widespread popularity of global industrial models (Jia et al., 2007; Ying et al., 2017). The difficulty of solving the DPFSP increases geometrically as the complexity and scale of the problem increase (Jing et al., 2021; Yang et al., 2022).

The actual production conditions cause that jobs are not processed in an ideal manner on machines. The jobs in some industries, such as the steel, pharmaceutical, and mining industries, must be processed continuously without interruption or waiting (Allahverdi et al., 2008). Otherwise, jobs are unqualified. The classic distributed permutation flowshop scheduling problem is based on the assumption that machines can run continuously. Indeed, machines are worn due to continuous work. To ensure

* Corresponding author
E-mail liyuanzhen@lcu.edu.cn (Y.Z. Li)
ISSN 1923-2934 (Online) - ISSN 1923-2926 (Print)
2025 Growing Science Ltd.
doi: 10.5267/j.ijiec.2025.2.003

the quality of jobs and the protection of machines, machines need regular preventive maintenance (PM) (Mao et al., 2021; Naderi et al., 2011). The distributed no-wait permutation flowshop scheduling problem with preventive maintenance (PM/DNWPFSP) with completion time criterion is addressed in this paper. A hybrid artificial bee colony (HABC) algorithm is proposed, which improves the local search operation and evolution strategy of the algorithm to obtain high-quality solutions. A large number of experiments verify the effectiveness of the HABC algorithm in solving the PM/DNWPFSP.

The rest of this paper is structured as follows. Section 2 provides a review of relevant research. The definition, examples and characteristics of the PM/DNWPFSP are introduced in section 3. In section 4, the improved HABC is described in detail. In section 5, the parameters of the HABC algorithm are adjusted to obtain the optimal algorithm performance. Section 6 shows a comprehensive comparison of HABC with state-of-the-art algorithms. Finally, section 7 summarizes the full text, and some suggestions for future work are provided.

2. Literature Review

This section summarizes three issues related to the PM/DNWPFSP, including the DPFSP, no-wait flowshop scheduling problem (NWFSP), and preventive maintenance of machines. The DPFSP involves *f* identical factories that are flowshopped with *m* machines. There are *n* jobs to be processed in one of the factories. It is an urgent problem to determine which factory the job is assigned to and the processing sequence in each factory for the DPFSP. Nader and Ruiz (2010) established six mixed integer linear programming (MILP) models for the DPFSP and proposed two simple factory allocation rules and 14 heuristics. Gao *et al*. (2013) presented a new tabu algorithm to solve the DPFSP. A large number of experiments showed that the proposed tabu search algorithm was superior to the heuristic proposed by Naderi and Ruiz (2010). Victor and Jose (2014) presented a bounded-search iterated greedy algorithm (BSIG) for solving the DPFSP. Wang *et al*. (2013) proposed an estimation of the distribution algorithm to solve the DPFSP, applying the earliest completed factory rule. A probabilistic model for describing the solution space was established, and a mechanism was proposed for updating the probabilistic model of the better solution. The algorithm used these probabilistic models to search for new high-quality solutions more efficiently. Pan *et al*. (2019) proposed four intelligent optimization algorithms and three heuristics to solve the DPFSP while minimizing the total flowtime. Meng *et al*. (2022) used MILP models and a constraint programming model to solve distributed flexible job shop scheduling problems. Li *et al*. studied the DPFSP with mixed no-idle constraints (Y. Li et al., 2021) and the distributed assembly mixed no-idle permutation flowshop scheduling problem with total tardiness criterion (Li et al., 2022). Rossi and Nagano (2021) presented an MILP formulation, a novel constructive heuristic, and iterated greedy algorithms for the distributed mixed no-idle flowshop. An effective iterated greedy algorithm based on a learning-based variable neighborhood search (Han et al., 2022) was proposed for the distributed blocking flowshop scheduling problem. A discrete artificial bee colony (Yu et al., 2022) was presented for the DPFSP with sequence-dependent setup time. The DPFSP has become a research hotspot.

The NWFSP is one of the key variants of the flowshop scheduling problem and one of the hotspots in scheduling research (Pei et al., 2019). Due to the special manufacturing process of some jobs, there are no-waiting constraints in many industries, such as steel, chemicals, plastics, electronics, pharmaceuticals, food processing, lean manufacturing, baking production, and robotic cell production. The processing of each job must be continuous. The NWFSP plays an important role in many industries and has received much research and attention. RöCK (1984) proved that the NWFSP with makespan criterion is an NP-difficult problem when the number of machines in the factory is greater than 2. As the scale of the problem increases, the NWFSP cannot be solved in a reasonable time by branch and bound or mixed integer programming methods. Researchers have proposed many heuristics to address this problem. Aldowaisan and Allahverdi (2004) proposed six construction heuristics to solve the NWFSP to minimize total flowtime. Subsequently, Ye *et al.* (2017) presented an average idle time heuristic to optimize the NWFSP with makespan criterion. In addition, researchers have proposed many effective metaheuristic algorithms to address the NWFSP. Pan *et al.* (2008) proposed an improved iterated greedy algorithm to solve the NWFSP with makespan criterion. Tseng and Lin (2010) improved the genetic algorithm by using a novel local search scheme for the NWFSP. Gao *et al.* (2011) proposed a discrete harmonic search algorithm for the NWFSP to minimize total flowtime. Ying and Lin (2020) addressed a multistart simulated annealing algorithm with bidirectional shift timetabling for the no-wait jobshop scheduling problem. Recently, Shao *et al.* (2021) proposed a constructive heuristic for variable neighborhood descent to solve the distributed no-wait flexible job shop scheduling problem with makespan criterion. Allali *et al.* (2022) presented three naturally inspired meta-heuristics: The genetic algorithm, the artificial bee colony algorithm and the migratory bird optimization algorithm for the distributed no-wait permutation flowshop scheduling problem. Zhu *et al.* (2022) proposed a discrete knowledge-guided learning fruit fly optimization algorithm for the distributed no-wait flowshop scheduling problem with total weighted earliness and tardiness criteria. Li *et al.* (2021) described a discrete artificial bee colony algorithm for the distributed heterogeneous no-wait flowshop scheduling problem to minimize makespan.

To eliminate phenomena such as equipment failure and unplanned interruption in actual production, PM should be conducted regularly or irregularly. Researchers have studied PM in scheduling. In view of pure machine PM, Wang *et al.* (2002) proposed that the existing machine PM scheduling model is mostly repaired in the form of a critical point. Cassady and Kutanoglu (2012) established an integrated optimization model for coordinated preventive maintenance and stand-alone scheduling. Ruiz *et al.* (2007) studied the flowshop sequencing problem with machine maintenance and analyzed PM strategies. Pan *et al.* (2010) studied the single machine scheduling problem with machine maintenance and proposed an integrated model of variable time-affected maintenance for machine aging. Miyata *et al.* (2019) proposed a mathematical model and a construction heuristic for the no-wait flowshop scheduling problem with dependent-sequence setup times and preventive maintenance. Lei and Liu (2020) studied distributed unrelated parallel machine scheduling with PM and proposed an artificial bee colony algorithm to handle it. Mao *et al.* (2021) proposed a multistart iterated greedy algorithm for the DPFSP with PM and makespan criterion. Subsequently, Mao *et al.* (2022) proposed a mathematical model and a hash map-based algorithm to solve the DPFSP with PM. Meng *et al.* (2022) studied mixed-model assembly line balancing considering PM. To speed up the decoding speed of the algorithm, a variable step decoding method for hybrid models was designed. Miyata and Nagano (2021) investigated distributed no-wait flowshop scheduling with sequence-dependent setup times and maintenance operations to minimize makespan optimization goals.

From the above overview, many heuristic and metaheuristic algorithms have been proposed to solve the NWFSP, PM and DPFSP. However, there are few research publications on the DPFSP considering the no-wait constraint and machine PM simultaneously. In practice, any scheduling problem needs to consider the nature of jobs and the actual working conditions of machines to ensure normal production. Therefore, it is meaningful to consider the no-wait constraints of jobs and the PM of machines in the DPFSP. In this paper, the PM/DNWPFSP, which has a variety of constraints and special characteristics, is investigated. A mathematical model of the considered problem is established. A hybrid artificial bee colony (HABC) algorithm with an iterated local search mechanism for neighborhood search is proposed and compared with the most advanced algorithms in the literature to verify the effectiveness of the HABC algorithm.

