Expert Systems With Applications 201 (2022) 117256

Contents lists available at ScienceDirect

Expert Systems With Applications

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

A collaborative iterative greedy algorithm for the scheduling of distributed heterogeneous hybrid flow shop with blocking constraints

Hao-Xiang Qin [a], Yu-Yan Han [a,*], Yi-Ping Liu [b], Jun-Qing Li [c], Quan-Ke Pan [d], Xue-Han [a]

[a] School of Computer Science, Liaocheng University, Liaocheng 252059, PR China
[b] The College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, PR China
[c] School of Information and Engineering, Shandong Normal University, Jinan 250014, PR China
[d] School of Mechatronic Engineering and Automation, Shanghai University, Shanghai 200072, PR China

ARTICLE INFO

Keywords:
Blocking
Distributed heterogeneous hybrid flow shop problems
Collaborative iterative greedy algorithm
Energy efficiency

ABSTRACT

The hybrid flow shop and distributed flow shop problems have been extensively studied due to their wide industrial applications. However, the distributed heterogeneous hybrid flow shop problems (DHHFSP) with blocking constraints have not yet been well studied up to date. This paper considers how to arrange a variety of jobs to different heterogeneous factories, and each factory has a minimal makespan. The innovations of this paper lie in presenting a mathematical model of the DHHFSP with blocking constraints and designing a collaborative iterative greedy (CIG) algorithm. The CIG contains the problem-specific initialization strategy, the neighborhood search strategy, the destruction-reconstruction strategy, and the local intensification strategy. The *cross-factory* and *inner-factory* neighborhood search strategies based on two swap operators are adopted to reduce the blocking time. The local intensification strategy is developed to optimize the scheduling sequence of each factory. The proposed algorithm is empirically compared with five state-of-the-art algorithms on 60 different instance sets. The experimental results show that the proposed algorithm significantly outperforms the compared ones in terms of objective values and relative percentage deviation values.

1. Introduction

1.1. Distributed hybrid flowshop scheduling

With the development of economic globalization, the cooperation between different enterprises is getting increasingly close. This cooperative feature is more evident in the flow shop scheduling problems. The traditional centralized manufacturing mode has been difficult to flexibly satisfy the current market demand (Shao et al., 2020). To respond to the rapidly changing global markets, the distributed or multi-plant manufacturing mode is used to improve enterprises' resource utilization and production efficiency (Wang and Wang, 2020). As a variant of the traditional permutation flow shop scheduling problem (PFSP), the distributed permutation flow shop scheduling problem (DPFSP) is more complex than PFSP (Li et al., 2020).

To further improve the processing efficiency of the products, enterprises begin to embed an efficient production mode, i.e., the identical parallel machine scheduling into each factory. This mode is a flexible flow shop routing and is usually called hybrid flow shop scheduling (HFS) (Ztop et al., 2019; Fernandez-Viagas and Framinan, 2020), which can handle multiple jobs simultaneously as long as the machine loads are not exceeded (Liu, et al., 2019). In each stage of HFS, there are m ($m \geq 1$) machines in parallel that process the jobs. Some published literatures have proved that this scheduling method can process more jobs in a shorter time and reduce the production cost of enterprises (Wang et al., 2015; Feng et al., 2016). Up to now, the HFS mode has been successfully used to solve real-world problems, such as the transistor-liquid crystal displays (Choi et al., 2011), steelmaking and refining (Long et al., 2018; Peng et al., 2018), etc. Combining the respective advantages of DPFSP and HFS, a scheduling mode with more practical application, that is, the distributed hybrid flowshop scheduling (DHFS) came into being. Obviously, this cross-factory scheduling method with parallel production lines is more efficient.

1.2. Distributed heterogeneous hybrid flowshop scheduling with blocking constraints

In the actual processing environment, decision makers are likely to

* Corresponding author.
E-mail address: hanyuyan@lcu-cs.com (Y.-Y. Han).

https://doi.org/10.1016/j.eswa.2022.117256
Received 29 October 2021; Received in revised form 11 March 2022; Accepted 13 April 2022
Available online 18 April 2022

build factories with different processing capacity according to cost input, production planning, target groups or other reasons, so as to better adapt to the processing conditions in different environments. Although the problem becomes complex, it benefits reducing the production lead time and the work-in-process inventory, interim storage, and associated space requirements. In addition, the research is also motivated by a practical engineering case of the heterogeneous factories (Shao et al., 2021). In this problem, different factories have different numbers of processing machines in each stage, which leads to the heterogeneous phenomenon of factories. Therefore, this paper also considers the difference in the number of machines as the reason for the heterogeneity of factories.

Furthermore, in the actual processing factory, due to the storage space, process characteristics, or technical reasons (Ribas et al., 2011), when the number of jobs exceeds the machine's load capacity, they will be blocked in the current processing stage, resulting in invalid work of the machines and extension of completion time. The problem is also named distributed heterogeneous hybrid flow shop scheduling problem (DHHFSP) with blocking constraints. This also encourages us to design appropriate scheduling schemes to reduce the makespan caused by blocking.

1.3. Motivations

Considering the problem characteristics of DHHFSP with blocking constraints, some sub-problems should be considered, such as the sequence sort, factory allocation, machine selection, and blocking condition of jobs, need to be addressed simultaneously. Since the sub-problems are highly coupled, it is natural to design neighborhood-based metaheuristics that implement different strategies. As we know, metaheuristics are often used to solve flow shop scheduling problems, and have achieved good performance. As a kind of metaheuristic algorithm, Iterative Greedy (IG) algorithm has been used by many scholars to solve the related flow shop scheduling problems due to the small number of parameters, easy operation and simple process (Ruben et al., 2007). Different from other metaheuristic algorithms, this algorithm iterates only one solution in the whole process, which makes it better to explore the solution more deeply. Moreover, IG algorithm has strong local search ability due to its greedy insertion strategy, but it has also become a limiting factor of the algorithm, resulting in the reduction of the diversity of solutions.

Based on the above advantages and limit, this paper proposed a collaborative IG (CIG) algorithm to solve the DHHFSP with blocking constraints. In CIG, a new initialization scheme is used to assign different jobs to the heterogeneous factories. Then, two cross-factory neighborhood strategies are presented to reorder the jobs. Next, we perform the destruction and reconstruction operation on the job sequence in each factory and suggest a local intensification strategy to further reduce the makespan. Finally, these two solutions conduct the substitution operations on the inferior solution for the following loop of the iteration.

The contributions of this paper are as follows.

1) Formulate the DHHFSP with blocking constraints and set up a mixed-integer linear programming (MILP) model.
2) For reducing the influence of unnecessary blocking constraints on machining, the CIG algorithm is presented to solve the DHHFSP with blocking constraints, in which the Nawaz–Enscore–Ham-Increase (NEH_IN) initialization strategy is designed to allocate the jobs to the heterogeneous factories.
3) To further improve the global search ability of the algorithm and reduce the blocking conditions of the sequence, this paper present the *cross-factory* and *inner-factory* neighborhood search strategies, respectively, to cooperatively optimize the scheduling sequence.
4) To reinforce the exploration ability of CIG algorithm and reduce the completion time of jobs, this paper develops the local intensification strategies to adjust the order of the job sequence in each factory.

The rest of this paper is organized as follows. In Section 2, comprehensive literature reviews are presented. In Section 3, the MILP model of the DHHFSP with blocking constraints is formulated. Section 4 explains the proposed CIG algorithm, including the framework and the details of the strategies. Section 5 tests the parameters and strategies of the CIG algorithm and compares it to state of the arts algorithms. The conclusions of this paper and future research direction are provided in Section 6.

2. Literature review

Many intelligent algorithms are proposed to solve the DPFSP. Deng and Wang (2016) presented a competitive memetic algorithm to solve the multiobjective DPFSP. Fu et al. (2019) proposed a new multi-objective brain storm optimization algorithm to solve the DPFSP with total tardiness constraint. Considering the assembly of jobs, Lin et al. (2017) used the backtracking search hyper-heuristic for solving the DPFSP. In (Wang et al., 2020), a multiobjective whale swarm algorithm is suggested to optimize the objective of energy-efficient in DPFSP. Considering the sequence-dependent setup times, Huang et al. (2020) presented an effective IG algorithm to solve it. In (Fernandez-Viagas et al., 2018; Bargaoui et al., 2017), an iterative improvement algorithm and a chemical reaction optimization algorithm are proposed to minimize the total flowtime and makespan of the DPFSP, respectively. In addition, Pan et al. (2020) proposed a cooperative co-evolutionary algorithm for the DPFSP with group to minimize the makespan.

