COMPLEX SYSTEM MODELING AND SIMULATION
ISSN 2096-9929 02/06 pp 282−306
Volume 3, Number 4, December 2023
DOI: 10.23919/CSMS.2023.0016

Intelligent Optimization Under Multiple Factories: Hybrid Flow Shop Scheduling Problem with Blocking Constraints Using an Advanced Iterated Greedy Algorithm

Yong Wang, Yuting Wang*, Yuyan Han*, Junqing Li, Kaizhou Gao, and Yusuke Nojima

Abstract: The distributed hybrid flow shop scheduling problem (DHFSP), which integrates distributed manufacturing models with parallel machines, has gained significant attention. However, in actual scheduling, some adjacent machines do not have buffers between them, resulting in blocking. This paper focuses on addressing the DHFSP with blocking constraints (DBHFSP) based on the actual production conditions. To solve DBHFSP, we construct a mixed integer linear programming (MILP) model for DBHFSP and validate its correctness using the Gurobi solver. Then, an advanced iterated greedy (AIG) algorithm is designed to minimize the makespan, in which we modify the Nawaz, Enscore, and Ham (NEH) heuristic to solve blocking constraints. To balance the global and local search capabilities of AIG, two effective inter-factory neighborhood search strategies and a swap-based local search strategy are designed. Additionally, each factory is mutually independent, and the movement within one factory does not affect the others. In view of this, we specifically designed a memory-based decoding method for insertion operations to reduce the computation time of the objective. Finally, two shaking strategies are incorporated into the algorithm to mitigate premature convergence. Five advanced algorithms are used to conduct comparative experiments with AIG on 80 test instances, and experimental results illustrate that the makespan and the relative percentage increase (RPI) obtained by AIG are 1.0% and 86.1%, respectively, better than the comparative algorithms.

Key words: blocking; distributed hybrid flow shop; neighborhood search; iterated greedy algorithm

1 Introduction

With the increased market competition, the traditional single-factory production model is no longer sufficient to meet market demands. Instead, a novel distributed manufacturing model has received widespread attention[1]. This decentralized manufacturing model offers many advantages such as improving efficiency, reducing costs, and shortening manufacturing cycle times. Due to the fact that the distributed permutation flow shop scheduling problem (DPFSP) simultaneously considers factories allocation and job arrangement, that is, jobs first are allocated for different factories and then scheduled within each factory in a reasonable manner, the DPFSP is significantly more complex than

• Yong Wang, Yuting Wang, and Yuyan Han are with the School of Computer Science, Liaocheng University, Liaocheng 252000, China. E-mail: wy_sulis@163.com; wangyuting@lcu-cs.com; hanyuyan@lcu-cs.com.
• Junqing Li is with the School of Computer Science, Shandong Normal University, Jinan 252000, China. E-mail: lijunqing@lcu-cs.com.
• Kaizhou Gao is with the Macau Institute of Systems Engineering, Macau University of Science and Technology, Macau 999078, China. E-mail: gaokaizhou@lcu-cs.com.
• Yusuke Nojima is with the Department of Computer Science and Intelligent Systems, Osaka Prefecture University, Osaka 599-8531, Japan. E-mail: nojima@cs.osakafu-u.ac.jp.
∗ To whom correspondence should be addressed.

※ This article was recommended by Associate Editor Xinyu Li.
Manuscript received: 2023-06-07; revised: 2023-07-04; accepted: 2023-07-12

the general permutation flow shop scheduling problem (PFSP)[2].

Additionally, to adjust variations in production speed and enhance productivity, some factories have integrated parallel machines into their production processes. This production model gives rise to a hybrid flow shop scheduling problem (HFSP). By incorporating parallel machines, HFSP allows for the simultaneous processing of multiple jobs, resulting in a significant boost in productivity, and is widely used in steelmaking, electronic production, the chemical industry, etc.[3–5] Given the increasing collaboration between enterprises and the trend towards intelligent manufacturing, the advantages of HFSP and DPFSP have been integrated to create a new scheduling problem called the distributed HFSP (DHFSP). Undoubtedly, DHFSP holds greater practical significance[6].

The DHFSP can be approximated as parallel factory scheduling, where each factory represents an HFSP. Recently, some intelligent optimization methods have been employed to solve DHFSP, i.e., multi-neighborhood iterated greedy algorithm[6], artificial bee colony (ABC) algorithm[7], bi-population cooperative memetic algorithm[8], and so on. However, in some production scenarios, jobs may be blocked on a machine because of cost or storage space limitations, impeding their transition to the next stage of processing. Consequently, DHFSP has been extended to a more complex variant known as the DHFSP with blocking constraints (DBHFSP). DBHFSP needs to solve the following issues: allocating a factory, assigning a suitable machine for each job, and scheduling the sequence of the jobs. Notably, DBHFSP is significantly more complex than DHFSP, more closely aligned with actual production processes, and hence more deserving of our attention and study.

In practical production scenarios, the occurrence of a blocking state can have significant negative consequences on the completion time and overall productivity. Therefore, it is crucial to thoroughly analyze the challenges and risks associated with the DBHFSP before devising suitable strategies. Some of the potential challenges should be considered: (1) Blocked jobs need to wait until the machine is idle before they can continue processing, which causes additional delays and affects the completion time of the jobs. (2) Poor job allocation strategies may create imbalances between factories, resulting in more instances of blocking. (3) Job blocking can increase the risk of the solution getting stuck in a local optimum, making it necessary to develop effective strategies to minimize blocking time.

Based on the above motivations, we know that it is crucial to select an appropriate algorithm and do some adjustments tailored to the properties of the DBHFSP. Through comparisons with various intelligent optimization algorithms[7–11], we observe that the iterated greedy algorithm (IGA) demonstrates superiority in some scheduling problems[12–14]. The structure of IGA is simple and easy to be implemented. Therefore, we develop an advanced IGA (AIG) method to minimize the makespan by reducing the blocking time. The contributions of our work are listed.

(1) We build a mixed integer linear programming (MILP) model for DBHFSP and use the Gurobi solver to prove its validity. Under the search framework of IGA, we design the AIG algorithm based on problem characteristics and provide the optimization gap for MILP and AIG.

(2) Two inter-factory neighborhood search strategies and a swap-based local search strategy are proposed to balance the global and local search capabilities of AIG, respectively.

(3) To prevent premature convergence, two shaking strategies are proposed.

(4) Extensive simulation experiments are conducted on 80 test instances and compared against five optimization algorithms to showcase the superiority of AIG in optimizing DBHFSP.

The paper is structured as follows. Section 2 offers a review of previous studies on the subject. Section 3 outlines the mathematical model of DBHFSP and presents a simple example for better understanding. Section 4 provides detailed information about the AIG algorithm. The experimental results and their analysis are presented in Section 5. Finally, Section 6 provides a summary and suggests potential directions for future research.

2 Literature Review

According to our best knowledge, the DBHFSP is rarely reported in the literature. Consequently, we conduct a comprehensive review of the most relevant research work in the literature, including the DPFSP, HFSP, DHFSP, and IGA.

2.1 Distributed permutation flow shop scheduling problem

DPFSP has been the subject of much research in the manufacturing industry. A hybrid estimation of distribution algorithm (EDA) was proposed, which combines the advantages of the memetic algorithm to solve DPFSP[15]. To accelerate the iteration speed of IGA, Ruiz et al.[13] designed an enhanced IGA to optimize the DPFSP, resulting in improving computational efficiency and enhancing solution quality. Moreover, Wang et al.[16] designed a hybrid discrete cuckoo search algorithm to address DPFSP, which has been demonstrated to be effective in optimizing this problem. Shao et al.[17] made notable contributions to optimizing the makespan in the DPFSP by developing three advanced iterated greedy algorithms that leverage the problem characteristics. For distributed assembly no-idle PFSP, a cooperative water wave optimization method was proposed to optimize assembly completion time[18]. Reference [19] focused on the energy-aware scheduling problem and proposed a cooperative memetic algorithm to address DPFSP. To reduce energy consumption and makespan in distributed no-wait PFSP, Zhao et al.[20] developed a Q-learning driven cooperative metaheuristic algorithm. The aforementioned studies have been effective in solving the distributed scheduling problem. However, they did not consider the presence of parallel machines. Incorporating parallel machines can significantly increase production speed by enabling multiple jobs to be processed simultaneously.

2.2 Hybrid flow shop scheduling problem

HFSP is a composite problem that encompasses both the parallel machine and the PFSP. The solution to HFSP involves two sub-problems: assigning the available machines to the jobs and sequencing the jobs. To optimize the HFSP with makespan as the objective, Pan et al.[21] designed a discrete ABC algorithm, which has been demonstrated to be effective in several studies. To optimize the HFSP on identical parallel machines, Wang et al.[22] developed an enhanced EDA, which combines local and global search strategies to improve the solution quality. To minimize energy consumption for green production, Li et al.[23] developed a two-level imperialistic competitive algorithm to optimize HFSP. Considering the impact of the human factor in HFSP, Marichelvam et al.[9] applied an enhanced particle swarm optimization (PSO) algorithm and proved its effectiveness experimentally. In addition, to cope with real-time scheduling in manufacturing systems, Wu et al.[24] proposed a gene expression programming real-time scheduling method to solve the real-time HFSP. While the studies mentioned above have effectively addressed the HFSP problem, they did not consider the blocking constraints and the manufacturing mode of multi-factory processing. However, with the increasing demand for multi-factory production in the market, it has become an inevitable trend to consider these factors in the optimization process.

2.3 Distributed hybrid flow shop scheduling problem

Due to the rapid economic development, HFSP within a single factory is no longer sufficient to meet the production demands of modern enterprises, which leads to the emergence of DHFSP. However, DHFSP is more challenging than HFSP. We note the following studies. To tackle the challenging DHFSP with multiprocessor tasks, an improved IGA was proposed by Ying and Lin[25]. Shao et al.[6] modeled DHFSP according to its characteristics and proposed a multi-neighborhood IGA. Xi and Lei[26] studied fuzzy distributed two-stage HFSP and introduced a Q-learning-based teaching-learning optimization method for makespan optimization. For multi-objective fuzzy DHFSP with due date, a co-evolutionary algorithm combining EDA and IGA features was developed by Zheng et al.[27] Jiang et al.[28] studied DHFSP with multiprocessor tasks and designed a decomposition-based multi-objective optimization method. For energy-aware DHFSP optimization, a reinforcement learning strategy and a cooperative memetic algorithm were proposed[29]. Similarly, to reduce energy consumption, Pan et al.[30] studied the scheduling problem of distributed energy-efficient parallel machines and proposed a knowledge-based two-population optimization algorithm. Shao et al.[31] investigated DHFSP with energy and labor-aware and proposed a hybrid memetic algorithm by combining the advantages of network optimization and memetic algorithm. The above research effectively solved the problems of parallel machines and multiple factory processing. However, they did not consider the blocking constraint. In the real world, there are many applications for train track scheduling[32], ship manufacturing[33], and concrete blocks[34]. It is important to develop new algorithms and strategies that can handle these additional constraints and complexities to further improve the efficiency and effectiveness of production systems in various industries.

We note the following studies on blocking constraints. Han et al.[35] extended the PFSP model to include blocking constraints and setup time and designed a discrete multi-objective optimization (DEMO) algorithm to optimize energy consumption. To optimize the setup time and the blocking constraint of HFSP, Aqil and Allali[36] presented two meta-heuristic algorithms. Han et al.[37] embedded the learning mechanism into the IGA and studied the DPFSP with blocking constraints. Shao et al.[1] analyzed the properties of distributed fuzzy PFSP with blocking constraints and designed some effective heuristic and meta-heuristic methods. Aiming at the blocking constraint and energy consumption in HFSP, an enhanced IGA was employed to optimize HFSP[38]. Recently, Zhao et al.[39, 40] focused on multi-objective distributed blocking PFSP and proposed two multi-objective algorithms: Pareto-based discrete Jaya algorithm and hyper-heuristic with Q-learning algorithm. The above studies demonstrate that blocking constraints have been explored by researchers and highlight the significance of considering blocking constraints in scheduling problems. Therefore, it is important to investigate the DHFSP with blocking to address real-world challenges and improve production efficiency.

2.4 Iterated greedy algorithm

IGA has been widely employed in PFSP due to its simplicity and efficient performance[12]. Fernandez-Viagas et al.[41] investigated PFSP with total tardiness as the optimization objective and designed eight different variants of IGA to address this problemx. Wang et al.[42] focused on the PFSP with mixed no-wait constraint and developed an improved IGA to adapt to the problem characteristics. To solve DPFSP, Han et al.[43] designed an effective IGA based on single-job and job-block swapping strategies. To solve the DPFSP with blocking constraints (DBFSP), Chen et al.[44] introduced a novel population-based IG (PBIG) that integrates the strengths of population search methods and IGA. Qin et al.[38] improved the IGA by proposing a special substitution strategy to handle HFSP with blocking constraints. In addition, for DHFSP, Qin et al.[45] analyzed the influence of heterogeneous factories and blocking constraints and designed a collaborative IGA (CIG) to address DHFSP.

According to the above literature, IGA shows good performance in solving the flow shop scheduling problem. (1) As mentioned in Ref. [43], compared with other swarm intelligence algorithms, IGA is a very simple and easily understandable algorithm with strong local search ability. (2) Reference [44] took full advantage of the extensibility of IGA, added a population-based search method to it, and proposed a PBIG to solve DBFSP. The test of 720 instances proves that PBIG is superior to the existing algorithms. (3) Recently, a collaborative IGA was designed in Ref. [45] based on the characteristics of DHFSP, and the performance of the algorithm was improved by the coevolution of two solutions. By comparing it with five other advanced algorithms, the effectiveness of CIG was validated. Based on the above analysis, we select IGA to solve DBHFSP.

Although IGA has good performance in solving scheduling problems, it also has some limitations, such as weak global search ability, poor solution diversity, and a tendency to converge prematurely. To overcome its limitations and improve its performance for DBHFSP, a customized strategy will be designed based on the problem characteristics. The local search ability of the IGA is strong, but the diversity is poor. Therefore, we design two inter-factory neighborhood search strategies, which can enhance the collaboration between factories. Additionally, we design a swap-based local strengthening strategy that enables the algorithm to explore the local neighborhood more thoroughly. Finally, we also propose two shaking strategies that help to prevent premature convergence.

3 Problem Statement

3.1 Problem definition

DBHFSP is described as follows. A job set containing J jobs is to be assigned to F factories for processing. For any Factory $f\,(f=1,2,\ldots,F)$, there are S identical processing stages, each processing stage s contains $M_s\,(M_s \geqslant 1)$ parallel machines, and the number of parallel machines in at least one stage is to be greater than or equal to 2. In addition, there is no intermediate buffer between any adjacent stages. To better visualize the multi-factory production process, an example of a two-factory production is given in Fig. 1. Factory 1 and Factory 2 are the same and have the same machines in each processing stage, and the job can be processed in any factory. We use $p_{j,s}$ to denote the processing time of job j at stage s.