3. Problem Description

The PM/DNWPFSP is described below. There are n jobs to be assigned to multiple factories with the same series of processing machines. In each factory, all assigned jobs are processed on machine set M along the same route from machine 1 to machine m. The processing time of a job $j \in \{1,2,\dots,n\}$ on a machine $i \in \{1,2,\dots,m\}$ is expressed as $P_{i,j}$. Each machine can process only one job at a time, and each job can be processed on only one machine at a time. At time 0, all jobs and machines are ready. Once a job starts processing on a machine, it cannot be interrupted. All job processing requires a no-wait constraint. Therefore, once processing of a job has begun, it is necessary to finish all processes without stopping. To ensure the maintainability and reliability of the production system, PM operations are conducted regularly on all machines. The health of a machine indicates whether maintenance is needed, which is expressed as ML. The operation of job j on machine i decreases the health value of machine i by $P_{i,j}$. When the health value of a machine is insufficient to process the next job, the machine must be maintained. After maintenance operations, a machine is restored to its best state; that is, the health of the machine is restored to its maximum value ML^i_{max}. The maintenance time on machine i is denoted MT_i. PM operations cannot be interrupted. $P_{i,j}$, MT_i, and ML^i_{max} are assumed to be the same in all factories. The symbols are shown in Table 1.

Table 1
Symbol definition

Notations	
$\boldsymbol{n}$	Number of jobs to be processed.
$\boldsymbol{j}$	Index for jobs, $j \in \{1,2,\dots,n\}$.
$\boldsymbol{m}$	Number of machines in each factory.
$\boldsymbol{i}$	Index for machines, $i \in \{1,2,\dots,m\}$.
$\boldsymbol{f}$	Number of factories.
$\boldsymbol{k}$	Index for factories, $k \in \{1,2,\dots,f\}$.
$\boldsymbol{n_k}$	Number of jobs assigned in factory *k*.
$\boldsymbol{l}$	Index for job positions in each factory, $l \in \{1,2,\dots,n_k\}$.
$\boldsymbol{O_{i,j}}$	The operation of job *j* on machine *i*.
$\boldsymbol{P_{i,j}}$	The processing time of $O_{i,j}$.
$\boldsymbol{MT_i}$	The maintenance time on machine *i*.
$\boldsymbol{C_{i,k,l}}$	Continuous variable for the completion time of the job in position *l* on machine *i* in factory *k*.
$\boldsymbol{ML_{i,k,l}}$	A continuous variable representing the maintenance level of machine *i* in factory *k* before processing the job assigned to position *l*.
$\boldsymbol{ML^i_{max}}$	Maximum of maintenance level of machine *i*.
$\boldsymbol{M}$	A number sufficiently large.
$\boldsymbol{C_{max}}$	The completion time of a scheduling.
Decision variables	
$\boldsymbol{X_{j,k,l}}$	Binary variable: 1 if job *j* is assigned to position *l* in factory *k* and 0 otherwise.
$\boldsymbol{Y_{i,k,l}}$	Binary variable: 1 if a PM operation is conducted on the machine *i* before processing the job assigned to position *l* in factory *k* and 0 otherwise.

3.1 The MILP model

Based on the above assumptions and defined notations, the MILP model of the addressed PM/DNWPFSP can be formulated as follows:

Objective:

$$min \quad C_{max} \tag{1}$$

subject to

$$ML_{i,k,0} = ML^i_{max}, \forall i,k \tag{2}$$
$$ML_{i,k,l} - ML^i_{max} \le M\left(1 - Y_{i,k,l-1}\right), \forall i,k,l > 1 \tag{3}$$
$$ML_{i,k,l} - ML^i_{max} \ge M\left(Y_{i,k,l-1} - 1\right), \forall i,k,l > 1 \tag{4}$$

$$ML_{i,k,l}-\left(ML_{i,k,l-1}-\sum_{j=1}^{n}X_{j,k,l-1}P_{i,j}\right)\leq M\left(Y_{i,k,l-1}\right),\forall i,k,l>1 \tag{5}$$

$$ML_{i,k,l}-\left(ML_{i,k,l-1}-\sum_{j=1}^{n}X_{j,k,l-1}P_{i,j}\right)\geq -M\left(Y_{i,k,l-1}\right),\forall i,k,l>1 \tag{6}$$

$$ML_{i,k,l}\geq\sum_{j=1}^{n}X_{j,k,l}P_{i,j}\,,\forall i,k,l \tag{7}$$

$$\sum_{l=1}^{n}\sum_{k=1}^{f}X_{j,k,l}=1\,,\forall j \tag{8}$$

$$\sum_{j=1}^{n}X_{j,k,l}\leq 1,\forall j,k \tag{9}$$

$$C_{i,k,l}\geq C_{i,k,l-1}+\sum_{j=1}^{n}X_{j,k,l}\,P_{i,j}+Y_{i,k,l}\cdot MT_i,\forall i,k,l>1 \tag{10}$$

$$C_{i,k,l}=C_{i-1,k,l}+\sum_{j=1}^{n}X_{j,k,l}\,P_{i,j},\forall i>1,k,l \tag{11}$$

$$C_{max}\geq C_{i,k,l},\forall i,k,l \tag{12}$$

$$C_{i,k,l}>0,\forall i,k,l \tag{13}$$

$$X_{j,k,l}\in\{0,1\},\forall j,k,l \tag{14}$$

$$Y_{i,k,l}\in\{0,1\},\forall i,k,l \tag{15}$$

The goal is to minimize the completion time (C_{max}), which is described by Eq.(1). Constraint set (2) means that the health value of machine *i* at position 0 is ML_{max}^{i}. Constraint sets (3) and (4) indicate that the health value of a machine returns to the original value ML_{max}^{i} after a maintenance operation. Constraint sets (5) and (6) indicate that the health value of a machine decreases after processing a job. Constraint set (7) ensures that the health value before a machine processes a job is larger than the processing time of the job. Otherwise, the machine is damaged. Constraint set (8) restricts each job to one position. Constraint set (9) means that at most one job is placed in each position. Constraint set (10) ensures that the processing of the job on position *l* in factory *k* must not be started on machine *i* until the processing of the previous job assigned to machine *i* in the same factory and the PM operation on the same machine are completed. Constraint set (11) means that jobs are processed without waiting; that is, the start time of the job on position *l* on machine *i*-1 in factory *k* is equal to the completion time of the job on position *l* on machine *i* in factory *k*. Constraint set (12) defines the completion time of a schedule. Constraint sets (13), (14) and (15) define the value ranges of the intermediate and decision variables.

3.2 An Illustrative Example

We consider an example in which there are two factories (*f* = 2), two machines (*m* = 2), and eight jobs (*n* = 8). All job processing has a no-wait constraint. The processing times are given in Table 2. One possible solution is $x_{1,1,1}= x_{3,1,2} = x_{5,1,3}= x_{7,1,4}= x_{2,2,1} = x_{4,2,2}= x_{6,2,3} = x_{8,2,4}=1$, and the other decision variables are equal to zero. That is, jobs 1, 3, 5, and 7 are processed sequentially in factory 1, and jobs 2, 4, 6, and 8 are processed sequentially in factory 2.

Table 2
Processing times $P_{i,j}$

	J₁	**J₂**	**J₃**	**J₄**	**J₅**	**J₆**	**J₇**	**J₈**	MT_i	ML_{max}^{i}
M₁	3	3	6	6	3	3	6	6	8	12
M₂	5	3	5	5	5	3	5	5	6	10

The Gantt diagram of the distributed flowshop scheduling with no-wait constraint is shown in Fig. 1. The wait times between the successive operations of jobs on all machines are zero. Finally, the makespan of this schedule is $C_{2,2,4}+p_{1,8}=20+5=25$.