Since the HFSP was raised in 1973 (Salvador, 1973), a wide range of scholars have propsed many efficient intelligent optimization algorithms to solve it. Li et al. (2018) proposed the energy-aware multi-objective optimization algorithm for HFSP with the energy consumptions and makespan minimization. Given the setup energy consumption, Zhang et al. (2019) developed a multiobjective evolutionary algorithm with decomposition to solve the HFSP. Considering the worker constraint, the multiobjective evolutionary algorithm based on heuristic decoding was proposed for the HFSP (Han et al., 2020). In (Marichelvam et al., 2019), a discrete particle swarm optimization (DPSO) algorithm was designed to solve the HFSP with the human factors. In (Yu et al., 2018), the genetic algorithm (GA) is used for solving the HFSP with machine eligibility and unrelated machines. Ztop et al. (2019) presented four variants of IG algorithms and a variable block insertion heuristic to solve the HFSP to minimize the total flow time. Qin et al. (2019) developed the genetic programming-based scheduling algorithm to solve the HFSP with waiting time and batch processor constraints.

Up to now, the research on the combination of distributed factory and parallel machine scheduling is still rather limited. As far as we know, the existing algorithms about the DHFSP are proposed by Shao et al. (2020), in which the modeling and multi-neighborhood IG algorithm was proposed to optimize the makespan. Zheng et al. (2020) developed the cooperative coevolution algorithm for solving the multiobjective fuzzy DHFSP. Considering the multiprocessor tasks, Cai et al. (2020) designed a dynamic shuffled frog-leaping algorithm to solve the DHFSP. To minimize the makespan of the job sequence, Li et al. (2020) used the hybrid discrete artificial bee colony (DABC) algorithm to solve the DHFSP with deteriorating jobs. Given the distributed heterogeneous factories, an improved artificial bee colony algorithm is proposed to solve the DHHFSP with sequence-dependent setup times (Li et al., 2020).

In view of these blocking constraints, many efficient algorithms have been proposed to address the related problems. Riahi et al. (2017) developed a scatter search for the mixed BFSP. Han et al. (2019) designed the evolutionary multiobjective robust scheduling algorithm to

solve the blocking lot-streaming FSP. Ribas et al. (2015) designed a DABC algorithm to solve the blocking FSP (BFSP) with the objective of total flowtime minimization. Later, the IG algorithm was proposed to solve the total tardiness parallel BFSP (Ribas et al., 2019). Zhang et al. (2018) utilized the discrete differential evolution (DDE) algorithm to optimize the makespan of the distributed blocking flow shop scheduling problem (DBFSP). For the distributed fuzzy BFSP, some effective heuristics and metaheuristics have been developed to solve this problem (Shao et al., 2020). In (Zhao et al., 2020), to minimize the makespan, an ensemble DDE is presented for solving the DBFSP. Considering the setup time of the BFSP, Han et al. (2020) proposed the discrete evolutionary multiobjective optimization algorithm to optimize the makespan and energy consumption.

As we can see, the DFSP, the HFSP, and the BFSP have received much attention in recent years. However, the DHFSP under the blocking and heterogeneous environment has not been addressed. Due to its practical relevance, it is worthwhile to develop effective and efficient algorithms for the above problem.

3. Problem formulation

In DHHFSP with blocking constraints, there are F heterogeneous factories, and each factory has the same number of processing stages. In one factory, at least one stage has identical, unrelated parallel machines. In stage s of the factory, there are m ($m \geq 1$) machines with no buffers to store the processed jobs. Each machine is available and without breakdowns. A series of n jobs have to be processed on one of these F factories. Each job is processed orderly in the sequential stages. Once the job is determined to be processed in one machine of the stage, it can not be interrupted. In a word, the DHHFSP with blocking constraints consists of three subproblems, i.e., scheduling and sorting of jobs, selecting factories for jobs, and assigning machines to jobs. The optimization objective of the DHHFSP with blocking constraints is the makespan. Based on these definitions and literature (Wang and Wang, 2020), this paper gives the MILP model of the DHHFSP with blocking constraints.

Parameters and sets:

J: The number of jobs.

F: The number of factories.

S: The number of stages in each factory.

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

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

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

m: The index of machines at each stage.

$m_{f,s}$: The number of parallel machines at stage s in factory$f$.

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

Decision variables:

$C_{\max}$: The makespan of the sequence.

$B_{j,s}$: The beginning time of job j at stage$s$.

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

$D_{j,s}$: The departure time of job j at stage$s$.

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

$y_{f,s,j,m}$: Decision variables, 1 if the 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.

Objective:

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

Constraints:

Table 1
Processing time of different jobs in each stage.

Job	Stage1	Stage2
1	5	15
2	7	7
3	12	3
4	3	5
5	3	3
6	6	13

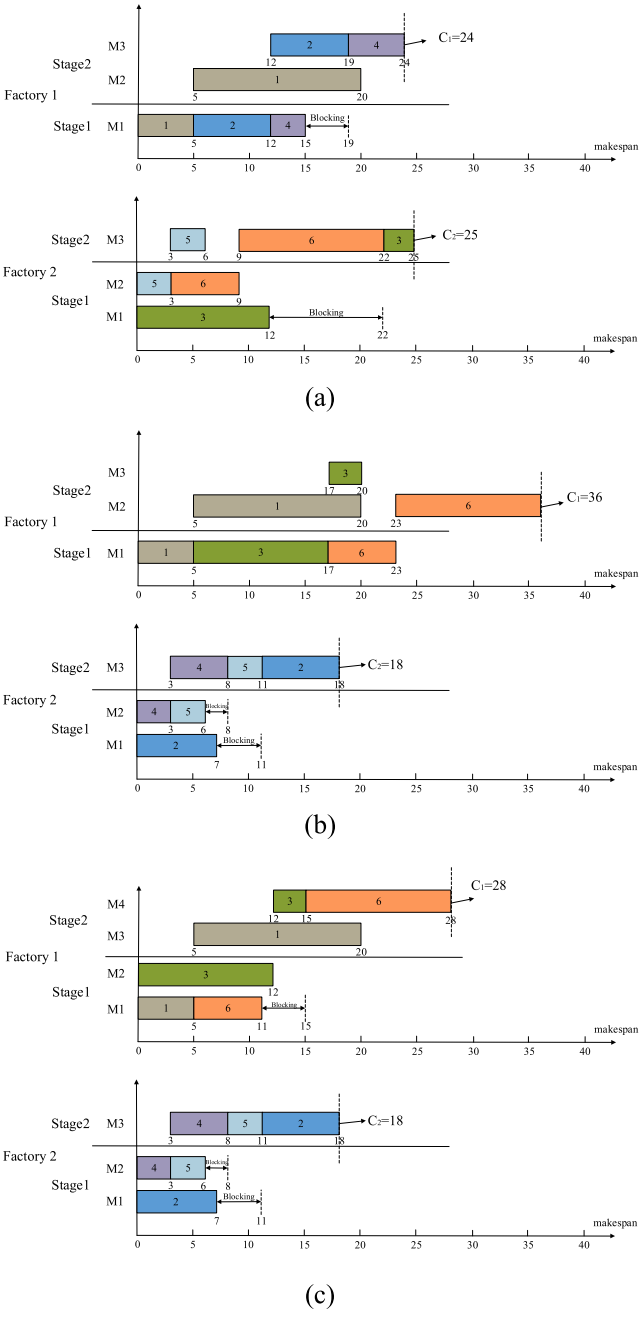


Fig. 1. Gantt charts of different scenes. (a) and (b) are the comparison of the different scheduling sequence under same factory configurations. (b) and (c) are the comparison of the same scheduling sequence under the different machine configurations of each factory (heterogeneous factories).