Otherwise, the discussion of DBHFSP is based on the following assumptions:

(1) At time 0, all jobs and machines are available for

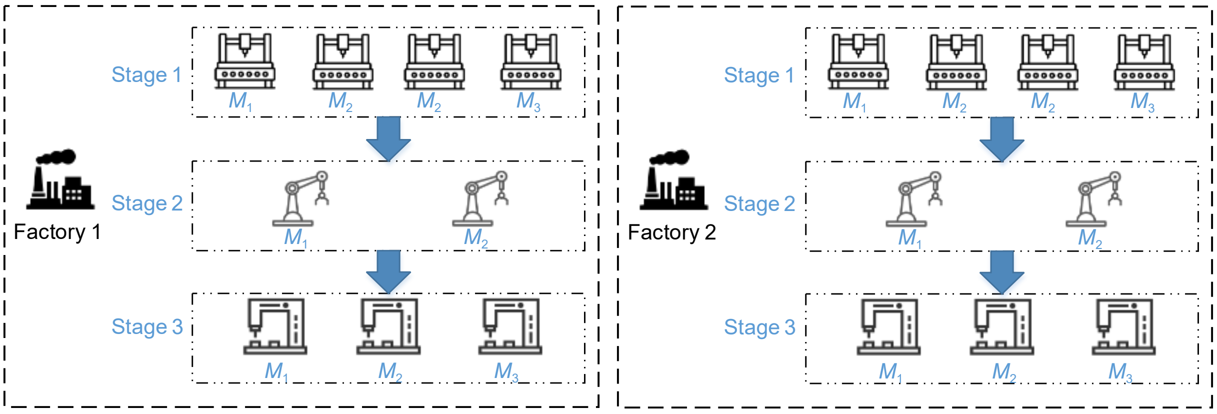

Fig. 1 Example of isomorphic factories.

processing.

(2) Once a job is allocated for a factory, it cannot be changed to a different factory midway.

(3) Each machine can only process one job at a time, and each job can only be processed by one machine at a time.

(4) All jobs should be processed continuously without any interruptions or preemptions.

(5) Each job follows a predetermined sequence of processing stages and cannot skip any stage or finish early.

(6) There exist no intermediate buffers between adjacent stages.

DBHFSP involves three interconnected sub-problems: determining the assignment of each job to a specific factory, assigning a machine for each job in each factory, and sequencing the jobs in each machine. And the mathematical model of DBHFSP is given based on the above assumptions and Ref. [46].

3.2 Mathematical model

Notations:

J: Number of jobs.

F: Number of factories.

S: Number of stages in each factory.

j: Index of jobs, $j \in \{1,2,\ldots,J\}$.

f: Index of factories, $f \in \{1,2,\ldots,F\}$.

s: Index of stages, $s \in \{1,2,\ldots,S\}$.

$M_{f,s}$: Number of machines at stage s in Factory f.

m: Index of machines at stage s, $m \in \{1,2,\ldots,M_{f,s}\}$.

L: A big positive integer.

$p_{j,s}$: Processing time of job j at stage s.

Decision variables:

$C_{\max}$: Makespan.

$C_{j,s}$: Completion time of job j at stage s.

$D_{j,s}$: Departure time of job j at stage s. It refers to the time that the job leaves stage s after finishing processing.

$x_{f,j}$: Decision variables, 1, if job j is processed in Factory f; 0, otherwise.

$y_{f,s,j,m}$: Decision variables, 1, if job j is processed on machine m at stage s in Factory f; 0, otherwise.

$z_{f,s,j,j'}$: Decision variables, 1, if job j is at any position before job j' at stage s in Factory f; 0, otherwise.

Constraints:

$$\text{Minimize } C_{\max} \tag{1}$$

$$\sum_{f=1}^{F} x_{f,j} = 1, \forall j \in \{1,2,\ldots,J\} \tag{2}$$

$$\sum_{m=1}^{M_{f,s}} y_{f,s,j,m} = x_{f,j},\ \forall f \in \{1,2,\ldots,F\},\\ \forall j \in \{1,2,\ldots,J\}, \forall s \in \{1,2,\ldots,S\} \tag{3}$$

$$z_{f,s,j,j'} + z_{f,s,j',j} \leqslant 1, \forall f \in \{1,2,\ldots,F\},\\ \forall s \in \{1,2,\ldots,S\},\ \forall j,j' \in \{1,2,\ldots,J\},\ j<j' \tag{4}$$

$$z_{f,s,j,j'} + z_{f,s,j',j} \geqslant y_{f,s,j,m} + y_{f,s,j',m} - 1,\\ \forall f \in \{1,2,\ldots,F\}, \forall s \in \{1,2,\ldots,S\},\\ \forall j,j' \in \{1,2,\ldots,J\},\ j<j', \forall m \in \{1,2,\ldots,M_{f,s}\} \tag{5}$$

$$C_{j,s} \geqslant p_{j,s}, \forall j \in \{1,2,\ldots,J\}, \forall s \in \{1,2,\ldots,S\} \tag{6}$$

$$C_{j',s} \geqslant D_{j,s} + p_{j',s} + \left(y_{f,s,j,m} + y_{f,s,j',m} + z_{f,s,j,j'} - 3\right) \times\\ L, \forall j,j' \in \{1,2,\ldots,J\},\ j \neq j', \forall f \in \{1,2,\ldots,F\},\\ \forall s \in \{1,2,\ldots,S\}, \forall m \in \{1,2,\ldots,M_{f,s}\} \tag{7}$$

$$C_{j,s+1} = D_{j,s} + p_{j,s+1}, \forall j \in \{1,2,\ldots,J\}, \forall s \in \{1,2,\ldots,S-1\} \tag{8}$$

$$D_{j,s} \geqslant C_{j,s}, \forall j \in \{1,2,\ldots,J\}, \forall s \in \{1,2,\ldots,S\} \tag{9}$$

$$C_{\max} \geqslant D_{j,S}, \forall j \in \{1,2,\ldots,J\} \qquad (10)$$

Formula (1) is the makespan objective. Constraint (2) ensures that each job is allocated for only one factory. For Factory f, Constraint (3) enforces that a job is allocated to only one machine at a given stage. In Factory f, Constraint (4) guarantees only one of $z_{f,s,j,j'}$ and $z_{f,s,j',j}$ can be 1 if j and j' are processed at the same stage. Constraint (5) guarantees that if both jobs j and j' are processed on the same machine at stage s in Factory f, either $z_{f,s,j,j'}$ or $z_{f,s,j',j}$ must be 1 and the other must be 0. Constraint (6) enforces that the processing time of job j at stage s is smaller than its completion time. Constraint (7) ensures that there exists no overlap in the processing of jobs on the same machine at the same stage. Constraint (8) states that the completion time of job j at stage s is equal to the processing time plus its departure time at the previous stage. Constraint (9) ensures that no job j leaves stage s before it is completed. Constraint (10) guarantees that the makespan is not smaller than the time required for all jobs to complete the processing.

3.3 Example

We give an example to illustrate the problem. Let $F=2$, $S=2$, $M_{1,1}=2$, $M_{1,2}=2$, $M_{2,1}=2$, $M_{2,2}=2$, and $J=6$. Table 1 gives the processing time of the job at each stage. Processing time uses integers (1, 2, 3, ...) to represent the length of time without units. We use the Gurobi solver to run this small-scale instance and find the smallest makespan. The optimal solution is $x_{1,3}=x_{1,4}=x_{1,6}=x_{2,1}=x_{2,2}=x_{2,5}=1$, $y_{1,1,3,2}=y_{1,1,4,1}=y_{1,1,6,1}=y_{1,2,3,2}=y_{1,2,4,2}=y_{1,2,6,1}=y_{2,1,1,2}=y_{2,1,2,1}=y_{2,1,5,1}=y_{2,2,1,1}=y_{2,2,2,2}=y_{2,2,5,2}=1$, $z_{1,1,6,4}=z_{1,2,3,4}=z_{2,1,5,2}=z_{2,2,5,2}=1$. Therefore, the job scheduling sequences in Factories 1 and 2 are $\{6,3,4\}$ and $\{5,2,1\}$, respectively. The makespan is 38. The running result is shown in Fig. 2, where $s_a m_b$ represents the b-th machine on the a-th stage. The gray part represents the blocking time.

Table 1 Processing time for six jobs.

Job No.	Processing time	
	Stage 1	Stage 2
1	18	20
2	11	14
3	9	15
4	17	12
5	5	15
6	6	20

4 AIG Algorithm for DBHFSP

For DBHFSP, it is essential to consider three sub-problems, i.e., allocating an appropriate factory and an appropriate machine for each job and sequencing the jobs. Although the basic IGA is known for its simple structure, flexibility, and few parameters, it is also prone to get trapped in local optima. Considering the coupled features of DBHFSP and the limitations of the basic IGA, we have designed the proposed AIG algorithm.

In Algorithm 1, first, we design an improved version of the NEH2[47] method, which considers the discrete characteristics of DBHFSP and utilizes a first-stage processing time descending scheme for solution initialization (see Line 1). Inside the while loop, two neighborhood search strategies, CriticalFactory_ExtractInsert (see Line 5) and CriticalFactory_Swap (see Line 7), are randomly used

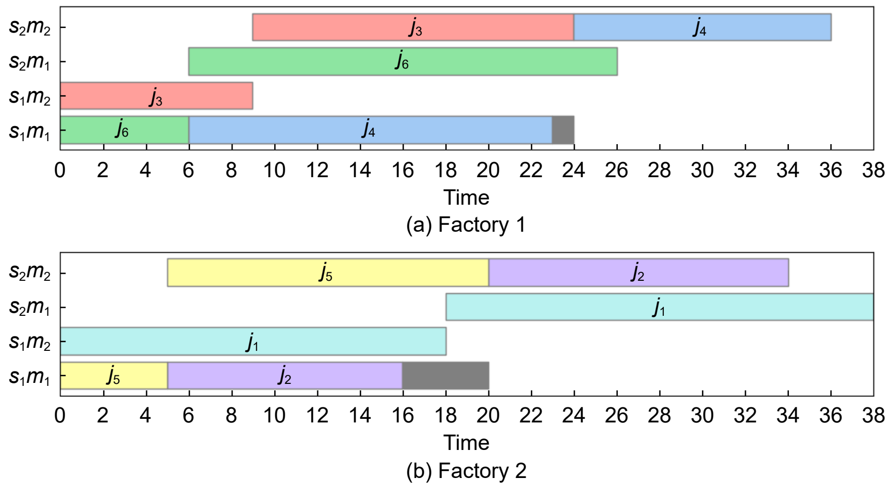

Fig. 2 Scheduling Gantt chart for the instance.

Algorithm 1 AIG algorithm

```
Input: An empty solution $\pi$, and $d$ is the number of jobs to be extracted in DR
Outpu: $\pi^{\text{best}}$
1  $\pi = \text{NEH2\_FSD}(\pi)$, $\pi^{\text{temp}} = \pi$, and $\pi^{\text{best}} = \pi$
2  While (stop time is not reached) do
3    $r = \text{rand}(0, R)$      %% $R$ is a parameter
4    If ($r = 0$)
5      $\pi^{\text{temp}} = \text{CriticalFactory\_ExtractInsert}\left(\pi^{\text{temp}}, d\right)$
6    Else
7      $\pi^{\text{temp}} = \text{CriticalFactory\_Swap}\left(\pi^{\text{temp}}\right)$
8    End If
9    For $f = 1$ to $F$
10     $\pi_f^{\text{temp}'} = \text{DR}\left(\pi_f^{\text{temp}}, d\right)$
11     If ($C_{\max}\left(\pi_f^{\text{temp}'}\right) < C_{\max}\left(\pi^{\text{temp}}\right)$)
12       $\pi_f^{\text{temp}} = \pi_f^{\text{temp}'}$
13     End If
14     $\pi_f^{\text{temp}} = \text{GreedSwap}\left(\pi_f^{\text{temp}}\right)$
15   End For
16   If ($C_{\max}\left(\pi^{\text{temp}}\right) < C_{\max}\left(\pi^{\text{best}}\right)$)
17     $\pi^{\text{best}} = \pi^{\text{temp}}$
18   Else
19     $r' = \text{rand}(0, 1)$
20     If ($r' = 0$)
21       $\pi^{\text{temp}} = \text{Shake\_DoubleInsert}\left(\pi^{\text{temp}}\right)$
22     Else
23       $\pi^{\text{temp}} = \text{Shake\_Swap}\left(\pi^{\text{temp}}\right)$
24     End If
25     If ($C_{\max}\left(\pi^{\text{temp}}\right) < C_{\max}\left(\pi^{\text{best}}\right)$)
26       $\pi^{\text{best}} = \pi^{\text{temp}}$
27     Else
28       $\pi^{\text{temp}} = \pi^{\text{best}}$
29     End If
30   End If
31 End While
```

to perform neighborhood perturbation between a critical factory and a non-critical factory. By utilizing these two neighborhood structures, our algorithm can enhance the collaboration between different factories. Second, the AIG algorithm incorporates the destruction-reconstruction (DR) strategy (Line 10) to introduce diversity into the solutions. Then a local search based on swap operator, denoted as GreedSwap (see Line 14), is employed to further enhance the quality of the solution. Finally, to further prevent the algorithm from being trapped in local optima, we employ two shaking strategies, Shake_DoubleInsert and Shake_Swap (see Lines 21 and 23), to slightly perturb the current solution and escape stagnation.

4.1 Analysis and representation of solution

We first choose a suitable encoding strategy to represent the solution of DBHFSP. Considering the multi-factory environment of DBHFSP, this paper uses a multiple permutation based encoding method[48]. The solution can be expressed as $\pi = \left\{\pi_1, \pi_2, ..., \pi_f, ..., \pi_F\right\}$, where π_f represents the order of jobs in Factory f. For each f, the sequence of jobs is denoted as $\pi_f = \left\{\pi_{f_1}, \pi_{f_2}, ..., \pi_{f_{\eta_f}}\right\}$, where π_{f_j} represents the j-th job in Factory f, $j = 1, 2, ..., \eta_f$. For example, a solution $\pi = \{\pi_1, \pi_2\} = \{\{6, 5, 8, 1\}, \{2, 7, 4, 3\}\}$, which indicates that jobs 6, 5, 8, and 1 are allocated for Factory 1, and jobs 2, 7, 4, and 3 are allocated for Factory 2.

The purpose of decoding is to compute the objective value. For example, a sequence of jobs in Factory f is $\pi_f = \left\{\pi_{f_1}, \pi_{f_2}, ..., \pi_{f_{\eta_f}}\right\}$, where the job in f is denoted as $j \in \left\{1, 2, ..., \eta_f\right\}$. There are S processing stages in f, and each stage s has M_s parallel machines. $[j]$ represents the index of the j-th job. $\text{fc}_{[j],s}$ represents the completion time of the job and $\text{fmt}_{s,m}$ represents the idle time of a machine at a specific stage. m^* is denoted as the machine with the earliest idle time at stage s. The makespan $C_{\max}^f$ for Factory f is calculated as follows:

$$C_{\max}^f = 0 \tag{11}$$

$$\text{fc}_{[j],0} = 0, j = 1, 2, ..., \eta_f \tag{12}$$

$$\text{fmt}_{s,m} = 0, s = 1, 2, ..., S, m = 1, 2, ..., M_s \tag{13}$$

$$m^* = \underset{m=1,2,...,M_s}{\arg\min} \left\{\text{fmt}_{s,m}\right\}, s = 1, 2, \ldots, S \tag{14}$$

$$\text{fc}_{[j],s} = \max\left\{\text{fmt}_{s,m^*}, \text{fc}_{[j],s-1}\right\} + p_{j,s},\\ j = 1, 2, ..., \eta_f, s = 1, 2, ..., S \tag{15}$$

$$\text{fmt}_{s,m^*} = \begin{cases} \text{fc}_{[j],s+1} - p_{[j],s+1}, s = 1, 2, ..., S-1; \\ \text{fc}_{[j],s}, s = S \end{cases} \tag{16}$$

$$C_{\max}^f = \max\left\{C_{\max}^f, \text{fmt}_{s,m}\right\}, s = S, m = 1, 2, ..., M_s \tag{17}$$

According to Section 3.2, the solution is expressed as $\pi = \{\pi_1, \pi_2\}$, where $\pi_1 = \{6, 3, 4\}$ and $\pi_2 = \{5, 1, 2\}$. The completion time for each job is $c_6 = 26$, $c_3 = 24$, $c_4 = 36$, $c_5 = 20$, $c_1 = 38$, and $c_2 = 34$. $C_{\max}^1 = \max\{0, 26,$

$36\} = 36$, $C^2_{\max} = \max\{0, 34, 38\} = 38$. Therefore, the makespan of this example is $C_{\max} = \max\left\{C^1_{\max}, C^2_{\max}\right\} = \max\{36, 38\} = 38$.