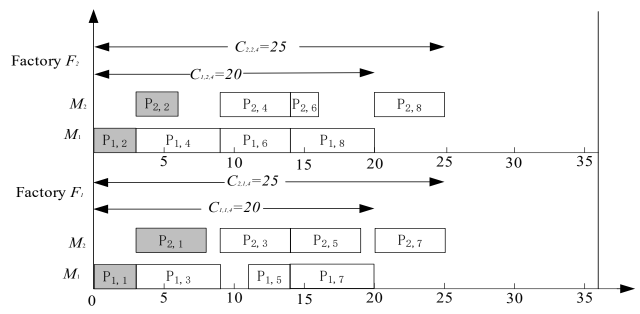


Fig. 1. Gantt chart of a solution for distributed flowshop scheduling with no-wait constraints

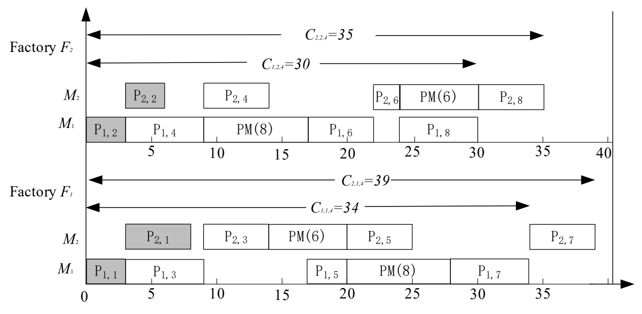
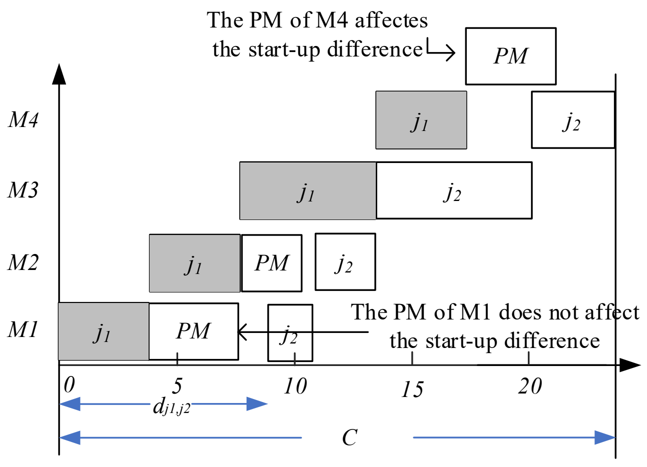


Fig. 2. Gantt chart of scheduling for distributed flowshop scheduling with no-wait constraints and machine PM

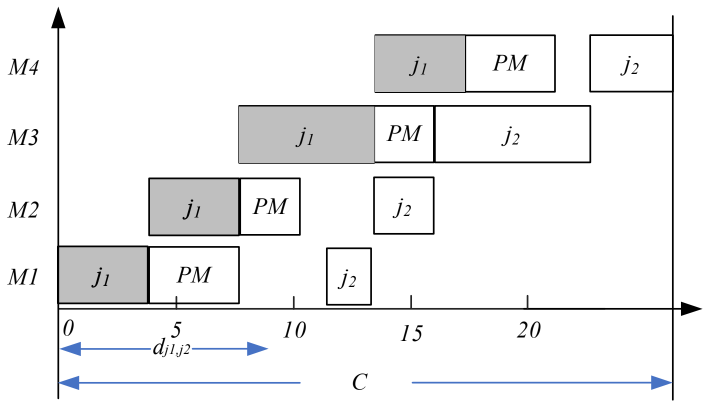
The Gantt diagram of the distributed flowshop scheduling considering no-wait constraints and machine PM is shown in Fig.2. As seen from the Gantt chart, a machine in the factories needs to be repaired after overseeing several operations. In Fig.2, the scheduling criterion of the PM operation is applied to the previous optimal solution. The PM operations disrupt tight schedules. The machines in each factory have considerable idle time during processing, which leads to greater completion times. Therefore, the minimum makespan is $C_{1,1,4} + p_{2,7}$ =34+5=39.

3.3 Problem Characteristics

In the no-wait flowshop, the start time difference between two adjacent jobs can be used to quickly calculate the makespan. We found that the maintenance of some machines (M_3 and M_4 in Fig. 3 (a)) affects the start time difference between the two jobs J_1 and J_2, while the maintenance of other machines (M_1 and M_2 in Fig. 3 (a)) does not affect the start time difference. Furthermore, Fig. 3 (b) shows that the PM of machines 1, 2 and 4 does not cause an additional starting time difference; in the case of machine 3 maintenance,



(a)



(b)

Fig. 3. Schematic of the problem characteristics of PM/DNWPFSP

For jobs j_1 and j_2 in Fig. 4, without maintenance operations, the start time difference d_{j_1,j_2} between the two jobs can be calculated using Eq. (16). The idle time of each machine $AT_{j_1,j_2,i}$ can be obtained using Eq. (17). The extra difference $Ed_{j_1,j_2,i}$ represents the increase in the start time difference caused by the maintenance of machine i, which can be calculated by Eq. (18).

$$d_{j_1,j_2} = \max\left\{\max_{2\le i\le m}\{\sum_{y=1}^{i+1} p_{j_1,y} - \sum_{y=1}^{i} p_{j_2,y}\}, p_{j_1,1}\right\} \tag{16}$$

$$AT_{j_1,j_2,i} = \left\{d_{j_1,j_2} - \sum_{y=1}^{i+1} p_{j_1,y} + \sum_{y=1}^{i} p_{j_2,y}\right\} \tag{17}$$

$$Ed_{j_1,j_2,i} = \max(0, MT_i - AT_{j_1,j_2,i}) \tag{18}$$

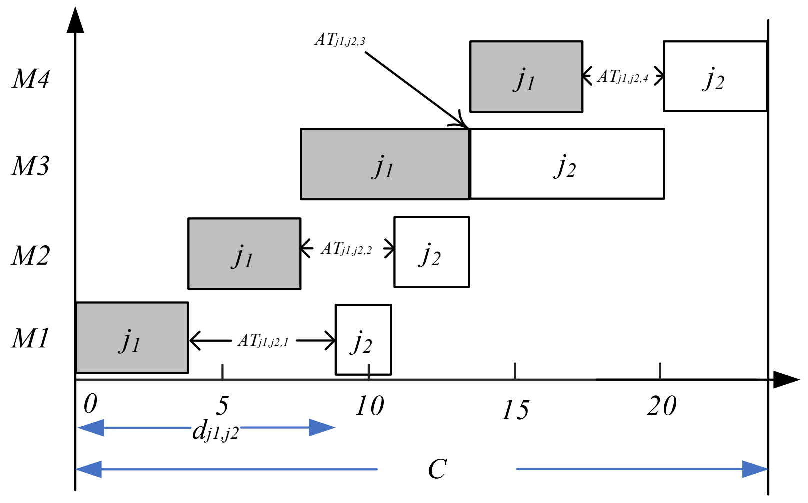


Fig. 4. The start time difference, idle time, and extra difference

The condition $MT_i \leq AT_{j_1,j_2,i}$ indicates that maintenance machine i does not increase the start time difference and $Ed_{j_1,j_2,i} = 0$. The condition $Ed_{j_1,j_2,i_1} > Ed_{j_1,j_2,i_2}$ means that the increase in the start time difference caused by the maintenance of machine i_1 is greater than the increase in the start time difference caused by the maintenance of machine i_2; the condition $Ed_{j_1,j_2,i_1} > Ed_{j_1,j_2,i_2}$ also means that if machine i_1 is being maintained, incidental maintenance machine i_2 does not cause a further increase in the start time difference.

3.4 Accelerations

Accelerations are widely used in scheduling problem research, which can greatly reduce the number of calculations and can improve the calculation efficiency. Based on the problem characteristics, we design the generation of a maintenance matrix and the quick calculation of the makespan when the sequences of jobs in factories are determined. The maintenance matrix is $R = [Y_{i,k,l}]_{m*f*n}$. The maintenance plan and the actual start time difference between the two jobs in Fig. 4 can be obtained as shown in **Algorithm 1**. $Ed_{j_1,j_2,i} (i \in M)$ is sorted from large to small, and the order of machines obtained is $i'(1), i'(2), \dots, i'(m)$. If machine $i'(1)$ PM before job j_2 is processed, the actual start time difference is $Ed_{j_1,j_2,i'(1)} + d_{j_1,j_2}$, and PM is conducted for machines $i'(1) - i'(m)$. If the first machine $i'(1)$ does not require PM, we check whether the second machine $i'(2)$ requires PM, and so on.

Algorithm 1. ActualStartTimeDifference

```
1:   ($i'_1$,$i'_2$,…,$i'_m$) is machine permutation according to descending order of $Ed_{j_1,j_2,i}$;
2:   for $i = 1$ to $m$ do
3:     if machine $i'_i$ need preventive maintenance  do;
4:       Actual start time difference $ad = Ed_{j_1,j_2,i'_i} + d_{j_1,j_2}$;
5:       for $ii = i'_i$ to $m$ do
6:         Perform preventive maintenance for machine $i'_{ii}$;
7:       endfor
8:       break
9:     endif
10:  endfor
11:  return $ad$
```

The completion time of a factory is equal to the sum of the actual start time difference of all jobs in the factory plus all the processing times of the last job in this factory. The start time difference, idle time, extra difference, and sorting of extra difference of each job pair can be calculated in advance.

4. The Proposed HABC Algorithm

The artificial bee colony (ABC) algorithm is a new swarm intelligence optimization algorithm based on the self-organizing model of the bee colony in nature, in which the honeybees are simulated to achieve the optimization process through communication, transformation, and collaboration between bees of different roles. The bees in the ABC consist of three main groups: employed bees, onlooker bees, and scout bees. The process of bees looking for high-quality food sources is equivalent to the process of finding the best feasible solution for the optimization problem. A food source represents a viable solution, and the quality of a food source represents the fitness of a viable solution. The speed at which a bee collects honey is equivalent to the speed at which the optimization problem is solved. Bees are constantly updating and looking for food sources, while the quality of feasible solutions is constantly improving, which is a continuous iterative process. The ABC (Pan and Zhao et al., 2008) has received widespread attention from researchers at home and abroad.