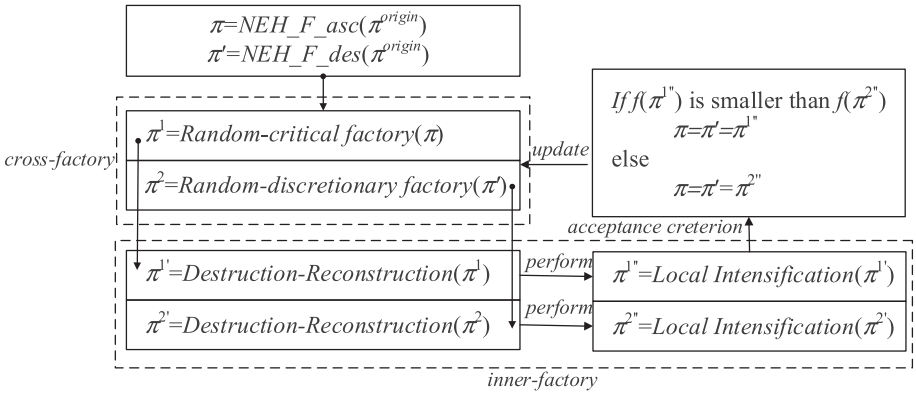


Fig. 2. collaboration process between cross-factory and inner-factory.

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

$$\sum_{m=1}^{m_{f,s}} y_{f,s,j,m} = x_{f,j}, \forall f, j, s \tag{3}$$

$$B_{j,s} \geqslant 0, \forall j, s \tag{4}$$

$$C_{j,s} = B_{j,s} + p_{j,s}, \forall j, s \tag{5}$$

$$z_{f,s,j,j'} + z_{f,s,j',j} \leqslant 1, \forall f, s, j \neq j' \tag{6}$$

$$z_{f,s,j,j'} + z_{f,s,j',j} \geqslant y_{f,s,j,m} + y_{f,s,j',m} - 1, \quad \forall f, s, j' \neq j, m \in \{1, 2..., m_{f,s}\} \tag{7}$$

$$B_{j',s} - D_{j,s} + U \cdot (3 - y_{f,s,j,m} - y_{f,s,j',m} - z_{f,s,j,j'}) \geqslant 0, \quad \forall j \neq j', f, s, m \in \backslash\{1, 2..., m_{f,s}\backslash\} \tag{8}$$

$$D_{j,s} = B_{j,s+1}, \forall j, s \in \{1, ..., S-1\} \tag{9}$$

$$C_{j,s} <= D_{j,s}, \forall j, s \tag{10}$$

$$C_{\max} >= D_{j,S}, \forall j \tag{11}$$

Equation (1) is the optimization objective of the problem. Constraint (2) determines that each job can only be assigned to one factory. Constraint (3) ensures that a job that allocated to one factory must be processed by one machine at each stage. Constraint (4) describes that the start time of the job at each stage is not less than 0. constraint (5) shows that the completion time of one job at each stage equals to the sum of its start time and processing time. Constraints (6–7) ensure that the machine can only operate one job at one time and the job can only be processed by one machine. Constraint (8) indicates that the job can be processed only when its previous job is completed in the same machine. Constraint (9) defines that, except the last stage, the start time of the job equals to the departure time of its previous stage, and Constraint (10) expresses that the departure time of the job is not less than its completion time at the same stage. Constraint (11) ensures that the makespan is not less than all of the jobs' departure time at the last stage.

To help readers understand the impact caused by different sequence and allocation of jobs more intuitively, this paper gives the Gantt charts with two heterogeneous factories, two stages, and six jobs to exemplify the process. The processing time of each job is given in Table 1. In factory 1, there is one machine in stage 1 and two parallel machines in stage 2. In factory 2, two parallel machines are set in stage 1, and one machine is set in stage 2. All the jobs have the same processing time in the same stage.

In Fig. 1 (a), the job sequence is π = {1, 2, 4; 3, 5, 6}, which illustrates the jobs 1, 2, and 4 are assigned to process sequentially in the first stage of factory 1, while jobs 3, 5, and 6 are processed in the first stage of factory 2. In Fig. 1 (b), the job sequence is π = {1, 3, 6; 2, 4, 5}, which shows that jobs 1, 3, and 6 are allocated to process sequentially in the first stage of factory 1, while jobs 2, 4, and 5 are processed sequentially in the first stage of factory 2. As shown from Fig. 1 (a) and (b), the different scheduling sequences make the makespan objective different under the same environment. the makespan of sequence {1, 2, 4; 3, 5, 6} and {1, 3, 6; 2, 4, 5} are 25 and 36, respectively. The gap is quite large between these two scheduling schemes. Moreover, with the continuous expansion of jobs' scale, the gap (sum of time interval which machines are in idle and blocking state) may expand due to a bad arrangement order. Therefore, it is essential to design reasonable scheduling strategies to help enterprises reduce makespan.

The scheduling sequence of Fig. 1 (c) is the same as Fig. 1(b). However, the machine configuration is different. That is, the factories are heterogeneous. By comparing Fig. 1 (b-c), we can observe that although the processing sequence is the same, the completion time is different due to the influence of machine configuration in heterogeneous factories. The completion time difference between the two processing factories is 8. We can see that the configuration of heterogeneous factories has a significant impact on the completion time.

4. CIG algorithm for DHHFSP with blocking constraints

In this section, we first describe the proposed CIG algorithm framework (see in Algorithm 1). CIG consists of the initialization, the neighborhood search strategies, the destruction-reconstruction, and the local intensification strategy. First, based on the characteristics of multiple factories, the Nawaz-Enscore-Ham with ascending order (*NEH_F_asc*) and descending order (*NEH_F_des*) initialization methods are utilized, respectively, to allocate the jobs to factories. Then, the *cross-factory* and *inner-factory* neighborhood search strategies are proposed to explore the globally better solutions. Next, the destruction-reconstruction method is adopted to improve the ordering of jobs. Finally, the local intensification strategy is presented to increase the quality of the solution further. The collaboration process between cross-factory and inner-factory is shown in Fig. 2.

Algorithm 1 The framework of the CIG algorithm

Input: $\pi = (1, 2, 3..., n)$, parameter d
Output: π^{best} and makespan
Begin:
$\pi = NEH\_F\_asc(\pi^{origin}), \pi' = NEH\_F\_des(\pi^{origin})$
While the termination criterion is not satisfied **do**
$\pi^1 \leftarrow Random\_critical\ factory\ swap(\pi)$
$\pi^2 \leftarrow Random\_discretionary\ factory\ swap(\pi')$ % *Cross-factory neighborhood search*
For $f = 1$ **to** F % *Inner-factory neighborhood search*
$\pi^{1'} \leftarrow Destruction\_Reconstruction(\pi^1, d)$
$\pi^{2'} \leftarrow Destruction\_Reconstruction(\pi^2, d)$

(*continued on next page*)

(*continued*)

Algorithm 1 The framework of the CIG algorithm

```
    π^{1'} ← Local Intensification (π^{1'})
    π^{2'} ← Local Intensification (π^{2'})
    If f(π^{2'}) < f(π^{1'}) then
      π = π^{1'} = π^{2'}
    Else
      π = π^{2'} = π^{1'}
    End
  End
  If f(π) < f(π^best) then
    π^best = π
  makespan = f(π^best)
  End
End
End
```

Notably, the key issues of the DHHFSP are how to allocate jobs to appropriate factories and how to generate the scheduling sequence of operations on machines with minimal makespan. Due to the above two issues being coupled, it is necessary to establish a collaboration between cross-factory and inner-factory. The above motivate us to implement a collaborative strategy. First, the different neighborhood search strategies based on different factories are executed, which changes or optimizes the allocating jobs to appropriate factories. Then, based on the above allocation, the destruction-reconstruction and local intensification strategies are performed to improve the quality of solution for each factory. Next, the current best solution obtained by an acceptance criterion is used to update or influence the neighborhood solution and continue to participate in the next cycle.

4.1. Encoding and decoding

A good encoding–decoding method can help us obtain a reasonable and effective scheduling sequence, especially for the complex combinatorial optimization problem. In DHHFSP with blocking constraints, the scheduling sequence and the machine selection should be determined in each factory. The solution is denoted as $\pi = \{\pi^1; \pi^2; \ldots; \pi^F\} = \{\pi^1_1, \pi^1_2, \ldots, \pi^1_{sum1}; \pi^2_{sum1+1}, \ldots, \pi^2_{sum2}; \ldots; \pi^F_{sumF-1}, \ldots, \pi^F_{sumF}\}$, π^f represents the job sequence which is allocated to factory f, π^f_j represents the j-th job processing in factory f, $f = 1, 2, \ldots, F$. *sumf* represents the sum of all jobs in the first f factories. Each job must be allocated to only one factory to process. As can be seen from Fig. 1 (a), sequence π is $\{\pi^1; \pi^2\}$.$\pi^1 = \{\pi^1_1, \pi^1_2, \pi^1_3\} = \{1,2,4\}$,$\pi^2 = \{\pi^2_4, \pi^2_5, \pi^2_6\} = \{3,5,6\}$, that is jobs 1, 2, and 4 are processed sequentially in factory 1, the jobs 3, 5, and 6 are processed in factory 2.