For DBHFSP, each factory is mutually independent, and the movement within one factory does not affect the others. Thus, we can obtain the following characteristics of DBHFSP:

Characteristic 1. Exchanging two jobs between different factories does not affect the completion time of other factories; exchanging two jobs within the same factory does not change the completion time of other factories.

Characteristic 2. When deleting a job from one factory and reinserting it into another factory, it does not affect the completion time of other factories; similarly, deleting a job from one position and reinserting it into another position within the same factory does not change the completion time of other factories.

Characteristic 3. Inserting a job into position p^* in the job sequence does not affect the completion time of jobs before position p^*.

These characteristics are obvious and can help the algorithm reduce computation time when exploring the neighborhood. Additionally, based on Characteristic 3, we specifically designed a memory-based decoding method for insertion operations. The job to insert is j^* and the position to insert is p^*. p is a position of the job sequence. The information of $\mathrm{fc}_{[j],m}$ and $\mathrm{fmt}_{s,m}$ before position p^* is known. The calculation formula is as follows:

$$m^* = \underset{m=1,2,\ldots,M_s}{\arg\min}\ \{\mathrm{fmt}_{s,m}\}, s = 1,2,\ldots,S \tag{18}$$

$$j = \begin{cases} j^*, p = p^*; \\ p-1, p \neq p^*, \end{cases}$$
$$p = p^*, p^*+1, \ldots, \eta_f, \eta_f+1 \tag{19}$$

$$\mathrm{fc}_{[j],s} = \max\left\{\mathrm{fmt}_{s,m^*}, \mathrm{fc}_{[j],s-1}\right\} + p_{j,s},$$
$$s = 1,2,\ldots,S \tag{20}$$

$$\mathrm{fmt}_{s,m^*} = \begin{cases} \mathrm{fc}_{[j],s+1} - p_{[j],s+1}, s = 1,2,\ldots,S-1; \\ \mathrm{fc}_{[j],s}, s = S \end{cases} \tag{21}$$

4.2 Initialization strategy

A suitable initialization strategy is crucial for achieving good performance in optimization algorithms. The NEH2 heuristic method has proven to be effective in solving DPFSP[47]. NEH2 follows several key steps. Firstly, a seed sequence is generated by sorting the jobs according to the descending order of the total processing time across all stages. Then the first F jobs from the seed sequence are allocated to F factories, ensuring that each factory receives at least one job. Finally, the rest ones are iteratively inserted into the location with the minimal completion time across all factories. However, if some jobs with large processing time in the first stage are put into the same factory, it can lead to an imbalance of the completion time between factories and may increase the blocking times. To illustrate this situation, we give an example including 8 jobs, 2 stages, and 2 factories. There are two parallel machines at each stage. Table 2 lists the processing time of the jobs. Figure 3a shows the scheduling sequence generated using NEH2. In Fig. 3a , although NEH2 can assign jobs to individual factories according to the total processing time, it may still generate long blocking time. Reducing the blocking time for each factory and avoiding blocking concentrated in one factory help to reduce the completion time.

According to the problem characteristics of DBHFSP and combined with the advantages of NEH2, this paper designs an improved NEH2 initialization method with the first stage processing time descending order (NEH2_FSD). NEH2_FSD first sorts the jobs according to the descending order of their first-stage processing time and then assigns one job to each factory. The rest ones are assigned to each factory using the same approach as NEH2. Figure 3b shows the scheduling sequence generated using NEH2_FSD. The makespan in Fig. 3a is 12 and the makespan in Fig. 3b is 10. From Fig. 3b we can see that using the NEH2_FSD strategy avoids blocking to be concentrated in one factory. The proposed initialization method is shown in Algorithm 2.

In NEH2_FSD, all jobs are first sorted according to

Table 2 Processing time for eight jobs.

Job No.	Processing time	
	Stage 1	Stage 2
1	4	1
2	3	4
3	5	1
4	2	6
5	1	5
6	3	5
7	2	4
8	3	4

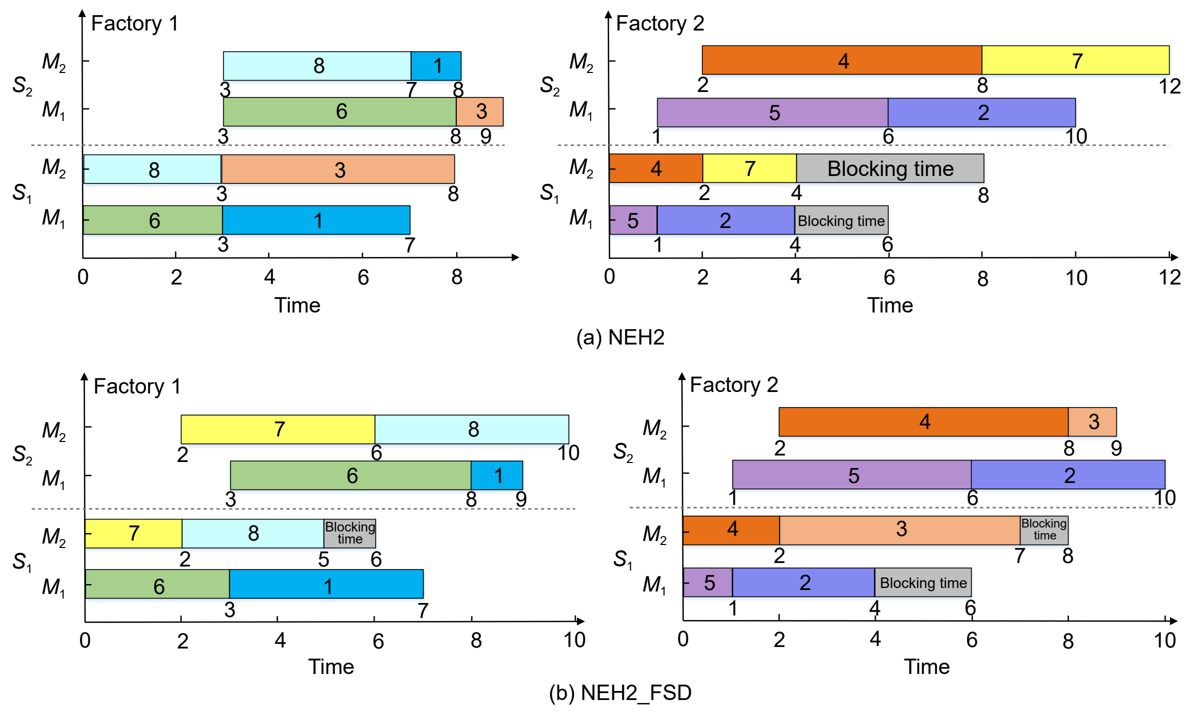

Fig. 3 Gantt chart of solutions obtained by NEH2 and NEH2_FSD.

Algorithm 2 NEH2_FSD(π)

Input: An empty solution π and a sequence of jobs $\Pi^{\text{permutation}}$

Output: π

```
1  $\Pi^{\text{Permutation}} = \text{DescendSort}\left(p_{j,0}\right), j \in \{1,2,\ldots,J\}$
2  For $f = 1$ to $F$
3      $\varpi_f = \Pi_f^{\text{Permutation}}$ %% job $f$ in the queue is scheduled to the Factory $f$
4      $\Pi^{\text{Permutation}} = \Pi^{\text{Permutation}} \backslash \Pi_f^{\text{Permutation}}$
5  End For
6  $\Pi^{\text{Permutation}} = \text{DescendSort}\left(\sum_{s=1}^{S} p_{j,s}\right)$
7  For $j = F+1$ to $J$
8      job = $\Pi_j^{\text{Permutation}}$
9      For $f = 1$ to $F$
10         Obtain the best position BestPos with minimal makespan in Factory $f$
11         Insert job into position BestPos of $\pi_f$
12     End For
13 End For
```

the descending order of their processing time at the first stage, forming a sequence, $\Pi^{\text{permutation}}$ (see Line 1). Allocate the first F jobs in the sequence for each factory separately and ensure that each factory has a job (see Lines 2−5). This is followed by the NEH2 process, which arranges the rest jobs (see Line 6). These jobs are inserted into the position BestPos with the smallest maximum completion time among all factories (see Lines 9−12). Repeat this process until all jobs have been inserted.

4.3 Inter-factory neighborhood search strategy

IGA is often employed to PFSP owing to its uncomplicated structure and excellent local search capability. However, considering the multi-factory feature of DBHFSP, it is crucial to boost the global search performance of IGA. The effectiveness of the current solution enhancement heavily relies on the neighborhood structure. To enhance the exploration of algorithms, we introduce two inter-factory neighborhood search strategies, i.e., CriticalFactory_ ExtractInsert and CriticalFactory_Swap. These two neighborhood search strategies are performed across factories to effectively improve the global search capability of the algorithm through the collaboration between factories.

CriticalFactory_ExtractInsert is shown in Algorithm 3. First, a critical factory $f_{\max}$ that is a factory with the maximum makespan and $f_{\min}$ that is a factory with the minimum makespan are found. Subsequently, the job is randomly selected in $f_{\max}$ (see Lines 5 and 6) and removed from $f_{\max}(\pi_{f_{\max}}^{\text{temp}} = \pi_{f_{\max}}^{\text{temp}} \backslash \text{job})$. Finally, job is inserted into the best location, BestPos, of $f_{\min}$ (see Lines 8 and 9).

The process of CriticalFactory_Swap is similar to that

Algorithm 3 CriticalFactory_ExtractInsert$\left(\pi^{\text{temp}}, d\right)$

```
Input: $\pi^{\text{temp}}$ and $d$
Output: $\pi^{\text{temp}}$
1  For $i = 1$ to $d/2$
2      For $f = 1$ to $F$
3          Find $f_{\max}$ and $f_{\min}$
4      End For
5      $\text{pt} = \text{rand}\left(1, \pi^{\text{temp}}_{f_{\max}}\right)$
6      job= the pt-th job in $\pi^{\text{temp}}_{f_{\max}}$
7      $\pi^{\text{temp}}_{f_{\max}} = \pi^{\text{temp}}_{f_{\max}} \backslash \text{job}$
8      Find the position BestPos with the minimum $C_{\max}$ in $\pi^{\text{temp}}_{f_{\min}}$
9      Insert job into position BestPos of $\pi^{\text{temp}}_{f_{\min}}$
10 End For
```

of CriticalFactory_ExtractInsert. (1) Find the two factories $f_{\max}$ and $f_{\min}$ with the maximum and minimum completion time, respectively. (2) Randomly select job_1 and job_2 from $f_{\min}$ and $f_{\max}$, respectively, and swap them. (3) Execute the acceptance criterion. If $C_{\max}$ becomes smaller, the swap is kept. (4) Repeat Step (2) until all jobs in $f_{\max}$ are tried to exchange with jobs in $f_{\min}$.

4.4 Destruction-reconstruction

The DR operator can improve the diversity of solutions by largely disturbing the current solution. Thus, this paper designs a new DR strategy inside the factory, as shown in Algorithm 4. In the destruction phase, a certain number of d jobs are randomly deleted from π^{temp}_f and placed into Π^{Des} (see Line 4). The remaining jobs compose of π^{temp}_f (see Line 5). In the reconstruction phase, we extract a job Π^{Des}_j from Π^{Des} and reinsert it into all locations in π^{temp}_f. Then find the position, BestPos, with the smallest makespan (see Lines 8 and 9). Repeat the above insertion process until all jobs of Π^{Des} are inserted into π^{temp}_f.

4.5 Inner-factory local search strategy

To strengthen the development of algorithms and explore the solution space inside the factory, a swap-based local strengthening strategy is designed. The reason why the swap operator is adopted as the local search strategy is that its time complexity is lower than the insertion. Under the same running time, the swap is executed more frequently than the insertion, allowing

Algorithm 4 DR$\left(\pi^{\text{temp}}_f, d\right)$

```
Input: $\pi^{\text{temp}}_f$, $d$
Output: $\pi^{\text{temp}}_f$
1  $\Pi^{\text{Des}} = \varnothing$ %% $\Pi^{\text{Des}}$ is a partial collection of jobs
2  While ($\Pi^{\text{Des}}$.size $< d$)
3      $\text{pt} = \text{rand}\left(1, \pi^{\text{temp}}_f.\text{size}\right)$
4      Extract $\pi^{\text{temp}}_{f_{\text{pt}}}$ from $\pi^{\text{temp}}_f$ and put it into $\Pi^{\text{Des}}$
5      $\pi^{\text{temp}}_f = \pi^{\text{temp}}_f \backslash \pi^{\text{temp}}_{f_{\text{pt}}}$
6  End While
7  For $j = 1$ to $d$
8      $\text{job} = \Pi^{\text{Des}}_j$
9      Find the position BestPos with the minimum $C_{\max}$ in $\pi^{\text{temp}}_f$
10     Insert job into position BestPos of $\pi^{\text{temp}}_f$
11 End For
```

the algorithm to explore the solution space more extensively. Therefore, this paper develops a swap-based local strengthening strategy, GreedSwap, presented in Algorithm 5.

In GreedSwap, the function $\text{swap}\left(\pi^{\text{temp}}_{f_i}, \pi^{\text{temp}}_{f_j}\right)$ refers to exchange jobs $\pi^{\text{temp}}_{f_i}$ and $\pi^{\text{temp}}_{f_j}$, and obtain a new sequence $\pi^{\text{temp}'}_f$ (see Line 3). Subsequently, $\pi^{\text{temp}'}_f$ is evaluated. If $\pi^{\text{temp}'}_f$ is better than π^{temp}_f, then $\pi^{\text{temp}'}_f$ is instead of π^{temp}_f (see Lines 4−8).

Algorithm 5 GreedSwap$\left(\pi^{\text{temp}}_f\right)$

```
Input: $\pi^{\text{temp}}_f$
Output: $\pi^{\text{temp}}_f$
1  For $i = 1$ to $\pi^{\text{temp}}_f$.size
2      For $j = i + 1$ to $\pi^{\text{temp}}_f$.size
3          $\pi^{\text{temp}'}_f = \text{swap}\left(\pi^{\text{temp}}_{f_i}, \pi^{\text{temp}}_{f_j}\right)$
4          If ($C_{\max}\left(\pi^{\text{temp}'}_f\right) < C_{\max}\left(\pi^{\text{temp}}_f\right)$)
5              $\pi^{\text{temp}}_f = \pi^{\text{temp}'}_f$
6          Else
7              $\pi^{\text{temp}'}_f = \pi^{\text{temp}}_f$
8          End If
9      End For
10 End For
```

4.6 Shaking strategy

When the local search fails to improve the current solution, shaking strategies are utilized to generate new neighborhoods and escape local optima. To suit the characteristics of DBHFSP, two shaking strategies are proposed in this paper, i.e., inter-factory double insertion Shake_DoubleInsert and inter-factory swap Shake_Swap. Furthermore, to increase the opportunity of exploring the solution space, the shaking procedure terminates as soon as a new and improved solution is obtained. Algorithm 6 gives the process of Shake_DoubleInsert.