4.1 Solution Representation

The representation of a solution is one of the keys to designing an algorithm, which has a significant impact on the efficiency of the algorithm (Pan, Gao, and Li et al., 2019). The representation of a solution uses the same representation methods in the literature (Pan, Gao, and Wang et al., 2019). A two-dimensional array is used to represent jobs assigned to factories. There are f rows in the array, which is a job sequence $\pi_k = (\lambda_k)$, consisting of a set of jobs assigned to the factory. The jobs in each sequence are sorted according to the order in which they are processed. Therefore, a solution is expressed as $sol = (\pi_1, \pi_2, \dots, \pi_f)$. This sequence-based solution is widely used in scheduling literature due to its intuitive representation and programming convenience (Pan and Zhao et al., 2008). The solution for the example in section 3.3 can be expressed as $sol = (\pi_1, \pi_2)$, where $\pi_1 = (1,3,5,7)$, and $\pi_2 = (2,4,6,8)$.

4.2 Heuristics

The initial population of the basic ABC algorithm is generated randomly, which does not guarantee the quality of the initial solutions. Heuristics based on the problem characteristics are necessary. The NEH algorithm has been proven to be a very effective algorithm in solving the PFSP. Jobs with large total processing times are inserted preferentially in the NEH algorithm to ensure the quality of the initial solution. Based on the NEH algorithm, the DNEH algorithm is proposed to generate an initial solution, the pseudocode for which is shown in Algorithm 2. First, the sum of the processing times of each job on all machines T_i is calculated (Line 1). A temporary job permutation λ is obtained according to the nondescending order of T_i (Line 2). The first f jobs in λ are put into f empty factories (Lines 3 and 4). The remaining jobs in λ are removed in turn and try to insert the optimal location (Lines 5-14). Finally, the repair matrix RA and the completion time C_{max} are updated.

Algorithm 2. DNEH

```
1:  Compute $T_i = \sum_{i=1}^{m} P_{i,i}$ for each job $i \in N$;
2:  Generate job permutation λ=($\lambda_1$,$\lambda_2$,…,$\lambda_n$) according to non-descending order of $T_i$;
3:  π=($\pi_1$,$\pi_2$,…,$\pi_f$),where $\pi_k = (\lambda_k), k = 1,2, \dots, f$;
4:  Remove jobs $\lambda_1$,$\lambda_2$,…,$\lambda_f$ from $\lambda$;
5:  **While** sizeof($\lambda$) **do**   //NEH enumeration procedure
6:      Extract the first job $j$ from $\lambda$;
7:      **for** $k = 1$ to $f$ **do**
8:          Test job $j$ at all the possible positions of $\pi_k$;
9:          $\Delta_k$ = minimum increase of makespan;
10:         $\xi_k$ = position resulting in $\Delta_k$;
11:     **endfor**
12:     $k^* = \arg(\min_{k=1,2,\dots,f} \Delta_k)$;
13:     Insert job $j$ at position $\xi_{k^*}$ of $\pi_{k^*}$;
14: **endwhile**
15: Update $RA$ and $C_{max}$;
16: **return** $sol = (\pi_1, \pi_2, \dots, \pi_F)$
```

4.3 Operators

In flow line production, if the processing time of one process is much greater than the processing time of other processes, the output of the entire flow line is restricted, and that process is called the bottleneck process. The bottleneck process determines the maximum production capacity, and the improvement of the bottleneck process can balance the production line and can improve production efficiency. Similarly, in the critical path method, to improve project efficiency, it is always necessary to optimize the longest or most time-consuming path in the entire project. Based on the idea of the bottleneck process and the critical path method, we think that the critical factory (denoted as F_c.) with the largest completion time should be optimized first, which ensures that the completion time is likely to decrease.

Based on the characteristics of the PM/DNWPFSP and the characteristics of the ABC algorithm, we define five operators, namely, Shift, Swap, ILSShift, ILSSwap, and ILSHybrid.

（1） The pseudocode of the shift operator is shown in **Algorithm 3**. It randomly selects a job j_1 from the critical factory F_c and moves job j_1 to another randomly selected position p in all factories. Finally, the repair matrix RA and the completion time C_{max} are updated.

Algorithm 3. Shift(individual sol)

```
1:  Find the critical factory $F_c$ with the maximum makespan in $sol$;
2:  Randomly select a job $j_1$ in factory $F_c$
3:  Randomly select a factory $f$
4:  Randomly select a position $p$ in factory $f$;
5:  Shift job $j_1$ to position $p$ ;
6:  Update $RA$ and $C_{max}$;
7:  return $sol$ and $C_{max}$;
```

(2) The Swap operator shown in **Algorithm 4** randomly selects a job j_1 in the critical factory F_c and another random job j_2 in all factories and then swaps jobs j_1 and j_2. Finally, the repair matrix RA and the completion time C_{max} are updated.

Algorithm 4. Swap(individual sol)

```
1:  Find the critical factory $F_c$ in $sol$;
2:  Randomly select a job $j_1$ in factory $F_c$
3:  Randomly select a factory $f$
4:  Randomly select a job $j_2$ in factory $f$;
5:  Swap job $j_1$ and job $j_2$;
6:  Update $RA$ and $C_{max}$;
7:  return $sol$ and $C_{max}$;
```

(3) The pseudocode of the operator ILSShift is shown in **Algorithm 5**. The ILSShift operator uses the Shift operator to make 60 attempts in the neighborhood of sol to find a better solution.

Algorithm 5. ILSShift(individual sol)

```
1:  $tempsol = sol$;
2:  for $i$ = 1 to 60 do
3:    $sol' = sol$;
4:    $[sol', C_{max}]$ = Shift($sol'$);
5:    if $sol'$ is better than $tempsol$ do $tempsol = sol'$ endif
6:  endfor;
7:  $sol = tempsol$;
```

(4) The pseudocode of the operator ILSSwap is shown **Algorithm 6**. The ILSSwap operator uses another operator, the Swap operator, to make 60 attempts in the neighborhood of sol to find a better solution.

Algorithm 6. ILSSawp(individual sol)

```
1:  $tempsol = sol$;
2:  for $i$ = 1 to 60 do
3:    $sol' = sol$;
4:    $[sol', C_{max}]$ = Swap($sol'$);
5:    if $sol'$ is better than $tempsol$ do $tempsol = sol'$ endif
6:  endfor;
7:  $sol = tempsol$;
```

(5) The pseudocode of the operator ILSHybrid is shown in **Algorithm 7**. It is a combined operator that picks the ILSShift operator with a 50% probability and picks the ILSSwap operator with a 50% probability.

Algorithm 7.ILSHybrid(individual sol)

```
1:  r=(double)random(0,1)
2:  if r < 0.5 do
3:    ILSShift($sol$)
4:  else
5:    ILSSwap($sol$);
```

4.4 Employed Bee Stage

In the traditional ABC algorithm, each employed bee searches the neighborhood of the existing solution to obtain a new solution. If the new solution is better than the current one, the new solution replaces the old one. In our new algorithm, bees use the ILSShift, ILSSwap, or ILSHybrid operator to randomly generate a neighboring solution for each solution in the population. The pseudocode of the employed bee stage is shown in **Algorithm 8**. Unlike the traditional ABC algorithm, the new solution does not immediately replace the old solution but is temporarily stored in X'.

Algorithm 8. Employed_Bee_Stage()
```
1:  $X' = \emptyset$;
2:  for $i = 0$ to $PSize$ do
3:      $sol = X[i]$;
4:      Switch{$OPType$}
5:        Case 0:ILSShift($sol$);
6:        Case 1:ILSSwap($sol$);
7:        Case 2:ILSHybird($sol$);
8:      end switch
9:      $X' = X' \cup \{sol\}$;
10: end for
11: return $X'$;
```

4.5 Onlooker Bee Stage

The pseudocode of the onlooker bee stage shown in Algorithm 9 uses the binary tournament selection rule. First, two solutions are randomly selected in the population. The better of the two solutions is selected. Using the same method as the employed bee, $PSize$ new neighborhood solutions are generated and temporarily stored in X.