For each factory, the processing of decoding adopts the first-input–output rule to determine the sequence according to the complement time of the jobs, and the first available machine rule to assign machine to the job (see in Fig. 1). The details refer to (Qin et al., 2021a; Pan et al., 2014).

4.2. The initialization methods

The existence of blocking constraints affects the completion time of the job sequence. For the CIG algorithm, the quality of the initial solution directly influences the makespan of the job sequence. The *NEH_F* method has been proved to be an effective initialization strategy when solving the DPFSP based on *NEH* (Huang et al., 2020). Based on the research, this paper designs an initial method with ascending order of processing time, named *NEH_F_asc* to complete the initialization of the solution. To improve the diversity and the convergence of the solution, *NEH_F_des* and *NEH_F_asc* are simultaneously chosen to generate the two initialization solutions. Here, the only difference between *NEH_F_des* and *NEH_F_asc* is that the initial solution π is generated by sorting jobs according to their descreasing total processing time. In *NEH_F_asc*, we firstly arrange the job sequence in ascending order. Secondly, extract the same number of jobs as the number of factories, and then place these jobs into each factory one by one in order to ensure that each factory can have one job in it. For the rest jobs of the sequence π, (1) we take out the first job and try to insert it in all positions of all factories, and the position with the minimal makespan is selected. (2) the above first job is deleted from π. (3) Repeated the above steps (1) and (2), until all the jobs are assigned into factory. At this time, we can guarantee the makespan of each factory is small as much as possible, but not guarantee the number of jobs in all factories is the same or average. The pseudocode of *NEH_F_asc* is shown in Algorithm 2.

Algorithm 2 NEH_F_asc initialization method

```
Input: π^origin = (1, 2, 3...n)
Output: π
Begin:
  π = SortAscending(Σ_{s=1}^{S} p_{j,s}), j = 1,2...n
  For f = 1 to F
    π^f = π_f % Assign the f th job to factory f one by one.
  End
  For j = F + 1 to n
    For f = 1 to F
      π^{f_temp} ← insert ith position to |π^f| extract(π_j)
    π^f = argmin_{i=1}^{|π^f|} f(π^{f_temp})
    End
  End
End
```

4.3. The neighborhood search strategy

Generally speaking, the IG algorithm iterates a solution continuously, which will improve its local search ability, but its performance in global search ability is not outstanding. The time complexity of the swap-based strategy is less than that of insert-based strategy ($O(n^2)$ vs. $O(n^3)$). For example, we perform the swap operations on n jobs. Each job exchanges positions with all other jobs, so there are total n-1 exchanges. The total time complexity is $O(n^2)$. When we execute the insertion operations on n jobs. Every job first selects the positions, there are n-1 positions to be selected. Once a position p is selected, there are n-p jobs should be moved. If all the n jobs should execute the insertion operation, the total time complexity is $O(n^3)$, which is larger than the time complexity of swap operation.

Thus, in the proposed algorithm, the neighborhood search strategy based on swap is proposed for the two solutions to enhance the global search ability of the IG algorithm, improve the efficiency of the job sequencing and reduce the makespan caused by blocking constraints. In addition, for the distributed heterogeneous characteristic of DHHFSP, there are two kinds of factories, i.e., critical and non-critical factories. It is a lack of the neighborhood disturbing strategies of assigning jobs to the two kinds of factories. Thus, in our neighborhood search, *random-critical factory* disturbing and *random-discretionary factory* disturbing are considered.

1) *Random-critical factory disturbing*: (a) The sequence in factory f with the maximum completion time (f_{max}) is selected, then a random factory (f_{random}) is selected. (b) Randomly select a job from each of the two factories and swap them. (c) If the makespan of the new sequence obtained is less than the original sequence, the new one replaces the old one. (d) Repeat the steps (a-c) until the termination criterion is met.
2) *Random-discretionary factory disturbing*: (a) Two job sequences are randomly selected from different factories, respectively. (b) Randomly select a job from each of the selected factories and swap them. (c) If the makespan of new sequence is less than the original

sequence, it becomes the new sequence. (d) Go to step (a) until the termination criterion is satisfied.

The framework of the neighborhood search strategy is given in Algorithm 3.

Algorithm 3 The neighborhood search strategy

Input: π, π'
Output: π^1, π^2
Begin:
$\pi^{temp1} = \pi, \pi^{temp2} = \pi'$
π^{temp1}: ***Random-critical factory disturbing:***
$f_{\max} = \text{argmin}_{f=1}^{F} f(\pi^{temp1})$% Find the factory with the maximum makespan
do {
$f_{randonm} = rand()\%F$
} **While** ($f_{randonm} == f_{\max}$)
For $j = 1$ **to** n^2
$pos1 = rand()\%|\pi^{temp1\_f_{\max}}|, pos2 = rand()\%|\pi^{temp1\_f_{randonm}}|$
$\pi^{temp1\_new} = swap(\pi^{temp1}_{pos1}, \pi^{temp1}_{pos2})$
If $f(\pi^{temp1\_new}) < f(\pi^{temp1})$ **then**
$\pi^{temp1} = \pi^{temp1\_new}$
Else
$\pi^{temp1\_new} = \pi^{temp1}$
End
End
If $f(\pi^{temp1}) < f(\pi)$ **then**
$\pi = \pi^{temp1}$
End
π^{temp2}: ***Random-discretionary factory disturbing:***
$f_{randonm1} = rand()\%F$
do {
$f_{randonm2} = rand()\%F$
} **While** ($f_{randonm1} == f_{randonm2}$)
For $j = 1$ **to** n^2
$pos3 = rand()\%|\pi^{temp2\_f_{randonm1}}|, pos4 = rand()\%|\pi^{temp2\_f_{randonm2}}|$
$\pi^{temp2\_new} = swap(\pi^{temp2}_{pos3}, \pi^{temp2}_{pos4})$
If $f(\pi^{temp2\_new}) < f(\pi^{temp2})$ **then**
$\pi^{temp2} = \pi^{temp2\_new}$
Else
$\pi^{temp2\_new} = \pi^{temp2}$
End
End
If $f(\pi^{temp2}) < f(\pi^2)$ **then**
$\pi^2 = \pi^{temp2}$
End
End

4.4. The destruction and reconstruction strategy

In the traditional IG algorithm, the destruction and reconstruction strategy can effectively explore local neighborhoods more in-depth. This operation has a direct impact on finding a better scheduling sequence. To enhance the local search ability of the proposed algorithm, we introduce the destruction and reconstruction strategy to reduce the makespan caused by the blocking constraints. The steps are as follows (see in Algorithm 4): (1) In a factory, we randomly extract d jobs from the current job sequence. (2) Insert the first extracted job into all positions of the remaining job sequence. (3) The minimum makespan of the sequence is selected to be the current remaining sequence. (4) Repeat the steps (2–3) until the d jobs are all inserted into the job sequence.

The details of the destruction and reconstruction strategy are shown in Algorithm 4.

Algorithm 4 The destruction and reconstruction strategy

Input: π^1, π^2, *parameter* d
Output: $\pi^{1'}, \pi^{2'}$
Begin:
For $f = 1$ **to** F
$\pi^{temp1\_f} = \pi^{1\_f}$ %π^{1_f}: *The job sequence of factory f*
$U_{i=1}^{d} = extract(\pi^{temp1\_f})$
For $j = 1$ **to** d

(*continued on next column*)

(*continued*)

Algorithm 4 The destruction and reconstruction strategy

$\pi^{temp1\_f'} = \pi^{temp1\_f} \setminus U_j, \pi^{temp1\_f'}{}^{insert\ ith\ posistion}_{i=1\ \ to\ \ |\pi^{temp1\_f}|} \leftarrow U_j$
$\pi^{temp1\_f} = \text{argmin}_{i=1}^{|\pi^{temp1\_f}|} f(\pi^{temp1\_f'})$
End
If $f(\pi^{temp1\_f}) < f(\pi^{1\_f})$ **then**
$\pi^{1'\_f} = \pi^{temp1\_f}$
End
End
π^2 *performs the same process as* π^1
End

4.5. The local intensification strategy

The blocking constraints prolong the completion time of the job sequence, which reduces the production efficiency of the enterprise. To further improve the algorithm's performance and reduce the blocking time, we develop a local intensification strategy based on swap to solve the DHHFSP with blocking constraints. The local intensification strategy contains two kinds of swap operators: random swap and sequential swap. Based on the sequence $\pi^{1'}$ and $\pi^{2'}$ obtained from the destruction and reconstruction strategy.