Firstly, two factories f_1 and f_2 are randomly chosen. Then jobs $\pi^{\text{temp}'}_{f_{1\text{pt}_1}}$ and $\pi^{\text{temp}'}_{f_{2\text{pt}_2}}$ are randomly selected from f_1 and f_2, respectively (see Lines 2−4). Remove $\pi^{\text{temp}'}_{f_{1\text{pt}_1}}$ and $\pi^{\text{temp}'}_{f_{2\text{pt}_2}}$ from their respective factories (see Lines 3 and 4). Next, insert $\pi^{\text{temp}'}_{f_{1\text{pt}_1}}$ into BestPos$_1$ of $\pi^{\text{temp}'}_{f_2}$, and insert $\pi^{\text{temp}'}_{f_{2\text{pt}_2}}$ into BestPos$_2$ of $\pi^{\text{temp}'}_{f_1}$ (see Lines 5−8). After the above insertion, a new complete solution $\pi^{\text{temp}'}$ is formed. If $\pi^{\text{temp}'}$ is better than the original π^{temp}, then π^{temp} will be updated with $\pi^{\text{temp}'}$, and the whole function ends (see Lines 9−11); otherwise, $\pi^{\text{temp}'}$ is updated with π^{temp} (see Lines 12 and 13).

For the Shake_Swap strategy, the steps are given.

Algorithm 6 Shake_DoubleInsert$\left(\pi^{\text{temp}}\right)$

```
Input: $\pi^{\text{temp}}$, $\pi^{\text{temp}'} = \pi^{\text{temp}}$
Output: $\pi^{\text{temp}}$
1  For $i = 1$ to $J/F$
2      $f_1 = \text{rand}(1, F)$, $f_2 = \text{rand}(1, F)$ //$f_2 \neq f_1$
3      $\text{pt}_1 = \text{rand}\left(1, \pi^{\text{temp}'}_{f_1}\right)$, $\text{job}_1 = \pi^{\text{temp}'}_{f_{1\text{pt}_1}}$, $\pi^{\text{temp}'}_{f_1} = \pi^{\text{temp}'}_{f_1} \backslash \pi^{\text{temp}'}_{f_{1\text{pt}_1}}$
4      $\text{pt}_2 = \text{rand}\left(1, \pi^{\text{temp}'}_{f_2}\right)$, $\text{job}_2 = \pi^{\text{temp}'}_{f_{2\text{pt}_2}}$, $\pi^{\text{temp}'}_{f_2} = \pi^{\text{temp}'}_{f_2} \backslash \pi^{\text{temp}'}_{f_{2\text{pt}_2}}$
5      Find BestPos$_1$ with the minimal $C_{\max}$ in $\pi^{\text{temp}'}_{f_2}$
6      Put job$_1$ into BestPos$_1$ of $\pi^{\text{temp}'}_{f_2}$
7      Find BestPos$_2$ with the minimal $C_{\max}$ in $\pi^{\text{temp}'}_{f_1}$
8      Put job$_2$ into position BestPos$_2$ of $\pi^{\text{temp}'}_{f_1}$
9      If ($C_{\max}\left(\pi^{\text{temp}'}\right) < C_{\max}\left(\pi^{\text{temp}}\right)$)
10         $\pi^{\text{temp}} = \pi^{\text{temp}'}$
11         Break        %% Ending the cycle
12     Else
13         $\pi^{\text{temp}'} = \pi^{\text{temp}}$
14     End If
15 End For
```

Firstly, two factories f_1 and f_2 are randomly chosen. Secondly, jobs $\pi^{\text{temp}'}_{f_{1\text{pt}_1}}$ and $\pi^{\text{temp}'}_{f_{2\text{pt}_2}}$ are randomly chosen from f_1 and f_2, respectively. Thirdly, swap $\pi^{\text{temp}'}_{f_{1\text{pt}_1}}$ and $\pi^{\text{temp}'}_{f_{2\text{pt}_2}}$ to form a new solution, $\pi^{\text{temp}'}$. If $C_{\max}\left(\pi^{\text{temp}'}\right) < C_{\max}\left(\pi^{\text{temp}}\right)$, then $\pi^{\text{temp}} = \pi^{\text{temp}'}$. Otherwise, skip to the first step until the end of the loop.

5 Simulation Experiments and Analysis

We have conducted a series of simulation experiments to comprehensively demonstrate the performance of AIG. Since DBHFSP is a relatively new research topic, it is important to prove the feasibility of the proposed model through small-scale examples. Additionally, we will use analysis of variance (ANOVA) to optimize the parameters of the proposed algorithm. Subsequently, the performance of each ingredient in the proposed algorithm will be evaluated and tested. Finally, we will verify the comprehensive effectiveness of AIG by comparing it with some related algorithms.

5.1 Experimental setup

Following Ref. [45], 80 medium and large test instances are formed by combining F, S, and J, where $F \in \{2, 3, 4, 5\}$, $S \in \{3, 5, 8, 10\}$, and $J \in \{100, 200, 300, 400, 500\}$. Each combination $F \times S \times J$ is considered as an instance that is executed independently 20 times to reduce the influence of randomness in all simulation experiments. The processing time of each job is uniformly distributed between [1, 99], and the number of machines is obtained within [1, 5] for each instance. To guarantee the fairness of experimental results, the same stop running time is set as $\text{TimeLimit} = F \times S \times J \times t$ for all algorithms, where t is the control parameter for the effective running time and is set to 3 and 5. Moreover, all algorithms are written in visual studio 2019 C++ and executed on a PC with an Intel(R) Core i7 processor running at 3.60 GHz and 32.0 GB RAM.

To clearly and intuitively reflect the search results of each algorithm, the relative percentage increase (RPI) is employed as an evaluation metric. RPI measures the disparity between the makespan of an algorithm and the best one found so far. RPI evaluation indicator is defined by the following formula:

$$\text{RPI} = (c_i - c_{\min}) / c_{\min} \times 100 \tag{22}$$

where c_i represents an average of the makespan generating by the i-th algorithm after independently running 20 times. $c_{\min}$ represents the minimum makespan found by all algorithms. In addition, an average RPI (ARPI) of 80 different test instances is calculated. A smaller RPI or ARPI indicates better algorithm performance.

5.2 Validation of MILP model

To verify the accuracy of the proposed MILP model, 12 small scale instances are selected and solved using Gurobi. The stop running time of Gurobi and AIG are 3600 s and TimeLimit $= F \times S \times J \times 3$, respectively. For AIG, each instance is independently executed 20 times, and the makespan value reported in Table 3 is the average makespan after independently running 20 times. For the Gurobi solver, the Gap value is 0, indicating that the optimal solution has been obtained. The instances and code for model validation can be found on GitHub (https://github.com/nideluckily/DBHFSP_MILP.git).

In Table 3, MILP can obtain the best makespan on small-scale instances, i.e., $2\times 8\times 2$, $2\times 8\times 3$, $2\times 8\times 4$, $2\times 13\times 2$, $2\times 13\times 3$, $2\times 13\times 4$, $2\times 18\times 2$, $2\times 18\times 3$, and $2\times 18\times 4$ instances. In 7 out of 12 instances, the MILP model achieves better makespan values than AIG, indicating its superior performance on small-scale instances. However, as the instance size increases, AIG performs better than the Gurobi solver in obtaining makespan values, i.e., $2\times 22\times 2$, $2\times 22\times 3$, and $2\times 22\times 4$. Furthermore, the running time of AIG is significantly lower than that of the Gurobi solver, indicating that AIG can efficiently handle large-scale instances. Therefore, it can be concluded that AIG is a better choice for solving large-scale and complex problems compared to the Gurobi solver.

5.3 Parameters calibration

The AIG algorithm involves calibration of two parameters: d and R (the upper bound of the random number when choosing a neighborhood search strategy). The sensitivity of these two parameters is tested using Taguchi experimental method.

There are 5 levels for each parameter, i.e., $d \in \{2,3,4,5,6\}$ and $R \in \{2,3,4,5,6\}$. Thus, we obtain $5\times 5 = 25$ combinations by orthogonal table, and their orthogonal table is shown in Table 4. We randomly selected eight instances with a relatively large span to ensure the comprehensiveness of the experiment, i.e., $2\times 100\times 3$, $2\times 300\times 8$, $3\times 200\times 5$, $3\times 400\times 10$, $4\times 400\times 3$, $4\times 500\times 5$, $5\times 200\times 8$, and $5\times 500\times 10$. Based on the RPI values obtained from the experiments, the trends of the factor levels are plotted to analyze their impact on the performance of AIG, i.e., Fig. 4.

According to Fig. 4, the ARPI value is minimized when $d = 5$, indicating that AIG performs the best under this parameter setting. As d increases beyond 5, ARPI also increases. This may be because a larger d value can lead to the destruction of near-optimal solutions, wasting more time and reducing AIG's efficiency. Conversely, setting d too small may result in an insufficient degree of job sequence disturbance, leading to less exploration of the search space. Another parameter, R, is used to control the selection of the inter-factory strategies. Considering that the completion time of each factory should be relatively even in the late iteration, we expect more swap operators to be used to maintain the balance between factories (the insertion operator is only used when $R = 0$). Figure 4 shows that AIG performs the best

Table 3 Experimental result of MILP and AIG.

$F\times J\times S$	MILP			AIG	
	Makespan	Time (s)	Gap (%)	Makespan	Time (s)
$2\times 8\times 2$	**65**	0.05	0	**65**	0.04
$2\times 8\times 3$	**117**	0.04	0	**117**	0.06
$2\times 8\times 4$	**112**	0.96	0	124	0.08
$2\times 13\times 2$	**97**	2.60	0	99	0.06
$2\times 13\times 3$	**141**	25.52	0	142	0.08
$2\times 13\times 4$	**166**	24.10	0	170	0.11
$2\times 18\times 2$	**149**	3600	23.49	150	0.08
$2\times 18\times 3$	**164**	3600	20.12	169	0.11
$2\times 18\times 4$	**175**	3600	16.57	176	0.16
$2\times 22\times 2$	166	3600	50	**165**	0.09
$2\times 22\times 3$	195	3600	42.05	**194**	0.14
$2\times 22\times 4$	221	3600	27.15	**217**	0.19

Table 4 Orthogonal table and response values.

Combination No.	Parameter		ARPI
	d	*R*	
1	2	2	0.810 388
2	2	3	0.824 091
3	2	4	0.749 053
4	2	5	0.769 257
5	2	6	0.775 704
6	3	2	0.774 233
7	3	3	0.755 607
8	3	4	0.707 455
9	3	5	0.721 388
10	3	6	0.729 333
11	4	2	0.735 936
12	4	3	0.703 336
13	4	4	0.663 382
14	4	5	0.694 982
15	4	6	0.672 266
16	5	2	0.740 136
17	5	3	0.721 102
18	5	4	0.667 080
19	5	5	0.611 030
20	5	6	0.684 753
21	6	2	0.790 138
22	6	3	0.728 888
23	6	4	0.705 961
24	6	5	0.686 460
25	6	6	0.637 642

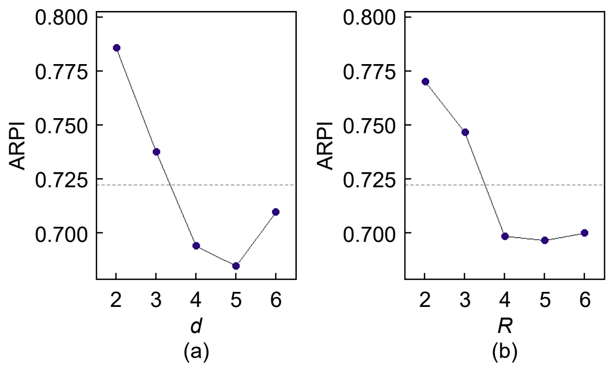

Fig. 4 Taguchi analysis for the AIG calibration.

when $R = 5$.

Table 5 displays the ARPI values and significance levels of the two parameters with Delta measuring the performance based on the disparity between the maximum and minimum ARPI across the five levels. A significant effect of the parameter is indicated by a larger Delta or a smaller Rank indicator. From Table 5, *d* has a large impact on the algorithm, while the

Table 5 ARPI response values of each parameter.

Level	ARPI	
	d={2, 3, 4, 5, 6}	R={2, 3, 4, 5, 6}
1	0.7857	0.7702
2	0.7376	0.7466
3	0.6940	0.6986
4	0.6848	0.6966
5	0.7098	0.6999
Delta	0.1009	0.0735
Rank	1	2

Note: When Level=1, d=2 and R=2. When Level=2, d=3 and R=3. When Level=3, d=4 and R=4. When Level=4, d=5 and R=5. When Level=5, d=6 and R=6.

difference between the Delta values of R and d is not significant. From the average RPI values given in Table 5, it can be seen that the AIG has the best performance when $d = 5$ and $R = 5$. According to the above analysis, we set $d = 5$ and $R = 5$.

5.4 Evaluation of initialization methods

To improve the convergence performance of AIG, an appropriate initialization strategy is crucial. The NEH2 heuristic, which has been widely utilized in solving DPFSP and has undergone numerous enhancements, can generate high-quality initial solutions. In this paper, we modify NEH2 according to the specific characteristics of DBHFSP to further enhance its performance. We develop an NEH2_FSD initialization method by combining NEH2 and the first stage processing time descending priority rule. Further, we compare the proposed initialization strategy with NEH2. The interval plot of the experimental results is shown in Fig. 5.

From Fig. 5, the interval of NEH2_FSD is lower than that of NEH2, and the two intervals do not overlap, suggesting that the algorithm can generate better initial solutions using NEH2_FSD. This improvement is attributed to the use of the first-stage processing time descending priority rule to mitigate the impact of

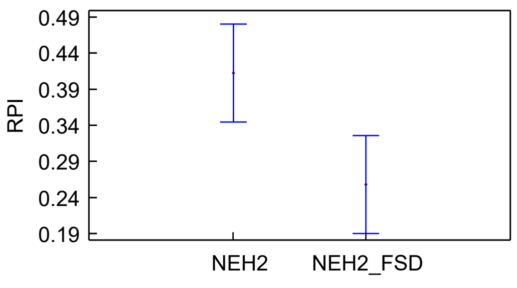

Fig. 5 RPI-based confidence intervals for different initialization methods.

blocking constraints and balance the completion time of individual factories. In conclusion, to ensure efficient convergence and obtain high-quality solutions rapidly, we adopt NEH2_FSD as the initialization method.

5.5 Performance analysis for the proposed components of AIG

This section presents four groups of simulation experiments aiming at demonstrating the effectiveness of the four main components. These components include the proposed decoding strategy, the inter-factory neighborhood search strategies, the different local search strategies, and the shaking methods.

5.5.1 Evaluation of memory-based decoding strategy

In Section 4, we propose a memory-based decoding strategy, which can help the algorithm save time for more iterations and can explore the neighborhood structure more deeply. To verify its effectiveness, the performance of AIG with the memory-based decoding method is compared to AIG with the normal decoding strategy. The confidence intervals of the experimental results are presented in Fig. 6.