Algorithm 9. Onlooker_Bee_Stage()
```
1:  $X'' = \emptyset$;
2:  for $i = 0$ to $PSize$ do
3:      $sol$ = solution selected from $X'$ using to tournament selection;
4:      Switch{ $OPType$ }
5:        Case 0:ILSShift($sol$);
6:        Case 1:ILSSwap($sol$);
7:        Case 2:ILSHybird($sol$);
8:      end switch
9:      $X'' = X'' \cup \{sol\}$;
10: end for
8:  return $X''$;
```

4.6 Local Search

In addition, local search strategies are generally used to further improve the performance of the algorithm. In the local search algorithm shown in **Algorithm 10**, the LocalSearchShift algorithm (shown in Algorithm 11) or the LocalSearchSwap algorithm (shown in Algorithm 12) are randomly selected for neighborhood search.

Algorithm 10. LocalSearch(individual sol)
```
1:  $r$=(double)random(0,1); //Random number between 0 and 1
2:  if $r < 0.5$ do
3:    LocalSearchShift($sol$)
4:  else
5:    LocalSearchSwap($sol$);
```

Algorithm 11 LocalSearchShift(individual sol)
```
1:  for $i = 0$ to 60 do to
2:    $sol' = sol$;
3:    $[sol', C_{max}]$ = Shift($sol'$);
4:    if $sol'$ is better than $sol$ do $sol = sol'$ endif
5:  endfor;
```

Algorithm 12. LocalSearchSwap(individual sol)
```
1:  for $i = 0$ to 60 do to
2:    $sol' = sol$;
3:    $[sol', C_{max}]$ = Swap($sol'$);
4:    if $sol'$ is better than $sol$ do $sol = sol'$ endif
5:  endfor;
```

4.7 Flow Chart for the HABC Algorithm

The specific process of the HABC algorithm is summarized in Fig. 5.

Step 1: We set the parameters, including *PSize* and *OperType*;
Step 2: We initialize the population (denoted as *X*): One initial solution is generated by DNEH, and the other *PSize*-1 initial solutions are randomly generated. The fitness of all solutions is calculated, and the optimal solution is recorded.
Step 3: Employed Bee Stage: The solution generated in the Employed Bee stage is stored in X'.
Step 4: Onlooker Bee Stage: The solution generated in the Onlooker Bee stage is stored in X''.
Step 5: Local search: We perform a local search for the optimal solution in the set $X' \cup X''$.

Step 6: The next generation population is updated. The best *PSize* solutions in $X \cup X' \cup X''$ are updated to the next generation population X.
Step 7: The optimal solution is updated. The best solution in X is used to update the optimal solution.
Step 8: If the stop condition is met, the algorithm ends; otherwise, it returns to step 3.

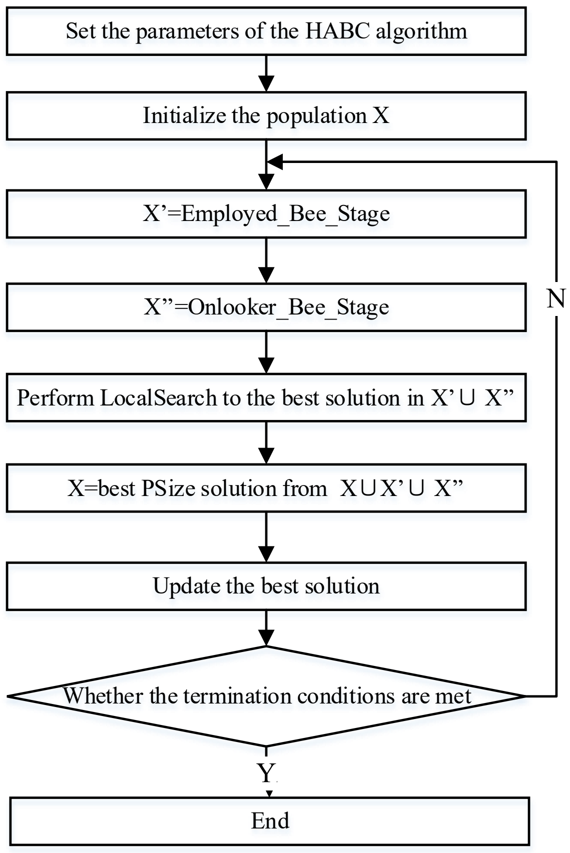

Set the parameters of the HABC algorithm
Initialize the population X
X'=Employed_Bee_Stage
X"=Onlooker_Bee_Stage
Perform LocalSearch to the best solution in X' ∪ X"
X=best PSize solution from X∪X' ∪ X"
Update the best solution
Whether the termination conditions are met
N
Y
End

Fig. 5. The flowchart of the HABC for the PM/DNWPFSP

5. Experimental Calibration

We conducted calibration experiments to adjust the parameters to obtain the optimal performance of the HABC algorithm. The data generation method in the literature (Cheng et al., 2019) was used to generate test instances. A test instance consisted of *n* jobs and *f* factories, with *m* machines in each factory, where $f \in \{2, 4, 6\}$, $n \in \{100, 200, 300, 400, 500\}$ and $m \in \{5, 10, 20\}$. There were 3×5×3=45 combinations. Two instances were generated for each combination. Finally, we obtained a total of 3×5×3×2=90 benchmark instances. All jobs have no waiting time between processing machines. Table 3 lists the detailed parameters of the instances. The HABC algorithm was implemented in C++ with Microsoft Visual Studio 2019 and all optimization flags enabled. All experiments were conducted on a PC running Windows 10 Pro with a quad-core Intel i7-4790 3.6 GHz and 16 GB of memory. For each instance, the experiment was run ten times independently. The relative percentage deviation (RPD) was used to evaluate algorithm performance, which can be obtained using Eq. (19), where C_{max} is generated by a specific algorithm on a particular instance and $C_{max}{}^{*}$ is the best C_{max} on the same instance.

$$RPI = \frac{C_{max} - C_{max}{}^{*}}{C_{max}{}^{*}} \times 100\% \tag{19}$$

Table 3
Parameters of instances.

Number of jobs	n	{100,200,300,400,500}
Number of machines	m	{5,8,10}
Number of factories	f	{2,4,6}
Processing time of job *j* on machine *i*	$P_{i,j}$	[1,100]
Maximum of maintenance level of machine *i*	ML_{max}	[25×*n*/*f*,37.5×*n*/*f*]
The maintenance time on the machine *i*	MT_i	[50,150]

These 90 instances calibrated the proposed HABC algorithm. The HABC involves two parameters that may affect its performance: *OperType* and *Psize*. For these two parameters, we first determined their approximate range based on the literature. Then, we conducted preliminary experiments to determine the optimal value for each parameter. For the HABC algorithm, the two parameters were set as follows: *OperType*: 0, 1 and 2; *Psize*: 2, 3 and 4. Thus, the HABC algorithm had a total of 3×3 = 9 different configurations. We obtained a total of 90×10×9=8100 RPI values. The CPU time, when the HABC algorithm terminates, was $30nm$ ms, where *n* and *m* are the number of jobs and machines, respectively. We used experimental design and analysis of variance (ANOVA) to analyze the experimental results [41]. The ANOVA results are shown in Table 4. We note that in the analysis, the factors *n*, *f*, and *m* were not considered controlled parameters. In this table, the magnitude of the p-value clearly indicates the importance of the corresponding factor. As shown in Table 4, *OperType* and *Psize* cause the response variables to differ statistically significantly; that is, they have a significant impact on the performance of the HABC algorithm.

Table 4
ANOVA table for the experiment on tuning the parameters of HABC

Main effects					
A:*P2Size*	24.2456	2	12.1228	14.10	0.0000
B: *OperType*	2520.35	2	1260.17	1465.81	0.0000
RESIDUAL	6959.34	8095	0.859709		
TOTAL(CORRECTED)	9503.94	8099			

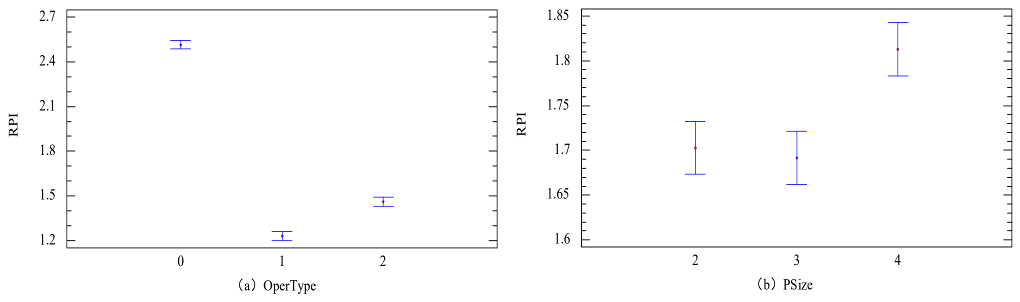


Fig. 6. Means and 95.0% Tukey HSD confidence intervals

Fig. 6 shows the mean plots with 95% Tukey honest significant difference (HSD) confidence intervals. From Fig. 6(a), the optimal value of *OperType* should be 1. Fig. 6(b) shows that *PSize* should be fixed to 3. According to the analysis of the experimental results, the two parameters of the HABC algorithm are set as follows: *OperType* = 1 and *PSize* = 3.