The details of the proposed strategy are given as follows:

1) *Random swap*: (a) Randomly select two different jobs in the same sequence. (b) Swap the positions of the two jobs. (c) If the completion time of the sequence is reduced, the new sequence replaces the old sequence. (d) Repeat steps (a-c) until the termination criterion is met.
2) *Sequential swap*: Set $i = 1, j = 1$. (a) Select the ith job in the sequence. (b) Select the jth job in the same sequence. (c) Swap the ith job and the jth job. If the job sequence is improved, it replaces the original sequence. (d) j++, repeat the step (c), until $j == n$. (e) i++, skip to step (b), until $i == n$.

To make it easier for readers to understand, we give the pseudo-code of the strategy. The process is shown in Algorithm 5.

Algorithm 5 The local intensification strategy

Input: $\pi^{1'}, \pi^{2'}$
Output: $\pi^{1''}, \pi^{2''}$
Begin:
$r = rand()\%2$
Case $r == 0$:
Random swap:
For $f = 1$ **to** F
For $j = 1$ **to** n^2
$\pi^{temp1\_f} = \pi^{1'\_f}$ %$\pi^{1'\_f}$: *The job sequence of factory f*
For $j = 1$ **to** n^2
$p_1 = rand()\%|\pi^{temp1\_f}|$
do {
$p_2 = rand()\%|\pi^{temp1\_f}|$
} **While** ($p_1 == p_2$)
$Swap(\pi^{temp1\_f}_{p_1}, \pi^{temp1\_f}_{p_2})$
If $f(\pi^{temp1\_f}) < f(\pi^{1'\_f})$ **then**
$\pi^{1''\_f} = \pi^{temp1\_f}$
End
End
End
End
Case $== 1$:
Sequential swap:
For $f = 1$ **to** F
For $i = 1$ **to** $|\pi^{temp1\_f}|$
$\pi^{temp1\_f} = \pi^{1'\_f}$
For $j = 1$ **to** $|\pi^{temp1\_f}|$
$Swap(\pi^{temp1\_f}_i, \pi^{temp1\_f}_j)$

(*continued on next page*)

(*continued*)

Algorithm 5 The local intensification strategy

If $f(\pi^{temp1\_f}) < f(\pi^{1'\_f})$ **then**
$\pi^{1'\_f} = \pi^{temp1\_f}$
End
End
End
End
$\pi^{2'}$ *performs the same process as* $\pi^{1'}$
End

4.6. The time complexity of CIG

Assume that there are n jobs, f factories. The numbers of jobs and the stages in each factory are n/f and s, respectively. The computational complexity of the whole CGI algorithm mainly consists of initialization, neighborhood search, destruction & reconstruction, and local intensification strategies. The time complexity of the *initialization strategy* is $O(n*f*s*n/f)$ that approaches $O(n^2s)$. In while loop, we assume that the number of iterations of *neighborhood search strategy* is w_1, the time complexity of this strategy is $O(w_1n^2)$. The time complexities of the *destruction and reconstruction* strategy and the *local intensification strategy* are $O(w_2*d*f*s*n/f)$ and $O(w_3(n^4 + s\,n^3/f^2))$, respectively, where w_2 and w_3 are the numbers of iterations of *destruction and reconstruction* and *local intensification strategy*, respectively. Thus, for the whole CGI algorithm, the time complexity of the CIG is $O(n(s\,n + w_1\,n + w_2ds + w_3(n^4 + s\,n^2/f^2))$ that approaches $O(n^5)$. In addition, in the proposed CIG, we propose the new strategies based on swap to replace the greedy insertion. Compared with other IG algorithms, these strategies reduce the time complexity. Thus, CIG can execute more times under the same termination condition. Compared with other swarm intelligent comparison algorithms, the number of solutions of this algorithm is far less than that of these comparison algorithms. Therefore, in the whole iterative process, we only need to iterate the two solutions at a deeper level, rather than focusing on many solutions.

5. Simulation experiments and analysis

The experimental simulation environment of all algorithms is a PC with 2.60 GHZ Intel Core i7 Pentium processor with16 GB RAM. The algorithms are coded by Visual Studio 2019C++ in Microsoft Windows 10 operating system. Referring to the literature (Zhang et al., 2017), this paper adopts the same elapsed CPU running time as the termination criterion of all comparison algorithms.

5.1. Test data

To more systematically verify the performance of all algorithms, we give comprehensive data. The number of job set is $n\in\{100, 200, 300, 400, 500\}$, the factory set is $f\in\{2, 3, 4, 5\}$, the stage set is $S\in\{5, 8, 10\}$. The number of jobs, factories, and stages constitute the instance scale$n \times f \times s$. Therefore, the benchmark set consists of 60 different scale instances. We refer to the literature (Huang et al., 2020; Taillard, 1993) to set the values of the termination criterion, noted as the$TimeLimit = \omega \times n \times f \times S$, ω is a parameter that controls the length of running time, in this paper, we set the value of ω as 5 and 10 to control the length of running time. The processing time of the jobs is generated randomly within the range [1, 99] (Qin et al., 2021b). These experimental settings are general in the flow shop scheduling research, the source code and test data in this paper can be found in https://github.com/wangyuting9836/DBHFSP. In addition, the number of processing machines is randomly generated from the range [1, 5]. We utilize the relative percentage increase (RPI) to estimate the difference between the current value obtained and the best value. The RPI is calculated as follows:

$$RPI = (c_i - c_{best})/c_{best} \times 100 \tag{12}$$

where RPI is the relative percentage increase, c_i is the average value of makespan of a instance obtained by an algorithm independently performed several times, and the c_{best} is the minimum value of a instance obtained by all the compared algorithms independently performed several times. We first calculate RPI of each instance, and then compute the average values of RPI for all the instances. It is notable that the range of RPI value obtained by the different scale, respectively, has a little difference according to the simulation experimental results. Thus, in the following tables, the "mean" values that is the average values of RPI of all the instances, can be calculated to test the overall performance for different factory configurations.

Table 2
RPI and Makespan values for the MILP Model and CIG algorithm.

	MILP			CIG		
F_n_S	RPI	makespan	Time(s)	RPI	makespan	Time(s)
2_8_2	**0.00**	**201**	0.66	4.98	211	0.32
2_8_3	**0.00**	**263**	0.73	1.90	268	0.48
2_8_4	**0.00**	**304**	0.72	2.63	312	0.64
2_12_2	**0.00**	**316**	1000	**0.00**	**316**	0.48
2_12_3	**0.00**	**385**	29.04	0.31	386.2	0.72
2_12_4	**0.00**	**419**	126.16	0.72	422	0.96
2_16_2	**0.00**	**439**	1000	0.06	439.25	0.64
2_16_3	**0.63**	**478**	1000	1.23	480.85	0.96
2_16_4	**0.57**	**530**	1000	0.99	532.2	1.28
2_20_2	**0.00**	**528**	1000	**0.00**	**528**	0.8
2_20_3	1.20	592	1000	**0.24**	**586.4**	1.2
2_20_4	4.38	644	1000	**0.52**	**620.2**	1.6

Table 3
ARPI values in terms of different factories and jobs (minimum APRI values are in bold).

	d = 2	*d = 3*	*d = 4*	*d = 5*	*d = 6*	*d = 7*
f = 2	**0.25**	0.68	0.27	0.41	0.31	0.33
f = 3	**0.62**	3.46	0.67	1.14	0.69	0.8
f = 4	2.63	2.8	2.03	2.61	**0.91**	1.23
f = 5	3.28	6.22	1.34	4.25	**1.31**	1.64
Mean	1.7	3.29	1.08	2.1	**0.81**	1
n = 100	0.86	1.34	0.92	0.85	0.73	**0.66**
n = 200	1.24	3.27	1.07	1.33	**1.04**	1.9
n = 300	0.71	2.87	**0.25**	2.56	0.34	0.5
n = 400	4.88	2.99	1.48	3.65	1	**0.97**
n = 500	**0.59**	4.83	1.5	1.45	0.93	0.91
Mean	1.66	3.06	1.04	1.97	**0.81**	0.99

5.2. Verification of the MILP model

In this section, we validate the proposed MILP and test the performance of the MILP and CIG algorithm, respectively, on 12 small-scale instances. The MILP of DHHFSP with blocking constraints is coded in the CPLEX 12.6 software, and the maximal termination criterion is set 1000 s. If the optimal solution can be found within 1000 s, it will be terminated. For the CIG algorithm, the termination time is set to$TimeLimit = 10 \times n \times f \times S$. In addition, the CIG algorithm independently performed 20 times for each instance. Table 2 gives the RPI, makespan, and time obtained by the MILP and CIG algorithm. Among them, the makespan of CIG algorithm is the average value of 20 experimental results.