From Fig. 6, the memory-based decoding interval is lower than the ordinary decoding interval, and the two intervals do not overlap. The results indicate that the memory-based decoding strategy outperforms the normal decoding strategy in solving DBHFSP. The reason may be that the memory-based decoding strategy can reduce unnecessary computation, facilitate deeper exploration of neighborhoods, and increase the possibility of discovering better solutions. Therefore, we use memory-based decoding for DBHFSP.

5.5.2 Evaluation of inter-factory neighborhood search strategies

Considering the multi-factory feature of DBHFSP, we develop two cross-factory neighborhood search strategies. They help algorithms explore wider neighborhood structures while enhancing the connections among factories. These strategies include an insertion-based approach and a swap-based approach. In addition, considering that a single neighborhood search may not significantly affect the algorithm's global search ability, we employ a hybrid approach of the two inter-factory neighborhood search strategies. To demonstrate the excellence of the neighborhood search strategy, we conducted three sets of experiments: one with only insertion-based neighborhood search, one with only swap-based neighborhood search, and one with hybrid neighborhood search.

Figure 7 demonstrates that the insertion operator is better than the swap operator. Moreover, the hybrid neighborhood search strategy by combining both insertion and swap operators outperforms the individual insertion and swap operators by a significant margin. This observation demonstrates the effectiveness of our hybrid neighborhood search strategy. By enhancing the collaboration among the factories, the strategy generates a more diverse set of neighborhoods, which in turn increases the probability of finding high-quality solutions.

5.5.3 Evaluation of local search strategies within the factory

The insertion operator can effectively reinforce the current solution in the IG algorithm. However, to accelerate the perturbation of job sequence and mitigate the effect of blocking constraints, we propose a swap-based local search strategy with lower time complexity. We test the performance of the insertion operator (AIG_insert), the swap operator (AIG_swap), and the no-perturbation strategy (AIG_No) under the same experimental condition. The experimental results are illustrated in Fig. 8. Figure 8 displays the confidence intervals to provide a more intuitive comparison.

In Fig. 8, the performance is greatly reduced when there is no local search strategy in AIG, which reflects the necessity of the local search strategy. Meanwhile, Fig. 8 indicates that the swap operator outperforms the insertion operator evidenced by the smaller RPI value.

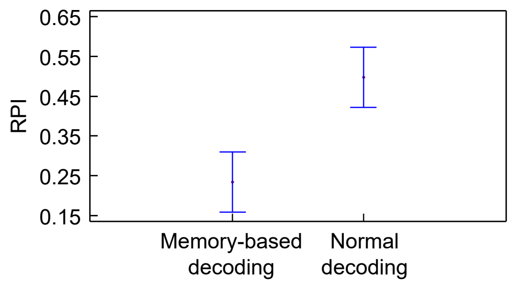

Fig. 6 RPI-based confidence intervals for different decoding methods.

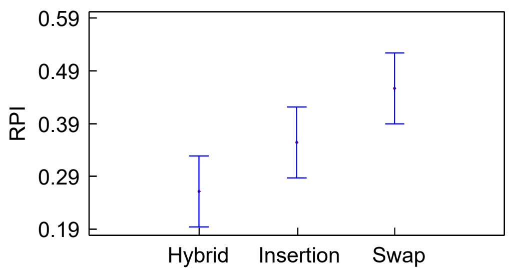

Fig. 7 RPI-based confidence intervals for different inter-factory neighborhood search strategies.

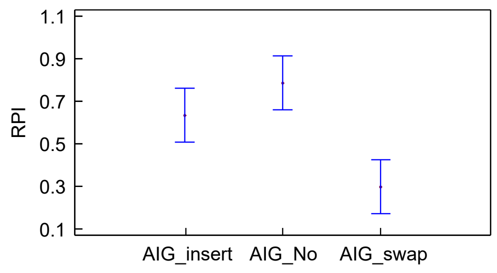

Fig. 8 RPI-based confidence intervals for different local search strategies within the factory.

It can be attributed to the fact that the algorithm utilizing the swap operator has more iterations than the insertion operator for the same termination time, which can enhance AIG's exploitation capabilities and help it explore deeper neighborhoods. On this basis, the algorithm can find more promising solutions. Based on the results, we adopt the swap-based local search strategy to obtain more potential solutions.

5.5.4 Evaluation of shaking methods

In DBHFSP, since there are multiple factories, the job sequence in each factory may reach a local optimum prematurely due to local search, leading to suboptimal solutions. To address this issue, we design two shaking strategies, i.e., inter-factory double insertion and inter-factory swap. To prove the effectiveness of these shaking strategies, AIG with a hybrid strategy (both shaking strategies are used) (AIG_hybrid), AIG with single insertion operator (AIG_insert), AIG with single swap operator (AIG_swap), and AIG without shaking strategy (AIG_No) are tested under the same experimental condition.

In Fig. 9, the RPI value of the algorithm without the shaking strategy is larger than the other three, which demonstrates that the shaking strategies are effective. In addition, among the three algorithms using the shaking strategy, it can be found that the algorithm with a hybrid strategy is better than the other two strategies. The hybrid strategy is effective due to its ability to increase the algorithm's diversity, explore more unknown neighborhoods, and facilitate the discovery of better solutions.

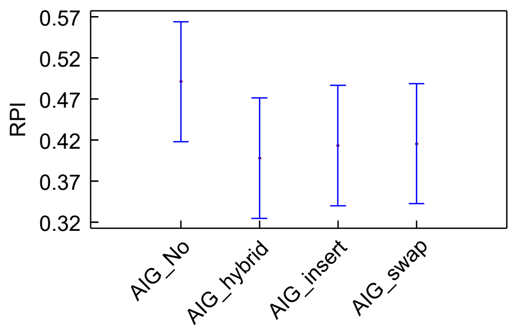

Fig. 9 RPI-based confidence intervals for different shaking strategies.

From Sections 5.5.1 to 5.5.4, it can be observed that the algorithm components designed in this paper can improve the performance of the algorithm and find better solutions. We calculate the percentage of performance improvement for each component using the formula $(P_N - P_Y)/P_N \times 100\%$, where P_Y and P_N represent the average RPI values of AIG with and without the proposed component, respectively. Through calculations, the percentage of improvement provided by the memory-based decoding strategy is 54%, the inter-factory neighborhood search strategy is 43%, the local search strategy is 62%, and the shaking strategy is 18%. Therefore, based on the above results, the local search strategy has the highest impact, followed by the memory-based decoding strategy, the inter-factory neighborhood search strategy, and the shaking strategy.

5.6 Performance verification of all comparative algorithms

To assess the effectiveness of AIG, AIG is compared with five advanced algorithms in this section, i.e., CIG[45], discrete differential evolution (DDE) algorithm[10], evolutionary algorithm (EA)[11], multi-neighborhood IGA (MN_IG)[1], and DPSO[9]. EA and DDE are two types of swarm intelligence algorithms, which have good global search ability and can improve the diversity of solutions. These algorithms have been proven effective in solving DPFSP and DBFSP, respectively. DPSO is a classic swarm intelligence algorithm, which outperforms other comparative algorithms in solving HFSP. MN_IG is an IGA with a multi-neighborhood search structure, which balances exploitation and exploration abilities. MN_IG has demonstrated good performance in solving the DBFSP. Additionally, CIG is a recently proposed algorithm specifically designed for solving the DHFSP with blocking constraints, which is same as our research topic. The above five comparative algorithms have shown good performance in solving related problems. Among five algorithms, the problems solved by EA, DDE, and MN_IG are very similar to our problem except for not considering hybrid flow shop. For experimental fairness, we have blended strategies related to hybrid flow shop into EA, DDE, and MN_IG to fit our problem. Similarly, we have improved DPSO by considering the distributed environment. Therefore, it is appropriate to choose them as comparison algorithms. The parameters of all comparison algorithms are set to the values recommended in the original literature and given in Table 6. We conducted extensive experiments in advance for each parameter setting and performed sensitivity analysis on the parameters that have a significant impact on the algorithm, as shown in Section 5.3. The remaining parameters are set according to empirical guidelines. All algorithms are performed under the termination conditions of $t = 3$ and $t = 5$. More profoundly, to visually demonstrate the convergence of all algorithms, the confidence intervals of test algorithms are given in Figs. 10 and 11.

In Tables 7 and 8, AIG demonstrates excellent performance in solving 80 instances of different sizes (the best values are bolded). AIG obtains 76 best minimal makespan (MIN) when $t = 3$, followed by CIG (4), EA (2), DDE (0), MN_IG (0), and DPSO (0). Meanwhile, AIG obtains the best minimal makespan in 77 instances, followed by EA (3) and CIG (1), and the number of the best values for DDE, MN_IG, and DPSO are all 0. We calculate the percentage of AIG's superiority over other comparison algorithms using the formula $(P_{\text{Com}} - P_{\text{AIG}}) / P_{\text{Com}} \times 100\%$, where P_{Com} and P_{AIG} represent the MIN or RPI values of the comparison algorithm and AIG, respectively. For the minimum makespan, AIG outperforms CIG, DDE, EA, MN_IG, and DPSO by 0.34%, 1.27%, 0.73%, 1.36%, and 1.05%, respectively. For the RPI, the percentages of AIG outperforming other comparison algorithms are 75%, 88%, 82%, 91%, and 86%, respectively. In addition, AIG obtains the maximum number of best makespan and RPI values when $t = 5$ (see Table 8). In

Table 6 Parameter configurations of the comparison algorithms.

Algorithm	Population size	Number of destruction jobs	Mutation rate	Crossover rate	Temperature coefficient
CIG	–	6	–	–	–
DDE	30	–	0.6	0.1	–
EA	5	–	–	–	–
MN_IG	–	4	–	–	0.4
DPSO	100	–	–	–	–

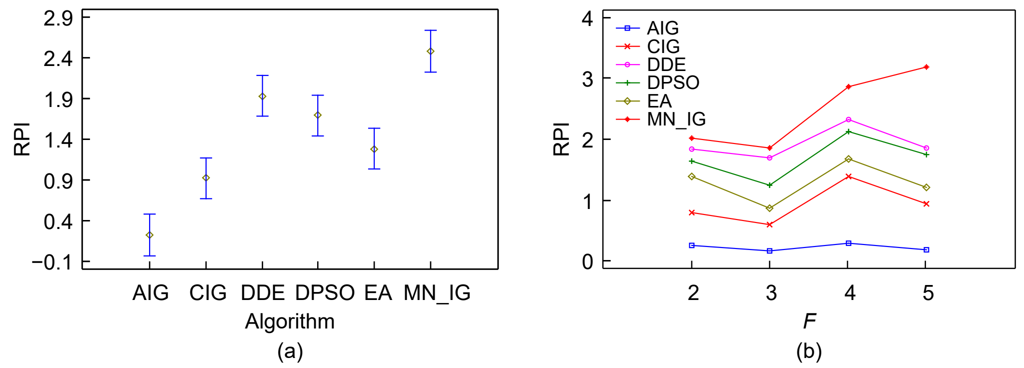

Fig. 10 Confidence intervals of all algorithms when $t = 3$.

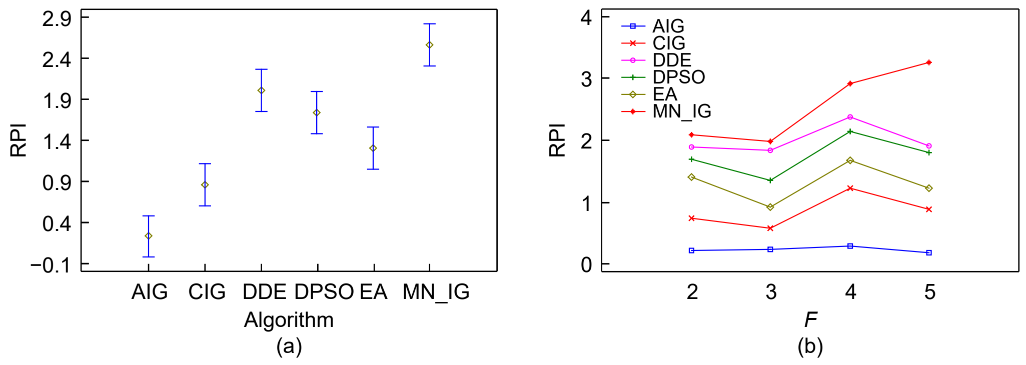

Fig. 11 Confidence intervals of all algorithms when $t = 5$.

Table 7 Experimental results of all comparison algorithms when t= 3.