6. Experimental Results

The proposed HABC algorithm was evaluated through a large number of numerical comparisons. In the final experiment, *f* had five values, 100, 200, 300, 400, and 500. There were 3×5×3=45 combinations. Each combination contained five instances, which resulted in a total of 45 × 5=225 instances. All jobs were no-wait. The dataset for the final experiment was generated using the method in Section 5. We compared the proposed HABC with EA (Fernandez-Viagas et al., 2018), IG (Pan, Gao, and Wang et al., 2019), DABC (Pan, Gao, and Wang et al., 2019), and ILS (Pan, Gao, and Wang et al., 2019). All algorithms were coded using C++ with Microsoft Visual Studio 2019. We note that the four competitive algorithms compared did not consider the no-wait constraint, and their initial optimization goal was not the completion time. Therefore, we made necessary modifications to these algorithms to fit the PF/DNWPFSP. In addition, we only made changes to the necessary code to try to maintain all the details of the competitive algorithms to ensure and achieve the performance they should have. The parameters of all competitive algorithms are tabulated in Table 5.

Table 5. Parameter setting for IG, ILS, DABC, and EA.

EA	$\gamma = 5$;
DABC	$PS = 5$; $\Xi = 0$;
ILS	$\tau = 3; a = 0; \varpi = 20; \Xi = 0; \beta = 0.7$;
IG	$d = 7$; $a = 0$; $\beta = 0.6$;

The experiments were performed on the same PC as described in Section 5. For equity, all algorithms used the accelerated operation described in section 3.4. All algorithms ran the same termination time and output results. The termination time was $v*m*n$ ms, where v took the 3 values of 20, 40 and 60, and m and n were the number of machines and number of jobs, respectively. The comparison of the results of all algorithms under these three termination conditions comprehensively demonstrated the performance of all algorithms. All algorithms independently ran five times to solve each of the 225 instances. As with calibration in Section 5, the RPI was used as a performance indicator for comparison.

Tables 6, 7, and 8 show the average RPI (ARPI) values grouped by number of factories, machines, and jobs. Table 6 is the ARPI value obtained by all algorithms at the termination time of $20mn$ ms. The HABC algorithm is superior to the other comparison algorithms. The DABC algorithm ranks second. Furthermore, the ARPI value obtained by the HABC is also significantly better than that of the DABC. In addition, the HABC algorithm has a similar absolute advantage in the comparison of the ARPI grouped by number of jobs, number of machines and number of factories.

Table 6
ARPI value at 20*mn* ms termination time (The optimal ARPI is in bold)

type	DABC	EA	ILS	IG1	HABC
***f*=2**	3.980	4.528	4.207	4.901	**1.711**
***f*=4**	6.728	7.637	7.571	7.706	**2.138**
***f*=6**	8.502	9.459	9.589	9.098	**2.507**
***n*=100**	5.745	6.941	7.592	7.782	**1.695**
***n*=200**	7.663	8.483	8.552	8.521	**2.233**
***n*=300**	7.788	8.511	8.222	8.349	**2.493**
***n*=400**	6.376	7.189	6.655	6.844	**2.301**
***n*=500**	4.444	4.917	4.592	4.679	**1.870**
***m*=5**	7.782	8.660	8.323	8.742	**2.617**
***m*=8**	5.954	6.747	6.686	6.729	**1.975**
***m*=10**	5.473	6.218	6.359	6.234	**1.763**
MEAN	6.403	7.208	7.122	7.235	**2.118**

We used multivariate ANOVA to determine whether the differences in Table 6 were significant. The type of algorithms, number of jobs, number of machines, and number of factories are considered factors. Fig. 7 shows mean plots and 95.0% Tukey HSD intervals of the algorithm type, the interaction and 95.0% Tukey HSD intervals of the algorithm and number of factories, the interaction and 95.0% Tukey HSD intervals of the algorithm and number of jobs, and the interaction and 95.0% Tukey HSD intervals of algorithms and number of machines. Fig. 7 shows that there is a large difference in the ARPI values generated by the five algorithms. Among them, HABC is the best, DABC is second, and the performance gap of the other three comparison algorithms is not too large overall.

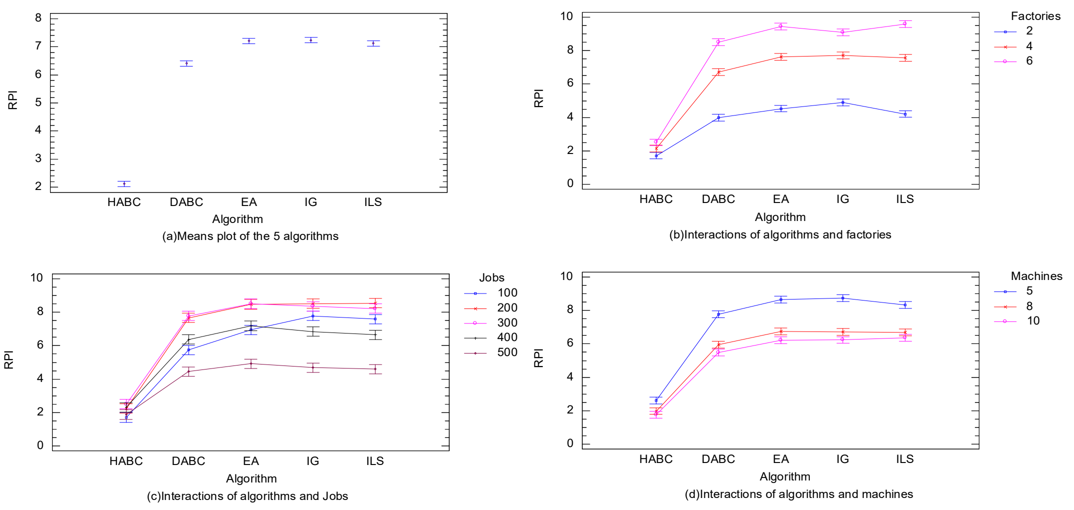


Fig. 7. Means plot, interaction, and 95% Tukey HSD intervals at t=20*mn* ms

Tables 7 and 8 show the ARPI of all algorithms when the termination time is 40*mn* and 60*mn* ms, respectively. Fig. 8 and 10 show the mean plot and interaction plot with termination time *t*= 40*mn* ms and 60*mn* ms, respectively. As seen from the tables and the figures, the ranking of all algorithms remains the same, the performance of the HABC algorithm is still much better than that of the other comparison algorithms, and the ARPI difference is still obvious.

Table 7
ARPI value at 40*mn* ms termination time (The optimal ARPI is in bold)

type	DABC	EA	ILS	IG	HABC
***f*=2**	3.207	3.825	3.606	4.314	**1.044**
***f*=4**	5.589	6.492	6.756	6.908	**1.156**
***f*=6**	7.344	8.347	8.769	8.204	**1.308**
***n*=100**	4.549	5.731	6.699	6.998	**1.215**
***n*=200**	6.451	7.437	7.809	7.763	**1.171**
***n*=300**	6.685	7.496	7.460	7.519	**1.244**
***n*=400**	5.421	6.177	5.906	6.074	**1.185**
***n*=500**	3.793	4.266	4.011	4.023	**1.032**
***m*=5**	6.647	7.567	7.394	7.908	**1.429**
***m*=8**	4.956	5.767	6.002	6.014	**1.079**
***m*=10**	4.536	5.331	5.735	5.505	**1.001**
MEAN	5.380	6.221	6.377	6.475	**1.169**