As can be seen from Table 2, MILP gets the best RPI and makespan values for the first seven instances and the first nine instances,

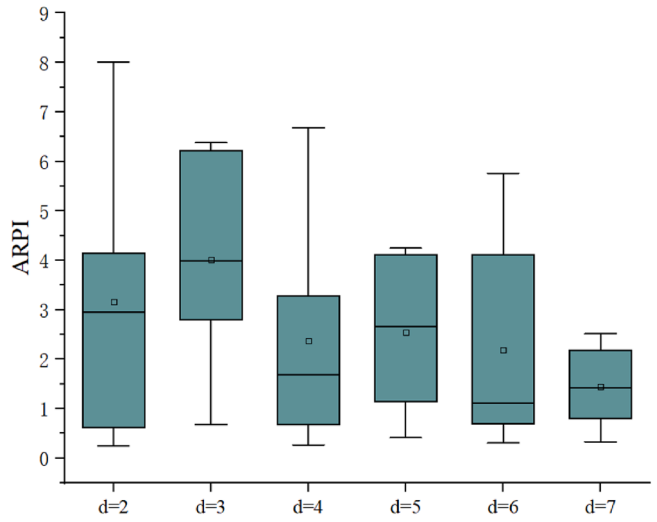


Fig. 3. Box plot for different d values.

Table 4
The ARPI values for comparison with different initialization strategies.

ARPI	CIG_des_asc	CIG_des_des	CIG_asc_asc
$f = 2$	0.244	0.410	**0.134**
$f = 3$	**0.299**	0.882	1.058
$f = 4$	**0.532**	1.270	0.854
$f = 5$	2.331	3.103	**1.492**
Mean	**0.851**	1.416	0.884

respectively. However, as the problem size increases, i.e., 2_20_3 and 2_20_4, the superiority of CIG algorithm over CPLEX increases with respect to RPI, makespan and run time. From Table 2, we can conclude that it is difficult to obtain a good solution in a short time for the large-scale instances of MILP. Thus, the proposed CIG algorithm is effective to solve the DHHFSP with blocking constraints.

5.3. The parameter test of the CIG algorithm

Although the CIG algorithm has few parameters, the parameter d (the number of destroyed jobs) is critical, which directly affects the algorithm's performance. Thus, in this subsection, we select a different number of destruction jobs, $d = \{2, 3, 4, 5, 6, 7\}$, to test the effect of parameter value on results. Further, we classify to test the influence from multiple angles, i.e., the scale of the factory $f = \{2, 3, 4, 5\}$and the scale of the job $n = \{100, 200, 300, 400, 500\}$, $S = \{5, 8, 10\}$, thus, there are $5 \times 4 \times 3 = 60$ differernt type combinations. For these 60 instances, we give the Average RPI (ARPI) value classified by the number of factories and the number of jobs to detect the effect of d parameter from different angles..

From Table 3, we can see that when the value of parameter d is equal to 6, the minimum mean value is obtained when considering different factories and different jobs. The results listed in Table 3 show that the different values of parameter d have a large sensitivity, and the variation trend of the influence on the proposed algorithm is consistent in terms of different factories and jobs. In fact, with the increase of d value, more and more jobs are destroyed and inserted into various positions to improve the quality of the job sequence. However, when the value of d is greater than 6, its mean value decreases. It shows that setting d at a large value will take a long time to explore all jobs, and lose opportunities to generate promising solutions by a number of iterations, which will result in reducing the performance of the algorithm. In order to more clearly see the differences of different d values in all scale instances, we draw a box diagram for all results. The results are shown in Fig. 3. As can be seen, when $d = 6$, the position of the median line in its rectangle is the lowest, and by analysing the values in Table 3. Through the experimental tests of the parameter d, we choose $d = 6$ as the parameter value used in the proposed algorithm.

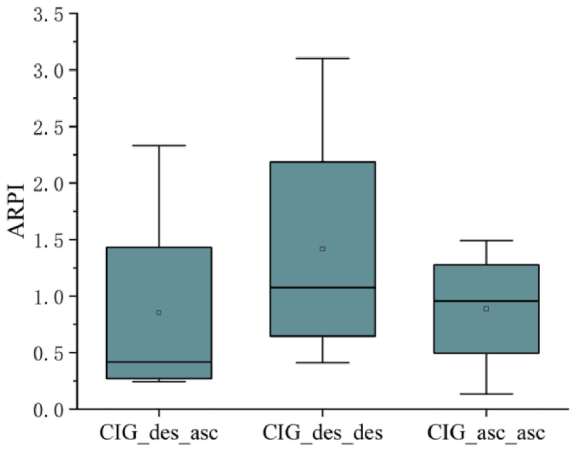


Fig. 4. Box plot for different initialization strategies.

Table 5
The ARPI values for comparison with and without neighborhood and intensification strategies.

ARPI	CIG	CIG_N_NSS	CIG_N_LIS
$f = 2$	**0.244**	1.581	3.221
$f = 3$	**0.299**	0.675	9.823
$f = 4$	**0.532**	2.534	12.574
$f = 5$	2.331	**0.533**	7.531
Mean	**0.851**	1.331	8.287

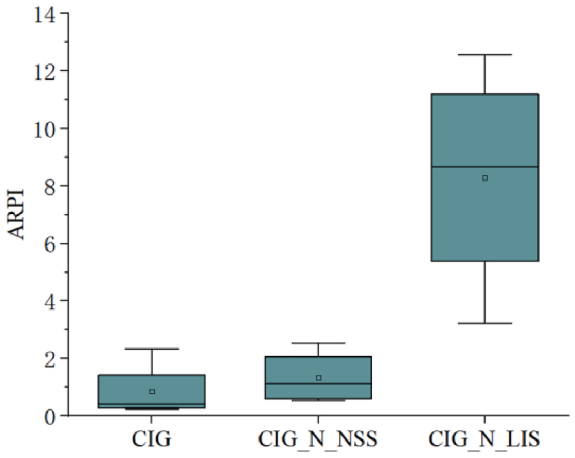
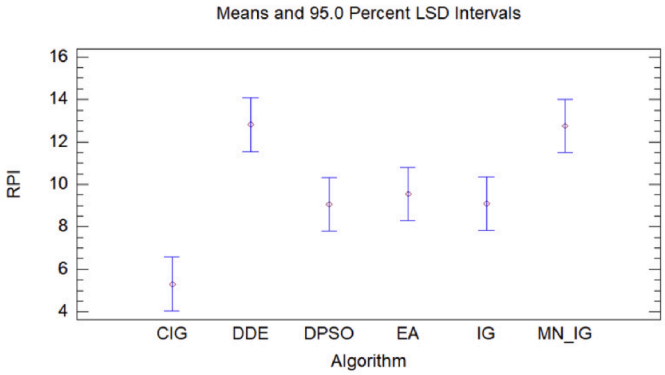


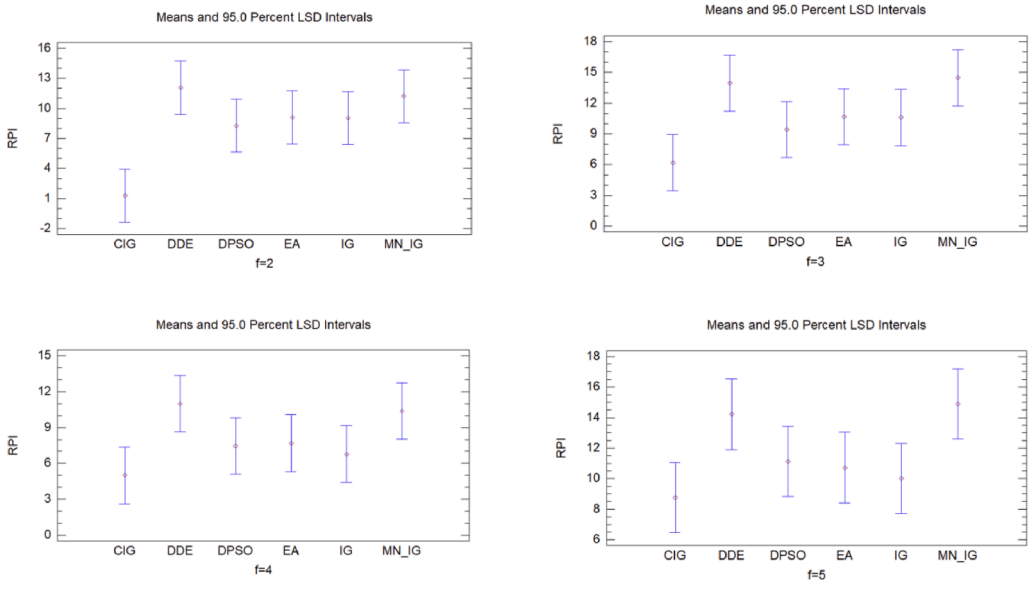
Fig. 5. Box plot with and without different strategies.