$F\times J\times S$	AIG		CIG		DDE		EA		MN_IG		DPSO	
	MIN	RPI	MIN	RPI	MIN	RPI	MIN	RPI	MIN	RPI	MIN	RPI
2×100×3	**2560**	**0.54**	2594	1.42	2600	1.56	2594	1.33	2615	2.32	2594	1.33
2×100×5	**1375**	**0.32**	1386	1.05	1400	1.82	1389	1.19	1401	2.75	1393	1.61
2×100×8	**2765**	**0.60**	2794	1.05	2801	1.30	2794	1.05	2836	2.99	2794	1.06
2×100×10	**2979**	**0.26**	3003	0.88	3028	1.64	3003	0.81	3019	1.73	3003	0.88
2×200×3	**2460**	**0.55**	2486	1.75	2582	4.96	2550	4.28	2573	4.90	2582	4.96
2×200×5	**4795**	**0.12**	4813	0.41	4826	0.65	4812	0.35	4844	1.06	4812	0.36
2×200×8	**6072**	**0.41**	6117	1.53	6456	6.32	6266	3.64	6295	4.20	6375	5.26
2×200×10	**5340**	**0.09**	5351	0.21	5370	0.56	5351	0.22	5373	0.71	5351	0.23
2×300×3	**3771**	**0.07**	3775	0.19	3783	0.32	3775	0.18	3789	0.81	3776	0.19
2×300×5	**8168**	**0.11**	8175	0.12	8217	0.60	8179	0.17	8248	1.19	8191	0.49
2×300×8	**4365**	**0.44**	4403	2.21	4635	6.19	4564	5.17	4573	5.23	4615	5.87
2×300×10	**8215**	**0.19**	8234	0.24	8234	0.23	8234	0.23	8242	0.49	8234	0.23
2×400×3	**2745**	**0.74**	2788	3.29	2937	6.99	2932	6.81	2944	7.62	2937	6.99
2×400×5	**10421**	**0.06**	10434	0.19	10451	0.29	10432	0.16	10455	0.45	10448	0.28
2×400×8	**5323**	**0.06**	5325	0.18	5330	0.13	5328	0.09	5345	0.47	5330	0.13
2×400×10	**10274**	**0.04**	10277	0.05	10291	0.17	10277	0.06	10278	0.32	10291	0.17
2×500×3	**12160**	**0.03**	12166	0.05	12168	0.07	12166	0.05	12207	0.40	12166	0.05
2×500×5	**12864**	**0**	**12864**	**0**	12873	0.07	**12864**	0.01	12884	0.16	12864	0.01
2×500×8	12816	**0.17**	**12805**	0.49	12981	1.37	12904	0.93	12862	0.67	12957	1.29
2×500×10	13099	**0.34**	**13066**	0.40	13259	1.48	13162	1.13	13312	2.06	13259	1.48
3×100×3	**1804**	**0.30**	1813	0.58	1877	4.05	1823	1.22	1855	3.33	1835	1.86
3×100×5	1801	**0.14**	**1800**	0.30	1853	2.94	1811	0.84	1834	2.78	1829	1.84
3×100×8	**1965**	**0.26**	1975	1.45	2049	4.27	1989	1.66	2020	3.38	2014	2.95
3×100×10	**2302**	**0.69**	2311	1.71	2449	6.39	2340	2.18	2413	5.56	2397	4.81
3×200×3	**3318**	**0.28**	3339	0.66	3347	0.87	3336	0.54	3374	1.75	3336	0.58
3×200×5	**1238**	**0.21**	1248	1.63	1318	6.46	1296	5.28	1295	5.46	1311	6.17
3×200×8	**3455**	**0.30**	3484	1.05	3512	1.65	3486	0.98	3525	2.37	3490	1.25
3×200×10	**3445**	**0.05**	3448	0.17	3451	0.17	3447	0.07	3469	1.57	3448	0.15
3×300×3	**1702**	**0.16**	1705	0.51	1727	1.47	1715	0.95	1719	1.32	1720	1.10
3×300×5	**5608**	**0.06**	5619	0.21	5622	0.25	5617	0.17	5656	0.92	5620	0.22
3×300×8	**5058**	**0.04**	5061	0.17	5079	0.42	5061	0.06	5084	0.60	5061	0.09
3×300×10	**5223**	**0.16**	5241	0.50	5263	0.77	5238	0.37	5283	1.26	5246	0.49
3×400×3	**2255**	**0.19**	2269	1.35	2305	2.22	2301	2.15	2299	2.23	2305	2.22
3×400×5	**6837**	**0.02**	6848	0.18	6850	0.19	6846	0.14	6897	0.93	6850	0.19
3×400×8	**6991**	**0.01**	6998	0.20	7006	0.21	6997	0.12	7026	0.80	7002	0.19
3×400×10	7224	0.33	7225	0.31	7271	0.75	**7217**	**0.07**	7260	0.71	7227	0.22
3×500×3	**8279**	**0.05**	8289	0.17	8295	0.19	8287	0.10	8326	0.57	8287	0.12
3×500×5	**8608**	**0.08**	8623	0.28	8639	0.36	8621	0.16	8651	0.53	8624	0.20
3×500×8	**8887**	**0.09**	8904	0.21	8906	0.21	8899	0.14	8939	0.61	8900	0.17
3×500×10	**8626**	**0.04**	8633	0.11	8634	0.09	8630	0.06	8647	0.29	8634	0.09
4×100×3	**1171**	**0**	1173	0.22	1176	0.43	1172	0.09	1203	3.09	1174	0.36
4×100×5	**721**	**0.75**	731	1.64	749	3.88	733	1.91	757	6.77	747	3.83
4×100×8	**1387**	0.25	1394	0.87	1396	0.65	1389	**0.14**	1428	3.07	1392	0.59
4×100×10	**1396**	**0.72**	1426	2.79	1458	4.44	1415	1.65	1432	3.07	1428	2.51
4×200×3	**2512**	**0.07**	2520	0.42	2533	0.84	2518	0.25	2554	1.71	2520	0.36
4×200×5	**2478**	**0.36**	2491	0.52	2498	0.81	2490	0.49	2537	2.38	2493	0.64

(to be continued)

Table 7 Experimental results of all comparison algorithms when $t = 3$.

(continued)

$F\times J\times S$	AIG		CIG		DDE		EA		MN_IG		DPSO	
	MIN	RPI	MIN	RPI	MIN	RPI	MIN	RPI	MIN	RPI	MIN	RPI
4×200×8	**2707**	**0.22**	2725	0.66	2725	0.66	2724	0.66	2749	1.77	2725	0.66
4×200×10	**1607**	**0.52**	1642	3.07	1707	6.22	1668	4.47	1670	4.73	1687	5.50
4×300×3	**1266**	**0.30**	1285	1.99	1307	3.24	1301	3.05	1301	3.71	1307	3.24
4×300×5	**3845**	**0.27**	3867	0.96	3951	2.76	3911	2.08	3902	1.97	3950	2.75
4×300×8	**3937**	**0.10**	3946	0.33	3951	0.36	3947	0.27	4000	1.80	3951	0.36
4×300×10	**4224**	**0.08**	4238	0.60	4271	1.11	4235	0.33	4269	1.16	4250	0.72
4×400×3	**4938**	**0.21**	4954	0.38	4959	0.43	4952	0.29	5009	1.45	4954	0.36
4×400×5	**5183**	**0.11**	5196	0.44	5215	0.62	5193	0.22	5221	1.10	5215	0.62
4×400×8	**5389**	**0.22**	5442	1.18	5482	1.73	5468	1.60	5476	2.21	5482	1.73
4×400×10	**5400**	**0.07**	5421	0.49	5438	0.70	5416	0.34	5445	1.15	5438	0.70
4×500×3	**2168**	**0.38**	2186	2.58	2256	4.06	2243	3.70	2241	4.01	2256	4.06
4×500×5	**2343**	**0.51**	2405	4.80	2504	6.87	2484	6.41	2470	6.23	2504	6.87
4×500×8	**3665**	**0.57**	3723	3.23	3889	6.11	3852	5.32	3822	4.89	3876	5.96
4×500×10	**6467**	0.25	6475	0.42	6498	0.63	6457	**0.15**	6509	0.96	6498	0.63
5×100×3	**932**	**0.11**	936	0.71	941	0.97	933	0.15	981	5.67	937	0.73
5×100×5	**590**	**0.51**	596	1.63	613	3.90	602	2.47	616	7.15	613	3.90
5×100×8	**1124**	**0.23**	1134	1.39	1137	1.16	1126	0.27	1190	6.07	1134	1.09
5×100×10	**1369**	**0.20**	1392	1.99	1452	6.06	1409	3.20	1440	6.36	1440	5.65
5×200×3	**2090**	**0.03**	2093	0.30	2098	0.38	2091	0.05	2142	2.49	2092	0.21
5×200×5	**683**	**0.76**	686	1.67	713	4.39	711	4.16	718	5.97	713	4.39
5×200×8	**2303**	**0.11**	2311	0.46	2322	0.83	2307	0.21	2363	2.83	2311	0.50
5×200×10	**2236**	**0.22**	2249	0.88	2265	1.30	2246	0.55	2291	2.59	2254	0.92
5×300×3	**3053**	**0.16**	3062	0.33	3067	0.46	3058	0.19	3097	1.45	3061	0.33
5×300×5	**1672**	**0.23**	1684	1.23	1723	3.05	1717	2.80	1738	4.35	1723	3.05
5×300×8	**3277**	**0.02**	3286	0.50	3290	0.40	3283	0.25	3329	1.95	3290	0.40
5×300×10	**3369**	**0.16**	3423	1.94	3454	2.52	3417	1.69	3448	2.75	3454	2.52
5×400×3	**4040**	**0.14**	4051	0.32	4054	0.35	4049	0.23	4101	1.54	4051	0.32
5×400×5	**2224**	**0.06**	2249	2.28	2349	5.62	2320	4.78	2329	5.02	2349	5.62
5×400×8	**4088**	**0.18**	4106	0.72	4134	1.13	4102	0.44	4136	1.33	4134	1.13
5×400×10	**4284**	**0.06**	4295	0.55	4314	0.70	4291	0.26	4341	1.50	4305	0.57
5×500×3	**5160**	**0.16**	5202	1.13	5289	2.50	5255	2.09	5264	2.09	5289	2.50
5×500×5	**5020**	**0.02**	5023	0.18	5032	0.24	5022	0.05	5053	0.79	5026	0.17
5×500×8	**5256**	**0.04**	5264	0.21	5271	0.29	5258	0.08	5287	0.73	5265	0.22
5×500×10	**5404**	**0.10**	5409	0.23	5446	0.78	5410	0.20	5457	1.10	5446	0.78
Mean	**4352**	**0.23**	4367	0.92	4408	1.93	4384	1.28	4412	2.48	4398	1.69
Percentage (%)	–	–	0.34	75	1.27	88	0.73	82	1.36	91	1.05	86

Note: Percentage means the percentage of AIG better than other comparison algorithms.

summary, AIG is better than the other comparison algorithms. The reason may be that our inter-factory perturbation strategies generate more neighborhood structures and hopefully find more approximate solutions. Furthermore, the local search strategy in the factory can strengthen the development of the algorithm, which can better solve DBHFSP.

The confidence intervals of all algorithms are presented in Figs. 10 and 11. In Fig. 10a, AIG has the smallest confidence interval, indicating that the overall performance of AIG outperforms CIG, EA, DPSO, DDE, and MN_IG. Meanwhile, in Fig. 10b, the curve of AIG is less undulating and smoother than other algorithms, which proves that the convergence ability of AIG is strong and its performance is stable. Similarly, in Fig. 11, AIG demonstrates superior performance compared to the other algorithms. This finding provides evidence to support the assertion that

Table 8 Experimental results of all comparison algorithms when $t=5$.

$F\times J\times S$	AIG		CIG		DDE		EA		MN_IG		DPSO	
	MIN	RPI	MIN	RPI	MIN	RPI	MIN	RPI	MIN	RPI	MIN	RPI
2×100×3	**2558**	**0.45**	2594	1.50	2600	1.64	2594	1.41	2615	2.40	2594	1.41
2×100×5	**1370**	**0.53**	1386	1.42	1400	2.19	1388	1.55	1401	3.12	1393	1.87
2×100×8	**2765**	**0.58**	2794	1.05	2801	1.30	2794	1.05	2836	2.99	2794	1.05
2×100×10	**2979**	**0.21**	3003	0.88	3028	1.64	3003	0.81	3019	1.73	3003	0.87
2×200×3	**2460**	**0.41**	2480	1.51	2582	4.96	2559	4.26	2573	4.90	2582	4.96
2×200×5	**4793**	**0.12**	4813	0.45	4826	0.69	4812	0.40	4844	1.10	4812	0.40
2×200×8	**6044**	**0.55**	6114	1.61	6456	6.82	6203	3.31	6295	4.69	6375	5.63
2×200×10	**5340**	**0.07**	5351	0.21	5370	0.56	5351	0.21	5373	0.71	5351	0.22
2×300×3	**3771**	**0.07**	3775	0.17	3783	0.32	3776	0.17	3789	0.81	3776	0.19
2×300×5	**8168**	**0.08**	8175	0.10	8217	0.60	8175	0.13	8248	1.19	8191	0.47
2×300×8	**4365**	**0.05**	4386	1.30	4635	6.19	4553	5.08	4573	5.23	4608	5.75
2×300×10	**8215**	**0.19**	8234	0.24	8234	0.23	8234	0.23	8242	0.49	8234	0.23
2×400×3	**2739**	**0.61**	2788	3.24	2937	7.23	2930	7.02	2944	7.86	2937	7.23
2×400×5	**10 421**	**0.06**	10 433	0.15	10 451	0.29	10 435	0.18	10 455	0.45	10 446	0.27
2×400×8	**5323**	**0.06**	5325	0.09	5330	0.13	5328	0.09	5345	0.47	5330	0.13
2×400×10	**10 271**	**0.04**	10 277	0.08	10 291	0.19	10 277	0.11	10 278	0.35	10 289	0.19
2×500×3	**12 160**	**0.03**	12 166	0.05	12 168	0.07	12 166	0.05	12 207	0.40	12 166	0.05
2×500×5	**12 863**	**0.01**	12 864	0.01	12 873	0.08	12 864	0.01	12 884	0.16	12 864	0.02
2×500×8	**12 810**	**0.12**	12 805	0.49	12 981	1.37	12 907	0.90	12 862	0.67	12 957	1.29
2×500×10	**13 076**	**0.21**	13 066	0.40	13 259	1.48	13 195	1.10	13 312	2.06	13 259	1.48
3×100×3	**1797**	**0.50**	1807	0.85	1877	4.45	1809	1.21	1855	3.73	1833	2.20
3×100×5	1801	**0.24**	**1798**	0.31	1853	3.06	1812	0.98	1834	2.89	1829	1.90
3×100×8	**1956**	**0.55**	1974	1.80	2049	4.75	1982	1.68	2020	3.85	2014	3.29
3×100×10	2300	**0.75**	**2295**	1.00	2449	6.71	2333	2.38	2413	5.88	2397	4.95
3×200×3	**3318**	**0.28**	3338	0.64	3347	0.87	3336	0.55	3374	1.75	3336	0.58
3×200×5	**1227**	**0.9**	1240	1.76	1318	7.42	1292	5.77	1295	6.41	1311	7.04
3×200×8	**3455**	**0.28**	3483	0.99	3512	1.65	3485	0.93	3525	2.37	3490	1.12
3×200×10	**3445**	**0.05**	3448	0.15	3451	0.17	3447	0.06	3469	1.57	3448	0.14
3×300×3	**1702**	**0.14**	1705	0.46	1727	1.47	1712	0.79	1719	1.32	1717	1.06
3×300×5	**5603**	**0.10**	5619	0.29	5622	0.34	5617	0.25	5656	1.01	5619	0.30
3×300×8	**5058**	**0.04**	5061	0.17	5079	0.42	5061	0.06	5084	0.60	5061	0.08
3×300×10	**5222**	**0.10**	5238	0.51	5263	0.79	5237	0.34	5283	1.28	5242	0.47
3×400×3	**2253**	**0.12**	2262	0.95	2305	2.31	2302	2.23	2299	2.32	2305	2.31
3×400×5	**6834**	**0.05**	6846	0.20	6850	0.23	6846	0.18	6897	0.97	6848	0.23
3×400×8	**6985**	**0.05**	6998	0.23	7006	0.30	6997	0.18	7026	0.89	7002	0.28
3×400×10	7224	**0.32**	7225	0.32	7271	0.78	**7215**	**0.07**	7260	0.74	7227	0.21
3×500×3	**8279**	**0.03**	8288	0.14	8295	0.19	8287	0.10	8326	0.57	8287	0.11
3×500×5	**8607**	**0.07**	8623	0.26	8639	0.37	8621	0.20	8651	0.54	8623	0.20
3×500×8	**8887**	**0.08**	8904	0.20	8906	0.21	8899	0.14	8939	0.61	8900	0.16
3×500×10	**8613**	**0.12**	8633	0.26	8634	0.24	8630	0.22	8647	0.44	8633	0.24
4×100×3	**1171**	**0**	1173	0.22	1176	0.43	1172	0.09	1203	3.09	1173	0.26
4×100×5	**721**	**0.42**	731	1.58	749	3.88	731	1.58	757	6.77	746	3.63
4×100×8	**1387**	0.25	1394	0.87	1396	0.65	1389	**0.14**	1428	3.07	1392	0.59
4×100×10	**1385**	**1.36**	1426	3.61	1458	5.27	1416	2.44	1432	3.88	1428	3.21
4×200×3	**2512**	**0.07**	2519	0.39	2533	0.84	2518	0.25	2554	1.71	2520	0.36

(to be continued)

Table 8 Experimental results of all comparison algorithms when t= 5.