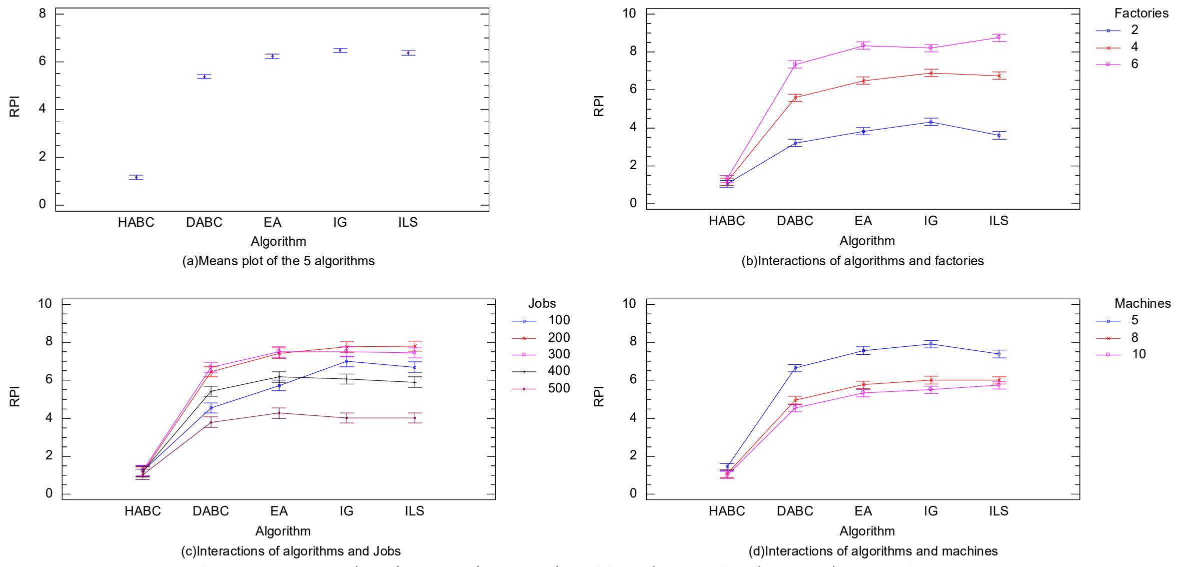


Fig. 8. Means plot, interaction, and 95% Tukey HSD intervals at *t*=40*mn* ms

Table 8
ARPI value at 60*mn* ms termination time (The optimal ARPI is in bold)

type	DABC	EA	ILS	IG	HABC
***f*=2**	2.762	3.396	3.257	3.944	**0.694**
***f*=4**	4.956	5.870	6.290	6.474	**0.638**
***f*=6**	6.673	7.615	8.372	7.721	**0.702**
***n*=100**	3.824	5.051	6.225	6.573	**0.984**
***n*=200**	5.820	6.850	7.391	7.280	**0.713**
***n*=300**	6.059	6.851	7.044	7.056	**0.589**
***n*=400**	4.885	5.582	5.536	5.636	**0.592**
***n*=500**	3.396	3.802	3.669	3.688	**0.512**
***m*=5**	5.996	6.865	6.879	7.473	**0.813**
***m*=8**	4.388	5.219	5.635	5.583	**0.605**
***m*=10**	4.006	4.797	5.405	5.083	**0.616**
MEAN	4.797	5.627	5.973	6.046	**0.678**

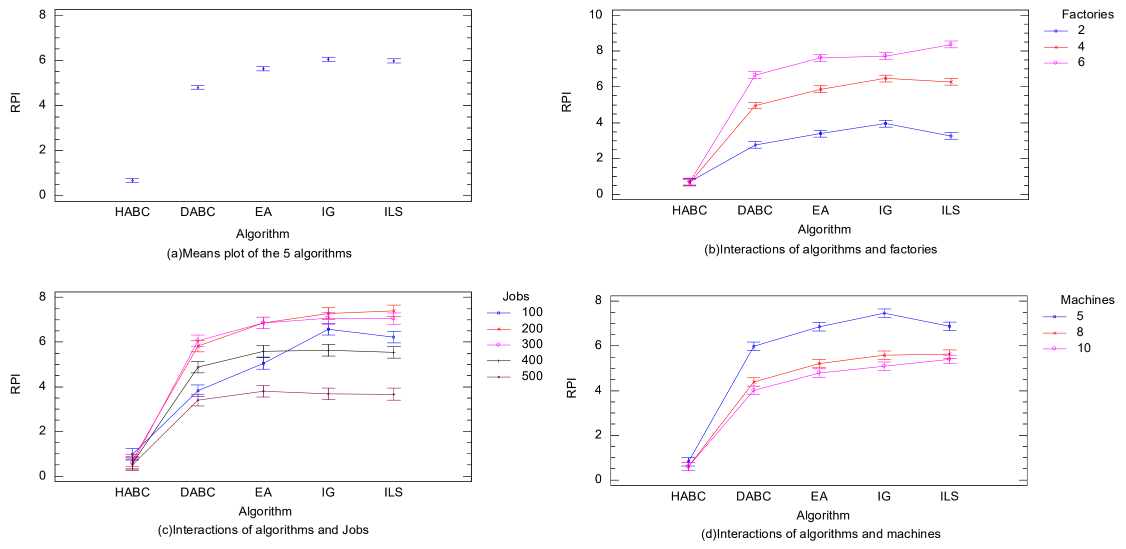


Fig. 9. Means plot, interaction, and 95% Tukey HSD intervals at t=60*mn* ms

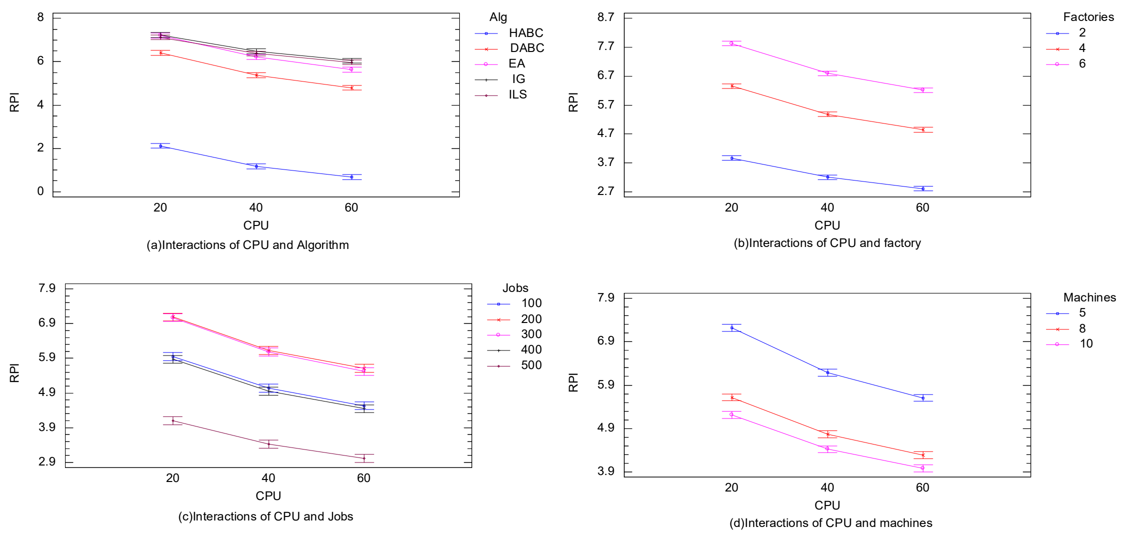


Fig. 10. Interactions of CPU time and type of algorithm, number of factories, number of jobs and number of machines, and 95% Tukey HSD intervals

Fig. 10 shows the performance comparison of all algorithms at different termination times. As shown, with increasing termination time, all algorithms can obtain better solutions. Finally, it can be concluded with certainty that the proposed HABC is the best algorithm for solving the PM/DNWPFSP with the completion time criterion.

7. Conclusions

In this paper, a hybrid artificial bee colony (HABC) algorithm with an iterated local search mechanism is proposed to solve distributed no-wait permutation flowshop scheduling problems with preventive maintenance (PM/DNWPFSP). First, the characteristics of the PM/DNWPFSP are analyzed, and a location-based mathematical model is established. An acceleration method is proposed to calculate the completion time for the PM/DNWPFSP. The HABC algorithm uses the framework of an artificial bee colony with an iterated local search mechanism for a neighborhood search, which greatly improves the quality of the searched solution. Finally, the performance of the HABC is verified by a large number of numerical experiments. The results show that the HABC algorithm has better performance than that of the four most recent algorithms. In addition, the proposed HABC algorithm can obtain high-quality feasible solutions under different stopping conditions.

In future research, the characteristics of the problem should be further studied, and more effective evolutionary operators and metaheuristic algorithms should be developed for the distributed permutation flowshop scheduling problem. In terms of the algorithm, follow-up work should be combined with other algorithms to improve the search efficiency. In addition, the problem can be extended to distributed flexible flowshops. We will continue to study other single objectives in the future.

Acknowledgments

This research is partially supported by the National Natural Science Foundation of China 52205529, and the Natural Science Foundation of Shandong Province (ZR2021QE195)

References

Aldowaisan, T. & Allahverdi, A. (2004). New heuristics for m-machine no-wait flowshop to minimize total completion time. *Omega-International Journal of Management Science, 32*(5), 345-352.

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research, 246*(2), 345-378.

Allahverdi, A., Ng, C.T., Cheng, T.C.E. & Kovalyov, M.Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research, 187*(3), 985-1032.