5.4. Comparison of different initialization strategies

To further investigate the performance of the proposed initialization strategies, we list the results obtained by the different initialization strategies in terms of the ARPI values. To further investigate the performance of the proposed initialization strategies, we list the results obtained by the different initialization strategies in terms of the ARPI values for 60 test instances. As can be seen from Table 4, CIG_des_asc indicates that the CIG used the *NEH_F_asc* and *NEH_F_des* strategies to generate two initial solutions, respectively. CIG_des_des utilizes the *NEH_F_des* and *NEH_F_des* strategies to generate two initial solutions.



(a) Means of algorithms



(b) Interactions of algorithms and factories

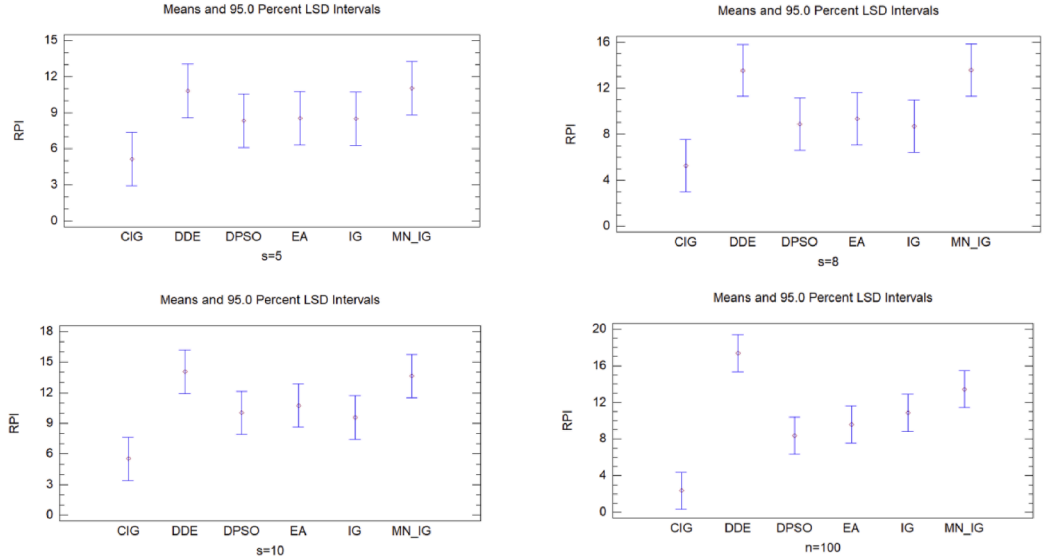
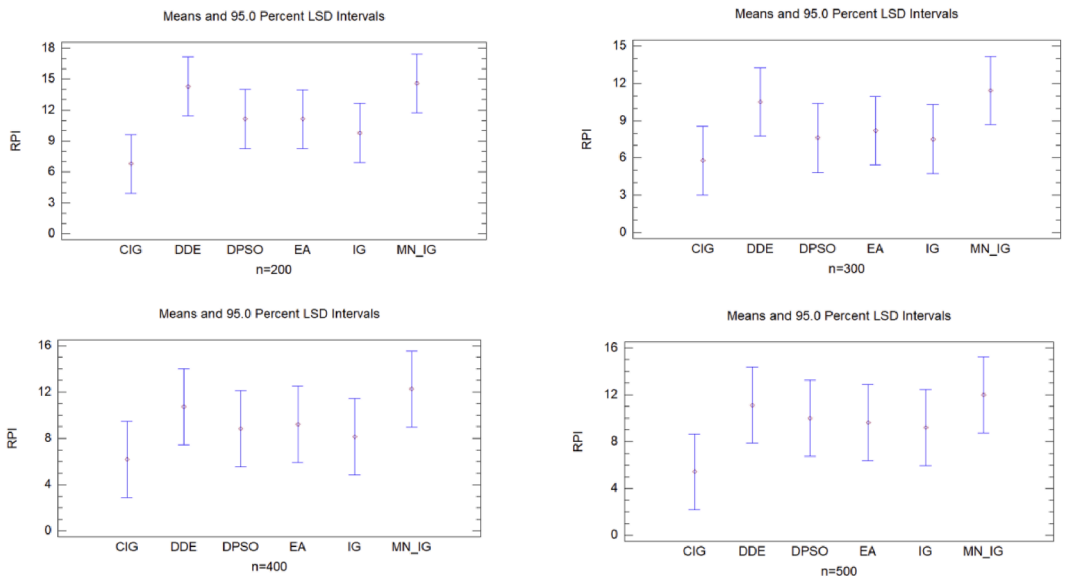


Fig. 6. Means plots and 95% LSD confidence intervals for all compared algorithms.



(c) Interactions of algorithms and stages and jobs

Fig. 6. (*continued*).

CIG_asc_asc employed the *NEH_F_asc* and *NEH_F_asc* strategies to generate the initial solutions. From the results of Table 4, the CIG_des_asc is superior to other algorithms. This shows that the performance of the proposed initialization strategy combined with *NEH_F_des* strategy is superior to other algorithms. The jobs with short and long processing time are selected to form a scheduling sequence, respectively, which can effectively reduce the impact of blocking constraints on completion time. Moreover, the proposed *NEH_F_asc* strategy not only improves the quality of the initial solution, but also improves the efficiency for the subsequent iteration. We also made a statistical test on the RPI values obtained from the 60 instances. As shown in Fig. 4, the box plot shows different levels of initialization strategies performance. Among them, the initialization strategy with *NEH_F_asc* strategy is effective. Considering the mean values obtained in Table 4 and Fig. 4, we finally choose the CIG_des_asc as the initialization strategy.

5.5. Performances of neighborhood search and local intensification strategies

The test of the CIG_N_NSS and CIG_N_LIS is based on the *NEH_F_asc* combined with the *NEH_F_des* strategies to execute. CIG_N_NSS is the CIG without the proposed neighborhood search strategy. CIG_N_LIS is the CIG without the local intensification strategy. From the experimental results, in each factory (as shown in Table 5 and Fig. 5), in terms of the ARPI value and range of box interval, the CIG with the above two strategies obtained the minimum mean value. In addition, Fig. 5 shows that the significance level of CIG algorithm is the best. The reason may be that the neighborhood search strategy improves the algorithm's global search ability and the local intensification strategy enhances the local exploitation ability. Without this strategy, the quality of the algorithm will decline sharply. It is possible that the proposed strategy makes a more in-depth exploration in sequence sorting, and many unknown neighborhoods are found, which improve the diversity of solutions. Thus, the blocking constraints of the job sequence are effectively reduced by the proposed strategies.