(continued)

$F\times J\times S$	AIG		CIG		DDE		EA		MN_IG		DPSO	
	MIN	RPI	MIN	RPI	MIN	RPI	MIN	RPI	MIN	RPI	MIN	RPI
4×200×5	**2478**	**0.36**	2491	0.52	2498	0.81	2490	0.48	2537	2.38	2493	0.62
4×200×8	**2707**	**0.22**	2725	0.66	2725	0.66	2724	0.64	2749	1.77	2725	0.66
4×200×10	**1607**	**0.36**	1633	2.73	1707	6.22	1656	4.17	1670	4.73	1687	5.33
4×300×3	**1266**	**0.30**	1284	1.75	1307	3.24	1304	3.08	1301	3.71	1307	3.24
4×300×5	**3843**	**0.27**	3865	0.97	3951	2.81	3911	2.16	3902	2.02	3950	2.81
4×300×8	**3935**	**0.13**	3946	0.38	3951	0.41	3945	0.29	4000	1.85	3951	0.41
4×300×10	**4224**	**0.08**	4238	0.58	4271	1.11	4234	0.40	4269	1.16	4243	0.66
4×400×3	**4938**	**0.19**	4954	0.38	4959	0.43	4952	0.30	5009	1.45	4953	0.33
4×400×5	**5183**	**0.11**	5196	0.39	5215	0.62	5193	0.20	5221	1.10	5212	0.61
4×400×8	**5389**	**0.18**	5441	1.11	5482	1.73	5450	1.32	5476	2.21	5482	1.73
4×400×10	**5400**	**0.06**	5420	0.46	5438	0.70	5414	0.35	5445	1.15	5438	0.70
4×500×3	**2168**	**0.18**	2186	1.77	2256	4.06	2242	3.71	2241	4.01	2249	3.99
4×500×5	**2339**	**0.50**	2391	3.51	2504	7.05	2483	6.58	2470	6.41	2503	7.05
4×500×8	**3665**	**0.47**	3717	2.31	3889	6.11	3851	5.27	3822	4.89	3876	5.95
4×500×10	**6467**	0.20	6467	0.18	6498	0.60	6459	**0.07**	6509	0.93	6495	0.59
5×100×3	**928**	**0.43**	936	1.14	941	1.40	932	0.54	981	6.12	937	1.14
5×100×5	**590**	**0.47**	596	1.63	613	3.90	599	2.07	616	7.15	613	3.90
5×100×8	**1124**	**0.16**	1134	1.39	1137	1.16	1124	0.23	1190	6.07	1134	1.05
5×100×10	**1369**	**0.20**	1374	1.43	1452	6.06	1403	3.11	1440	6.36	1440	5.55
5×200×3	**2090**	**0.02**	2093	0.30	2098	0.38	2091	0.05	2142	2.49	2092	0.24
5×200×5	**683**	**0.20**	686	1.38	713	4.39	709	3.98	718	5.97	713	4.39
5×200×8	**2298**	**0.25**	2311	0.64	2322	1.04	2305	0.39	2363	3.05	2311	0.71
5×200×10	**2236**	**0.22**	2249	0.87	2265	1.30	2246	0.48	2291	2.59	2249	0.81
5×300×3	**3053**	**0.16**	3061	0.32	3067	0.46	3058	0.17	3097	1.45	3061	0.31
5×300×5	**1667**	**0.38**	1677	1.03	1723	3.36	1714	3.00	1738	4.67	1723	3.36
5×300×8	**3270**	**0.20**	3286	0.69	3290	0.61	3282	0.42	3329	2.17	3290	0.61
5×300×10	**3369**	**0.16**	3418	1.76	3454	2.52	3421	1.78	3448	2.75	3454	2.52
5×400×3	**4040**	**0.14**	4051	0.32	4054	0.35	4049	0.22	4101	1.54	4051	0.32
5×400×5	**2221**	**0.14**	2249	2.06	2349	5.76	2328	5.01	2329	5.16	2349	5.76
5×400×8	**4088**	**0.14**	4106	0.57	4134	1.13	4101	0.40	4136	1.33	4125	1.08
5×400×10	**4284**	**0.06**	4295	0.43	4314	0.70	4291	0.28	4341	1.50	4305	0.57
5×500×3	**5160**	**0.16**	5202	1.13	5289	2.50	5253	2.09	5264	2.09	5289	2.50
5×500×5	**5020**	**0.02**	5023	0.15	5032	0.24	5021	0.04	5053	0.79	5025	0.14
5×500×8	**5256**	**0.04**	5264	0.21	5271	0.29	5258	0.05	5287	0.73	5265	0.22
5×500×10	**5404**	**0.09**	5405	0.21	5446	0.78	5414	0.25	5457	1.10	5446	0.78
Mean	**4350**	**0.23**	4365	0.86	4408	2	4382	1.31	4412	2.56	4397	1.74
Percentage (%)	–	–	0.34	73	1.3	89	0.73	82	1.4	91	1.07	87

Note: Percentage means the percentage of AIG better than other comparison algorithms.

the proposed AIG is suited to solve DBHFSP.

Meanwhile, to concretely illustrate the superiority of AIG, simulation experiments are conducted on four randomly selected instances with different scales, i.e., $2\times100\times3$, $3\times400\times10$, $4\times300\times8$, and $5\times500\times3$. Figure 12 shows the box plot drawn from the experimental results. From the box plots of the four instances, it is evident that AIG is better than the other algorithms significantly in terms of makespan. And the results obtained by AIG are very stable for three instances of $3\times400\times10$, $4\times300\times8$, and $5\times500\times3$, which indicates that the convergence of AIG is also outstanding.

In addition, to show the convergence ability of AIG more intuitively and enrich the experiments, we also randomly chose four test instances of different sizes,

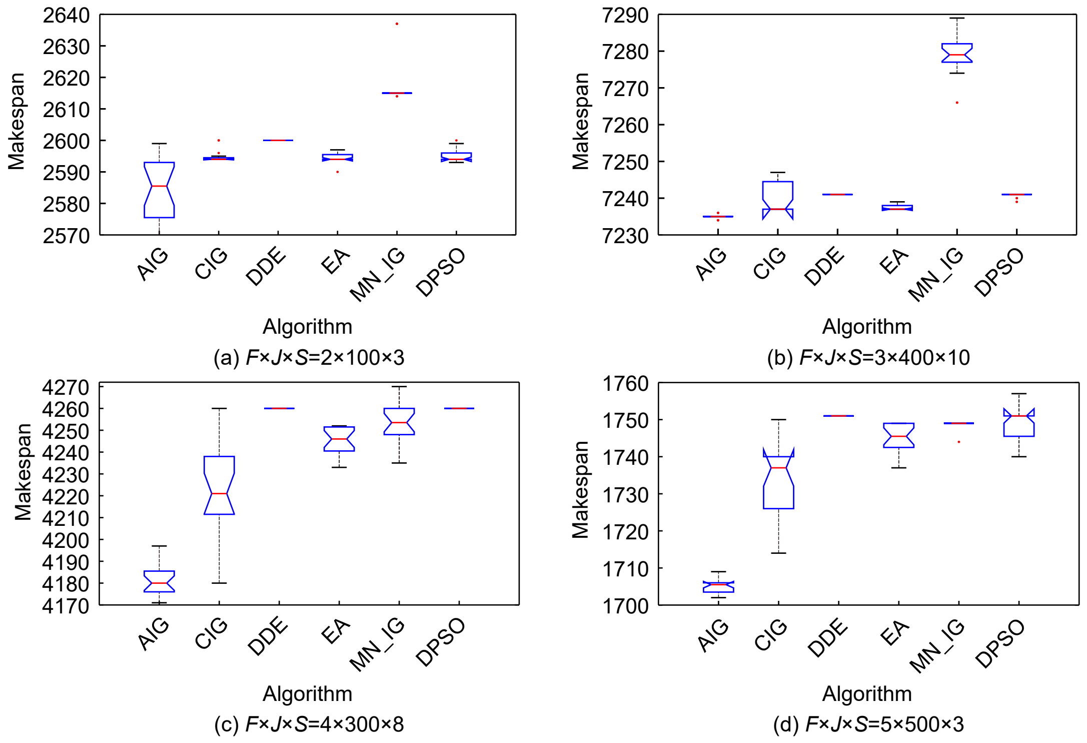

Fig. 12 Box plots of all comparison algorithms.

i.e., $2\times100\times3$, $3\times200\times5$, $4\times400\times8$, and $5\times300\times10$ to plot the evolutionary curves of all algorithms. Figure 13 shows the details of the evolutionary curves. The X-axis denotes the time taken by the algorithm to complete its evolutionary process, while the Y-axis represents the corresponding makespan generated by the algorithm. In Fig. 13, AIG obtains the lowest convergence curve with a good degree of convergence. AIG can explore more neighborhood structures by using inter-factory neighborhood search strategies and local search strategy, which increases the chances of finding better solutions. Moreover, AIG demonstrates the ability to find better solutions and show better convergence curves when solving different test instances. Therefore, we can believe that AIG has the ability to solve DBHFSP effectively.

5.7 Friedman test

The simulation results are analyzed to determine if there are significant differences in the overall distributions of the compared methods[49]. Initially, it is assumed that there are no significant differences between methods. If p value is less than 0.05, this assumption is considered rejected, and there are significant differences among the comparative methods. Conversely, if the p value is greater than or equal to 0.05, the assumption is deemed acceptable, indicating that no significant differences are found among the compared methods. Tables 9 and 10 provide the statistical results of 80 instances when $t=3$ and $t=5$, respectively. The Friedman test yielded a p value of 0.000, which is below the significance level $\alpha=0.050$. This demonstrates that compared algorithms have significant differences. AIG algorithm achieved the lowest rank values of 1.08 and 1.08 for $t=3$ and $t=5$, respectively, indicating its superior performance over other algorithms. Moreover, AIG exhibits the smallest maximum RPI value of 0.76, the smallest mean RPI value of 0.23, and the smallest standard deviation of 0.20, further highlighting its superior performance over other algorithms. To sum up, AIG is a very suitable algorithm for solving DBHFSP.

6 Conclusion and Future Work

Considering the constraints in actual production, this paper focuses on the DBHFSP. The primary objective is to optimize the makespan of all factories. The proposed approach is an AIG algorithm, which demonstrates effective performance in solving DBHFSP. First, we establish the MILP model of DBHFSP and validate its correctness using the Gurobi solver. Second, we design two cross-factory neighborhood search strategies to enhance collaboration between factories and explore a wider

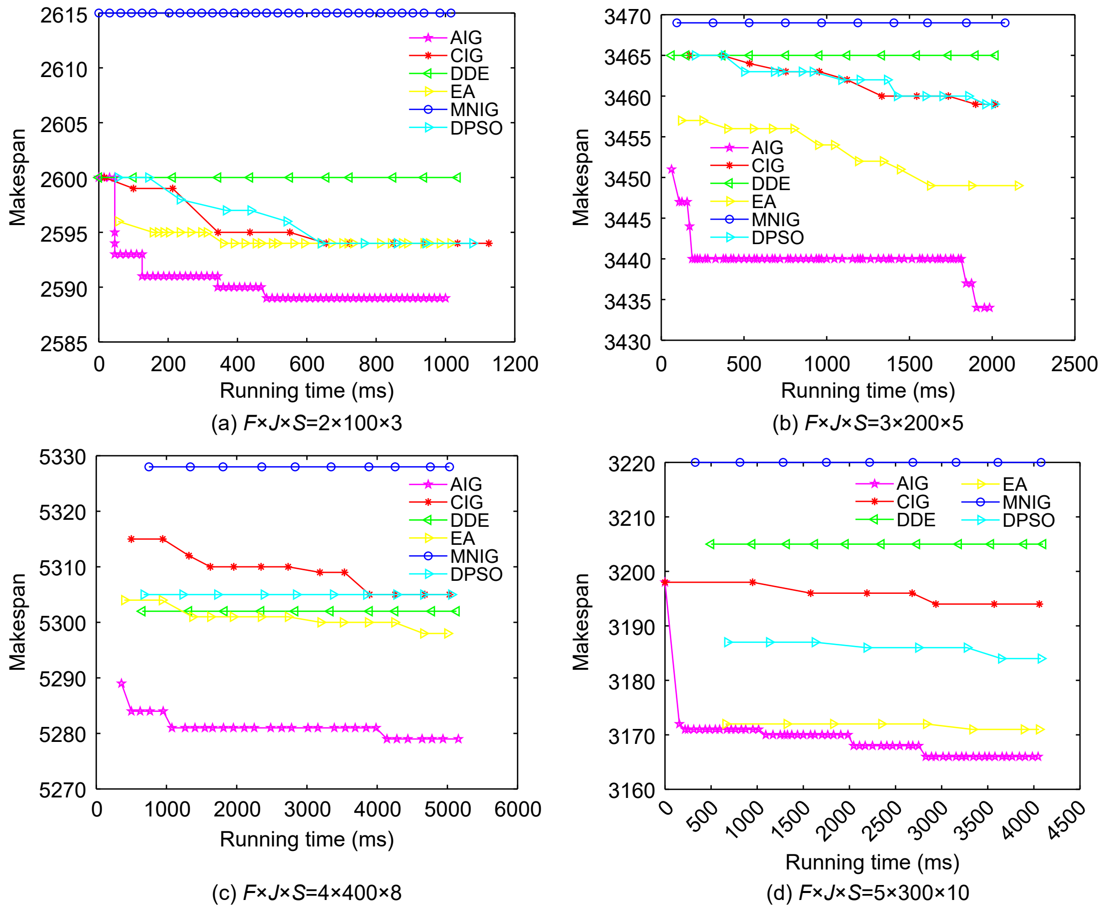

Fig. 13 Evolution curve of all the comparison algorithms.

Table 9 Statistical results obtained by Friedman test when *t*= 3 (*α*=0.050).

Algorithm	Rank	CN	Min	Max	Mean	Standard deviation
AIG	**1.08**	80	0.00	**0.76**	**0.23**	**0.20**
CIG	2.95	80	0.00	4.80	0.92	0.92
DDE	4.98	80	0.07	7.00	1.93	2.08
EA	2.46	80	0.00	6.81	1.28	1.68
MN_IG	5.55	80	0.16	7.62	2.48	1.92
DPSO	3.99	80	0.01	7.00	1.69	1.96
p-value	0.000	–	–	–	–	–

Note: CN is the number of instances.

Table 10 Statistical results obtained by Friedman test when *t*= 5 (*α*=0.050).

Algorithm	Rank	CN	Min	Max	Mean	Standard deviation
AIG	**1.08**	80	0.00	**1.357**	**0.23**	**0.23**
CIG	2.92	80	0.00	3.61	0.86	0.80
DDE	5.04	80	0.07	7.42	2.01	2.17
EA	2.48	80	0.00	7.02	1.31	1.70
MN_IG	5.56	80	0.16	7.86	2.56	1.99
DPSO	3.91	80	0.02	7.23	1.74	2.01
p-value	0.000	–	–	–	–	–

range of neighborhoods. Furthermore, the swap-based local perturbation strategy enables quick changes to the job sequence, thereby reducing computation time and increasing the chances of exploring the search space more thoroughly, ultimately leading to improved algorithmic performance. Finally, to mitigate premature convergence, we adopt two shaking strategies to increase the diversity of solutions. The experimental results in Section 5 demonstrate that AIG is significantly more effective in solving DBHFSP than the other five compared algorithms.

Although AIG shows promising performance in solving DBHFSP, there are still many challenges to be investigated. Firstly, considering energy consumption or electricity cost as the primary optimization objective can be a valuable extension in line with sustainability goals. Secondly, incorporating practical production constraints such as machine breakdowns and uncertain due dates would make the problem more realistic and applicable to real-world scenarios. Finally, we can explore multi-objective optimization methods to simultaneously optimize multiple objectives, such as energy consumption, delivery time, and total delay time.