Allali, K., Aqil, S. & Belabid, J. (2022). Distributed no-wait flow shop problem with sequence dependent setup time: Optimization of makespan and maximum tardiness. *Simulation Modelling Practice and Theory, 116*, 102455.

Cheng, C., Ying, K., Chen, H. & Lu, H. (2019). Minimising makespan in distributed mixed no-idle flowshops. *International Journal of Production Research, 57*(1), 48-60.

Fernandez-Viagas, V. & Framinan, J.M. (2014). A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research, 53*(4), 1111-1123.

Fernandez-Viagas, V., Perez-Gonzalez, P. & Framinan, J.M. (2018). The distributed permutation flow shop to minimise the total flowtime. *Computers & Industrial Engineering, 118*, 464-477.

Fitouhi, M. & Nourelfath, M. (2012). Integrating noncyclical preventive maintenance scheduling and production planning for a single machine. *International Journal of Production Economics, 136*(2), 344-351.

Gao, J., Chen, R. & Deng, W. (2013). An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research, 51*(3), 641-651.

Gao, K., Pan, Q. & Li, J. (2011). Discrete harmony search algorithm for the no-wait flow shop scheduling problem with total flow time criterion. *The International Journal of Advanced Manufacturing Technology, 56*(5-8), 683-692.

Han, X., Han, Y., Zhang, B., Qin, H., Li, J., Liu, Y. & Gong, D. (2022). An effective iterative greedy algorithm for distributed blocking flowshop scheduling problem with balanced energy costs criterion. *Applied Soft Computing, 129*,109502.

Hans, R.C. (1984). The Three-Machine No-Wait Flow Shop Is NP-Complete. *Journal of the Acm, 31*, 336-345.

Jia, H.Z., Fuh, J.Y.H., Nee, A.Y.C. & Zhang, Y.F. (2007). Integration of genetic algorithm and Gantt chart for job shop scheduling in distributed manufacturing systems. *Computers & Industrial Engineering, 53*(2), 313-320.

Jing, X., Pan, Q. & Gao, L. (2021). Local search-based metaheuristics for the robust distributed permutation flowshop problem. *Applied Soft Computing, 105*, 107247.

Lei, D. & Liu, M. (2020). An artificial bee colony with division for distributed unrelated parallel machine scheduling with preventive maintenance. *Computers & Industrial Engineering, 141*, 106320.

Li, H., Li, X. & Gao, L. (2021). A discrete artificial bee colony algorithm for the distributed heterogeneous no-wait flowshop scheduling problem. *Applied Soft Computing, 100*, 106946.

Li, Y., Pan, Q., Li, J., Gao, L. & Tasgetiren, M.F. (2021). An Adaptive Iterated Greedy algorithm for distributed mixed no-idle permutation flowshop scheduling problems. *Swarm and Evolutionary Computation, 63*, 100874.

Li, Y., Pan, Q., Ruiz, R. & Sang, H. (2022). A referenced iterated greedy algorithm for the distributed assembly mixed no-idle permutation flowshop scheduling problem with the total tardiness criterion. *Knowledge-Based Systems, 239*(3), 108036.

Lu, C., Liu, Q., Zhang, B. & Yin, L. (2022). A Pareto-based hybrid iterated greedy algorithm for energy-efficient scheduling of distributed hybrid flowshop. *Expert Systems with Applications, 204*, 117555.

Mao, J., Pan, Q., Miao, Z. & Gao, L. (2021). An effective multi-start iterated greedy algorithm to minimize makespan for the distributed permutation flowshop scheduling problem with preventive maintenance. *Expert Systems with Applications, 169*(5).

Mao, J., Pan, Q., Miao, Z., Gao, L. & Chen, S. (2022). A hash map-based memetic algorithm for the distributed permutation flowshop scheduling problem with preventive maintenance to minimize total flowtime. *Knowledge-Based Systems, 242*(4), 108413.

Meng, K., Tang, Q., Cheng, L. & Zhang, Z. (2022). Mixed-model assembly line balancing problem considering preventive maintenance scenarios: MILP model and cooperative co-evolutionary algorithm. *Applied Soft Computing, 127*,109341.

Meng, L., Gao, K., Ren, Y., Zhang, B., Sang, H. & Chaoyong, Z. (2022). Novel MILP and CP models for distributed hybrid flowshop scheduling problem with sequence-dependent setup times. *Swarm and Evolutionary Computation, 71,* 101058.

Miyata, H.H. & Nagano, M.S. (2021). Optimizing distributed no-wait flow shop scheduling problem with setup times and maintenance operations via iterated greedy algorithm. *Journal of Manufacturing Systems, 61*, 592-612.

Miyata, H.H., Nagano, M.S. & Gupta, J.N.D. (2019). Integrating preventive maintenance activities to the no-wait flow shop scheduling problem with dependent-sequence setup times and makespan minimization. *Computers & Industrial Engineering, 135*,79-104.

Naderi, B. & Ruiz, R. (2010). The distributed permutation flowshop scheduling problem. *Computers & Operations Research, 37*(4), 754-768.

Naderi, B., Zandieh, M. & Aminnayeri, M. (2011). Incorporating periodic preventive maintenance into flexible flowshop scheduling problems. *Applied Soft Computing, 11*(2), 2094-2101.

Pan, E., Liao, W. & Xi, L. (2010). Single-machine-based production scheduling model integrated preventive maintenance planning. *The International Journal of Advanced Manufacturing Technology, 50*(1), 365-375.

Pan, Q., Gao, L., Li, X. & Jose, F.M. (2019). Effective constructive heuristics and meta-heuristics for the distributed assembly permutation flowshop scheduling problem. *Applied Soft Computing, 81,*105492.

Pan, Q., Gao, L., Wang, L., Liang, J. & Li, X. (2019). Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem. *Expert Systems with Applications, 124,* 309-324.

Pan, Q., Wang, L. & Zhao, B. (2008). An improved iterated greedy algorithm for the no-wait flow shop scheduling problem with makespan criterion. *The International Journal of Advanced Manufacturing Technology, 38*(7), 778-786.

Pan, Q., Zhao, B. & Qu, Y. (2008). Heuristics for the No-Wait Flow Shop Problem with Makespan Criterion. *Chinese Journal of Computers, 31*,1147-1154.

Pei, Z., Zhang, X., Zheng, L. & Wan, M. (2019). A column generation-based approach for proportionate flexible two-stage no-wait job shop scheduling. *International Journal of Production Research, 58,* 487-508.

Rossi, F.L. & Nagano, M.S. (2021). Heuristics and iterated greedy algorithms for the distributed mixed no-idle flowshop with sequence-dependent setup times. *Computers & Industrial Engineering, 157*,107337.

Ruiz, R., Carlos García-Díaz, J. & Maroto, C. (2007). Considering scheduling and preventive maintenance in the flowshop sequencing problem. *Computers & Operations Research, 34*(11), 3314-3330.

Shao, W., Shao, Z. & Pi, D. (2021). Effective constructive heuristics for distributed no-wait flexible flow shop scheduling problem. *Computers & Operations Research, 136*, 105482.

Tseng, L. & Lin, Y. (2010). A hybrid genetic algorithm for no-wait flowshop scheduling problem. *International Journal of Production Economics, 128*(1),144-152.

Wang, H. (2002). A survey of maintenance policies of deteriorating systems. *European Journal of Operational Research, 139*(3), 469-489.

Wang, S., Wang, L., Liu, M. & Xu, Y. (2013). An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem. *International Journal of Production Economics, 145*(1), 387-396.

Yang, S., Wang, J. & Xu, Z. (2022). Real-time scheduling for distributed permutation flowshops with dynamic job arrivals using deep reinforcement learning. *Advanced Engineering Informatics, 54*, 101776.

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.

Ying, K. & Lin, S. (2020). Solving no-wait job-shop scheduling problems using a multi-start simulated annealing with bi-directional shift timetabling algorithm. *Computers & Industrial Engineering, 146,* 106615.

Ying, K., Lin, S., Cheng, C. & He, C. (2017). Iterated reference greedy algorithm for solving distributed no-idle permutation flowshop scheduling problems. *Computers & Industrial Engineering, 110*, 413-423.

Yu, Y., Zhang, F., Yang, G., Wang, Y., Huang, J. & Han, Y. (2022). A discrete artificial bee colony method based on variable neighborhood structures for the distributed permutation flowshop problem with sequence-dependent setup times. *Swarm and Evolutionary Computation, 75,* 101179.

Zhu, N.N., Zhao, F.Q., Wang, L., Ding, R.Q., Xu, T.P. & Jonrinaldi. (2022). A discrete learning fruit fly algorithm based on knowledge for the distributed no-wait flow shop scheduling with due windows. *Expert Systems with Applications, 198*.