Acknowledgment

This work was jointly supported by the National Natural Science Foundation of Shandong Province (No. ZR2023MF022); National Natural Science Foundation of China (Nos. 61973203, 62173216, and 62173356); and Guangyue Youth Scholar Innovation Talent Program Support from Liaocheng University (No. LCUGYTD2022-03).

References

[1] Z. Shao, W. Shao, and D. Pi, Effective heuristics and metaheuristics for the distributed fuzzy blocking flow-shop scheduling problem, *Swarm Evol. Comput.*, vol. 59, p. 100747, 2020.

[2] Q. -K. Pan, L. Gao, L. Wang, J. Liang, and X. -Y. Li, Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem, *Expert Syst. Appl.*, vol. 124, pp. 309–324, 2019.

[3] Y. -D. Kim, S. -O. Shim, B. Choi, and H. Hwang, Simplification methods for accelerating simulation-based real-time scheduling in a semiconductor wafer fabrication facility, *IEEE Trans. Semicond. Manuf.*, vol. 16, no. 2, pp. 290–298, 2003.

[4] M. K. Marichelvam, T. Prabaharan, and X. S. Yang, A discrete firefly algorithm for the multi-objective hybrid flowshop scheduling problems, *IEEE Trans. Evol. Comput.*, vol. 18, no. 2, pp. 301–305, 2014.

[5] K. Peng, Q. -K. Pan, L. Gao, B. Zhang, and X. Pang, An improved artificial bee colony algorithm for real-world hybrid flowshop rescheduling in steelmaking-refining-continuous casting process, *Comput. Ind. Eng.*, vol. 122, pp. 235–250, 2018.

[6] W. Shao, Z. Shao, and D. Pi, Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem, *Knowl. Based Syst.*, vol. 194, p. 105527, 2020.

[7] Y. Li, X. Li, L. Gao, and L. Meng, An improved artificial bee colony algorithm for distributed heterogeneous hybrid flowshop scheduling problem with sequence-dependent setup times, *Comput. Ind. Eng.*, vol. 147, p. 106638, 2020.

[8] J. -J. Wang and L. Wang, A bi-population cooperative memetic algorithm for distributed hybrid flow-shop scheduling, *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 5, no. 6, pp. 947–961, 2021.

[9] M. K. Marichelvam, M. Geetha, and Ö. Tosun, An improved particle swarm optimization algorithm to solve hybrid flowshop scheduling problems with the effect of human factors–a case study, *Comput. Oper. Res.*, vol. 114, p. 104812, 2020.

[10] G. Zhang, K. Xing, and F. Cao, Discrete differential evolution algorithm for distributed blocking flowshop scheduling with makespan criterion, *Eng. Appl. Artif. Intell.*, vol. 76, pp. 96–107, 2018.

[11] V. Fernandez-Viagas, P. Perez-Gonzalez, and J. M. Framinan, The distributed permutation flow shop to minimise the total flowtime, *Comput. Ind. Eng.*, vol. 118, pp. 464–477, 2018.

[12] R. Ruiz and T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, *Eur. J. Oper. Res.*, vol. 177, no. 3, pp. 2033–2049, 2007.

[13] R. Ruiz, Q. -K. Pan, and B. Naderi, Iterated greedy methods for the distributed permutation flowshop scheduling problem, *Omega*, vol. 83, pp. 213–222, 2019.

[14] H. Öztop, M. F. Tasgetiren, D. T. Eliiyi, and Q. -K. Pan, Metaheuristic algorithms for the hybrid flowshop scheduling problem, *Comput. Oper. Res.*, vol. 111, pp. 177–196, 2019.

[15] S. -Y. Wang and L. Wang, An estimation of distribution algorithm-based memetic algorithm for the distributed assembly permutation flow-shop scheduling problem, *IEEE Trans. Syst. Man Cybern.: Syst.*, vol. 46, no. 1, pp. 139–149, 2016.

[16] J. Wang, L. Wang, and J. Shen, A hybrid discrete cuckoo search for distributed permutation flowshop scheduling problem, in *Proc. 2016 IEEE Congress on Evolutionary Computation* (*CEC*), Vancouver, Canada, 2016, pp. 2240–2246.

[17] W. Shao, D. Pi, and Z. Shao, Optimization of makespan for the distributed no-wait flow shop scheduling problem with iterated greedy algorithms, *Knowl. Based Syst.*, vol. 137, pp. 163–181, 2017.

[18] F. Zhao, L. Zhang, J. Cao, and J. Tang, A cooperative water wave optimization algorithm with reinforcement learning for the distributed assembly no-idle flowshop scheduling problem, *Comput. Ind. Eng.*, vol. 153, p. 107082, 2021.

[19] J. -J. Wang and L. Wang, A cooperative memetic algorithm with feedback for the energy-aware distributed flow-shops with flexible assembly scheduling, *Comput.*

Ind. Eng., vol. 168, p. 108126, 2022.

[20] F. Zhao, T. Jiang, and L. Wang, A reinforcement learning driven cooperative meta-heuristic algorithm for energy-efficient distributed no-wait flow-shop scheduling with sequence-dependent setup time, *IEEE Trans. Ind. Inf.*, vol. 19, no. 7, pp. 8427–8440, 2023.

[21] Q. -K. Pan, L. Wang, J. -Q. Li, and J. -H. Duan, A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimisation, *Omega*, vol. 45, pp. 42–56, 2014.

[22] S. -Y. Wang, L. Wang, M. Liu, and Y. Xu, An enhanced estimation of distribution algorithm for solving hybrid flow-shop scheduling problem with identical parallel machines, *Int. J. Adv. Manuf. Technol*., vol. 68, no. 9, pp. 2043–2056, 2013.

[23] M. Li, D. Lei, and J. Cai, Two-level imperialist competitive algorithm for energy-efficient hybrid flow shop scheduling problem with relative importance of objectives, *Swarm Evol. Comput.*, vol. 49, pp. 34–43, 2019.

[24] X. Wu, Z. Cao, and S. Wu, Real-time hybrid flow shop scheduling approach in smart manufacturing environment, *Complex System Modeling and Simulation*, vol. 1, no. 4, pp. 335–350, 2021.

[25] K. -C. Ying and S. -W. Lin, Minimizing makespan for the distributed hybrid flowshop scheduling problem with multiprocessor tasks, *Expert Syst. Appl*., vol. 92, pp. 132–141, 2018.

[26] B. Xi and D. Lei, Q-learning-based teaching-learning optimization for distributed two-stage hybrid flow shop scheduling with fuzzy processing time, *Complex System Modeling and Simulation*, vol. 2, no. 2, pp. 113–129, 2022.

[27] J. Zheng, L. Wang, and J. -J. Wang, A cooperative coevolution algorithm for multi-objective fuzzy distributed hybrid flow shop, *Knowl. Based Syst*., vol. 194, p. 105536, 2020.

[28] E. Jiang, L. Wang, and J. Wang, Decomposition-based multi-objective optimization for energy-aware distributed hybrid flow shop scheduling with multiprocessor tasks, *Tsinghua Science and Technology*, vol. 26, no. 5, pp. 646–663, 2021.

[29] J. -J. Wang and L. Wang, A cooperative memetic algorithm with learning-based agent for energy-aware distributed hybrid flow-shop scheduling, *IEEE Trans. Evol. Comput*., vol. 26, no. 3, pp. 461–475, 2022.

[30] Z. Pan, D. Lei, and L. Wang, A knowledge-based two-population optimization algorithm for distributed energy-efficient parallel machines scheduling, *IEEE Trans. Cybern.*, vol. 52, no. 6, pp. 5051–5063, 2022.

[31] W. Shao, Z. Shao, and D. Pi, A network memetic algorithm for energy and labor-aware distributed heterogeneous hybrid flow shop scheduling problem, *Swarm Evol. Comput.*, vol. 75, p. 101190, 2022.

[32] Q. Zhang and Z. Yu, Population-based multi-layer iterated greedy algorithm for solving blocking flow shop scheduling problem, (in Chinese), *Comput. Integrated Manufacturing Syst.*, vol. 22, no. 10, pp. 2315–2322, 2016.

[33] Y. Zheng, G. Mo, and J. Zhang, Blocking flow line scheduling of panel block in shipbuilding, (in Chinese), *Comput. Integrated Manufacturing Syst.*, vol. 22, no. 10, pp. 2305–2314, 2016.

[34] V. Riahi, M. A. H. Newton, K. Su, and A. Sattar, Constraint guided accelerated search for mixed blocking permutation flowshop scheduling, *Comput. Oper. Res.*, vol. 102, pp. 102–120, 2019.

[35] Y. Han, J. Li, H. Sang, Y. Liu, K. Gao, and Q. Pan, Discrete evolutionary multi-objective optimization for energy-efficient blocking flow shop scheduling with setup time, *Appl. Soft Comput.*, vol. 93, p. 106343, 2020.

[36] S. Aqil and K. Allali, Two efficient nature inspired meta-heuristics solving blocking hybrid flow shop manufacturing problem, *Eng. Appl. Artif. Intell.*, vol. 100, p. 104196, 2021.

[37] X. Han, Y. Han, B. Zhang, H. Qin, J. Li, Y. Liu, and D. Gong, An effective iterative greedy algorithm for distributed blocking flowshop scheduling problem with balanced energy costs criterion, *Appl. Soft Comput.*, vol. 129, p. 109502, 2022.

[38] H. -X. Qin, Y. -Y. Han, B. Zhang, L. -L. Meng, Y. -P. Liu, Q. -K. Pan, and D. -W. Gong, An improved iterated greedy algorithm for the energy-efficient blocking hybrid flow shop scheduling problem, *Swarm Evol. Comput*., vol. 69, p. 100992, 2022.

[39] F. Zhao, H. Zhang, and L. Wang, A pareto-based discrete jaya algorithm for multiobjective carbon-efficient distributed blocking flow shop scheduling problem, *IEEE Trans. Ind. Inform.*, vol. 19, no. 8, pp. 8588–8599, 2023.

[40] F. Zhao, S. Di, and L. Wang, A hyperheuristic with Q-learning for the multiobjective energy-efficient distributed blocking flow shop scheduling problem, *IEEE Trans. Cybern.*, vol. 53, no. 5, pp. 3337–3350, 2023.

[41] V. Fernandez-Viagas, J. M. S. Valente, and J. M. Framinan, Iterated-greedy-based algorithms with beam search initialization for the permutation flowshop to minimise total tardiness, *Expert Syst. Appl.*, vol. 94, pp. 58–69, 2018.

[42] Y. Wang, X. Li, R. Ruiz, and S. Sui, An iterated greedy heuristic for mixed no-wait flowshop problems, *IEEE Trans. Cybern.*, vol. 48, no. 5, pp. 1553–1566, 2018.

[43] X. Han, Y. Han, Q. Chen, J. Li, H. Sang, Y. Liu, Q. Pan, and Y. Nojima, Distributed flow shop scheduling with sequence-dependent setup times using an improved iterated greedy algorithm, *Complex System Modeling and Simulation*, vol. 1, no. 3, pp. 198–217, 2021.

[44] S. Chen, Q. -K. Pan, L. Gao, and H. -Y. Sang, A population-based iterated greedy algorithm to minimize total flowtime for the distributed blocking flowshop scheduling problem, *Eng. Appl. Artif. Intell.*, vol. 104, p. 104375, 2021

[45] H. -X. Qin, Y. -Y. Han, Y. -P. Liu, J. -Q. Li, and Q. -K. Pan, A collaborative iterative greedy algorithm for the scheduling of distributed heterogeneous hybrid flow shop with blocking constraints, *Expert Syst. Appl*., vol. 201, p. 117256, 2022.

[46] L. Meng, K. Gao, Y. Ren, B. Zhang, H. Sang, and C. Zhang, Novel MILP and CP models for distributed hybrid flowshop scheduling problem with sequence-dependent

setup times, *Swarm Evol. Comput.*, vol. 71, p. 101058, 2022.

[47] B. Naderi and R. Ruiz, The distributed permutation flowshop scheduling problem, *Comput. Oper. Res.*, vol. 37, no. 4, pp. 754–768, 2010.

[48] B. Naderi and R. Ruiz, A scatter search algorithm for the distributed permutation flowshop scheduling problem, *Eur. J. Oper. Res.*, vol. 239, no. 2, pp. 323–334, 2014.

[49] J. -P. Huang, Q. -K. Pan, and L. Gao, An effective iterated greedy method for the distributed permutation flowshop scheduling problem with sequence-dependent setup times, *Swarm Evol. Comput.*, vol. 59, p. 100742, 2020.

Yong Wang received the BEng degree from Liaocheng University, Liaocheng, China in 2021. He is currently pursuing the MSc degree at Liaocheng University, Liaocheng, China. His current research interests include intelligent optimization and scheduling.

Yuyan Han received the MS degree from Liaocheng University, Liaocheng, China in 2012 and the PhD degree in control theory and control engineering from China University of Mining and Technology, Xuzhou, China in 2016. Since 2016, she has been an associate professor at the School of Computer Science, Liaocheng University. She has authored over 30 refereed papers. Her current research interests include evolutionary computation, multiobjective optimization, and flowshop scheduling.

Kaizhou Gao received the BSc degree in computer science and technology from Liaocheng University, Liaocheng, China, in 2005, the master degree in computer science and application from Yangzhou University, Yangzhou, China, in 2008, and the PhD degree in artificial intelligence and system engineering from Nanyang Technological University (NTU), Singapore in 2016. From 2008 to 2012, he was at the School of Computer Science, Liaocheng University, China. From 2012 to 2013, he was a research associate at the School of Electronic and Electrical Engineering, NTU, where he has been a research fellow from 2015 to 2018. He is currently an assistant professor at the Macau Institute of Systems Engineering, Macau University of Science and Technology. His research interests include intelligent computation, optimization, scheduling, and intelligent transportation. He has published over 100 refereed papers. He is an associate editor of *Swarm and Evolutionary Computation*, *IET Collaborative Intelligent Manufacturing*, and *The Chinese Journal of Artificial Intelligence*.

Yuting Wang received the master degree in computer software and theory from China University of Petroleum (Beijing) in 2005. Since 2013, he has been an associate professor at the School of Computer Science, Liaocheng University. He has published more than 20 papers. His current research interests include mathematical modeling, intelligent optimization algorithms, and software development technology.

Junqing Li received the master degree in computer science and technology from Shandong Economic University, Shandong, China in 2004 and the PhD degree in system engineering from Northeastern University, Shenyang, China in 2016. Since 2004, he has been with the School of Computer Science, Liaocheng University. Since 2017, he has been with the School of Computer Science, Shandong Normal University, where he became a professor in 2017. His current research interests include intelligent optimization and scheduling. He has authored more than 70 refereed papers.

Yusuke Nojima received the BS and MS degrees in mechanical engineering from Osaka Institute of Technology, Osaka, Japan in 1999 and 2001, respectively, and the PhD degree in system function science from Kobe University, Hyogo, Japan in 2004. Since 2004, he has been with Osaka Prefecture University, Osaka, Japan, where he is currently a professor at the Department of Computer Science and Intelligent Systems. His research interests include evolutionary fuzzy systems, evolutionary multiobjective optimization, and parallel distributed data mining. He was a guest editor for several special issues in international journals. He was a task force chair on Evolutionary Fuzzy Systems in Fuzzy Systems Technical Committee of IEEE Computational Intelligence Society. He was an associate editor of *IEEE Computational Intelligence Magazine* (2014−2019).