5.6. Performance comparison of all the compared algorithms

To comprehensively and intuitively evaluate the performance of the CIG algorithm, we compare the CIG against the state-of-the-art metaheuristics for related problems, the DDE algorithm (Zhang et al., 2018), the evolutionary algorithm (EA) (Fernandez-Viagas et al., 2018), the modeling and multi-neighborhood iterated greedy algorithm (MN_IG) (Shao et al., 2020), the IG algorithm (Ruiz et al., 2019), and the DPSO algorithm (Marichelvam et al., 2019). For the DHHFSP problem, there are two key issues to be solved. One is how to allocate jobs to appropriate factories and the other is how to generate the scheduling sequence of operations on machines with minimal makespan. DPSO has not been developed for HFSP without distributed environment. However, DPSO can deal with the second issue of DHHFSP. Thus, for the sake of fairness, DPSO adopts the same allocation strategy as the CIG algorithm to solve the first issue of DHHFSP. Specifically, DPSO first adopts the allocation strategy based on *NEH_ F_ des* and *NEH_ F_ asc* considered in this paper to initialize the population. Then, for each factory, the implementation strategy is employed according to the original literature. Thus, it is reasonable when DPSO can achieve the good performance. The reason may be that the evolutionary strategies are effective when optimizing the solutions of each factory.

Besides the DPSO algorithm, all the compared algorithms, i.e., DDE, EA, IG, and MN_IG, are developed for the distributed scheduling problems. For MN_IG, it used the *NEH_F_asc* initialization strategy to allocate the jobs to different factories. After allocating the factories, the sequence in each factory is adjusted according to the strategy proposed in the original literature. For EA, DDE, IG, they allocate jobs to factories according to the strategy of the original paper. In the code, we separate the problem from the algorithm. When the factory problem is finished, the subsequent operations of each algorithm and the problem will not interfere with each other. Thus, it is reasonable to select the above compared algorithms.

To better show the performance of all the algorithms, we set the parameters used in the compared algorithms to the same value as in the original literature. To make fair comparisons, we run all the compared algorithms under the same environment and adopt the same maximal elapsed CPU time, *Timelimit*, as the termination criterion. In addition, the Means and 95% Least Significant Difference (LSD) confidence intervals (verify whether there is a significant difference between the two values) for all test algorithms are illustrated in Fig. 6.

Tables 6 and 7 show the RPI values and best target values of all algorithms, in which the minimum values of the results are marked in bold. We can find that the number of the minimum best values produced

Table 6
Experimental results of the best and RPI values for all test algorithms when $\omega = 5$.

f	$n \times s$	CIG		DDE(2018)		EA(2018)		IG(2019)		DPSO(2019)		MN_IG(2020)	
		best	RPI	best	RPI	best	RPI	best	RPI	best	RPI	best	RPI
	100 × 5	**2759**	**0.32**	2928	6.13	2791	2.02	2814	3.42	2772	1.82	2932	6.27
	100 × 8	**2682**	**1.15**	3348	24.83	3050	15.72	3035	15.53	3030	15.27	3383	26.14
	100 × 10	**2858**	**1.6**	3329	16.48	3101	9.41	3104	10.52	3081	9.83	2996	4.83
	200 × 5	**5113**	**0.22**	5176	1.23	5138	0.9	5119	0.33	5128	0.73	5235	2.39
	200 × 8	**5778**	**0.26**	6166	6.8	5870	2.51	6079	5.66	5806	1.16	5855	1.33
	200 × 10	**4970**	**0.34**	5280	8.59	5107	3.88	5261	6.81	5024	2.12	5037	1.35
	300 × 5	**4402**	**2.52**	5199	18.11	5078	15.88	4999	14.56	5062	16.76	5341	21.33
$f = 2$	300 × 8	**7538**	**0.35**	7764	4.7	7603	1.7	7840	4.03	7562	0.62	7694	2.07
	300 × 10	**7732**	**1.02**	8571	10.85	8158	6.52	8119	5.92	8066	5.26	8513	10.1
	400 × 5	**10,152**	0.32	10,329	2.21	10,203	0.94	10,318	1.91	10,168	0.34	**10,152**	**0**
	400 × 8	**9871**	**0.83**	10,351	4.86	10,296	4.99	10,153	3.37	10,121	3.21	10,739	8.79
	400 × 10	**10,788**	**1.85**	13,564	25.73	13,164	23.78	12,842	20.14	12,561	17.98	13,211	22.46
	500 × 5	**6335**	**1.17**	7035	11.05	6963	10.29	6866	8.96	7018	11.02	7234	14.19
	500 × 8	**6832**	**3.26**	8112	18.74	8040	18.07	7930	17.41	8087	18.7	8441	23.55
	500 × 10	**7539**	**4**	9098	20.68	8988	19.73	8731	16.61	8869	19.19	9308	23.46
	mean	**6356.6**	**1.28**	7083.33	12.07	6903.33	9.09	6880.67	9.01	6823.67	8.27	7071.4	11.22
	100 × 5	**1801**	**2.79**	2039	13.21	1900	7.77	2065	17.04	1874	6.22	1974	9.61
	100 × 8	**1828**	**1.79**	2210	20.9	1938	7.54	1976	10.67	1882	4.57	2080	13.79
	100 × 10	**2394**	**3.3**	2924	22.14	2745	16.45	2739	17.21	2728	16.57	2945	23.02
	200 × 5	**1917**	**4.1**	2273	18.57	2158	14.31	2109	11.86	2151	13.57	2243	17.01
	200 × 8	**3701**	**5.53**	4521	22.16	4307	18.15	4207	15.44	4216	16.05	4712	27.32
	200 × 10	**4276**	**4.28**	5256	22.92	5091	20.26	5016	18.59	4949	17.27	5231	22.33
	300 × 5	**5766**	**17.6**	7118	23.45	6984	21.69	6731	18.29	6865	20.34	7224	25.29
$f = 3$	300 × 8	**5460**	2.96	5782	5.9	5555	2.57	5608	3.17	5519	**2**	5693	4.27
	300 × 10	**5754**	15.63	6742	17.17	6724	17.63	6536	15.35	6533	**15.17**	7140	24.09
	400 × 5	**7353**	17.4	8721	18.6	8745	19.33	8391	**15.74**	8658	18.43	9174	24.77
	400 × 8	**6899**	3.28	7046	2.13	6999	1.81	6952	**1.27**	6981	1.79	7205	4.44
	400 × 10	7182	2.93	7532	6.38	**7119**	1.19	7357	4.48	7125	**0.67**	7382	3.69
	500 × 5	**8689**	2.43	9010	3.69	8848	2.48	8777	1.92	8749	**1.46**	9040	4.04
	500 × 8	**8500**	3.23	9096	7.01	8836	4.63	8719	3.71	8763	3.82	9085	6.88
	500 × 10	**8988**	5.77	9438	5.01	9345	4.28	9308	4.4	9223	**3.4**	9580	6.59
	mean	**5367.2**	**6.2**	5980.53	13.95	5819.6	10.67	5766.07	10.61	5747.73	9.42	6047.2	14.47
	100 × 5	**1290**	**2.48**	1444	11.94	1336	6.99	1380	8.06	1322	4.37	1380	6.98
	100 × 8	**1550**	**2**	1849	19.29	1649	7.88	1618	7.04	1603	5.26	1709	10.26
	100 × 10	**1714**	**4.8**	2021	17.91	1891	11.86	1824	8.81	1900	13.45	2077	21.18
	200 × 5	**1419**	**9.51**	1668	17.55	1620	15.27	1591	13.5	1650	18.33	1669	17.62
	200 × 8	**1484**	**8.28**	1835	23.65	1660	13.73	1631	11.73	1659	13.35	1729	16.51
	200 × 10	**2800**	4.08	2968	6	2884	4.22	2847	**2.62**	2858	3.03	3013	7.61
	300 × 5	4104	3.58	4075	0.3	4074	**0.29**	**4063**	0.35	4107	1.08	4230	4.11
$f = 4$	300 × 8	**3980**	5.36	4204	5.63	4136	4.82	4084	**3.26**	4097	4.92	4379	10.03
	300 × 10	**4324**	3.61	4521	4.8	4332	1.85	4314	**0.61**	4330	1.33	4465	3.5
	400 × 5	**5095**	2.17	5189	1.84	5165	1.69	5133	**1.09**	5161	1.59	5191	1.88
	400 × 8	**5774**	**12.23**	6990	21.06	6782	18.7	6656	17	6765	18.95	7255	25.65
	400 × 10	**5220**	**5.07**	5657	8.37	5541	7.04	5468	5.68	5498	6.12	5778	10.69
	500 × 5	**6342**	1.92	6386	0.69	6372	**0.56**	6382	0.83	6386	0.99	6479	2.16
	500 × 8	**2816**	**9.38**	3365	19.5	3307	18.01	3243	15.83	3296	17.87	3283	16.58
	500 × 10	**6508**	**0.67**	6915	6.25	6634	2.38	6842	5.21	6557	1.29	6561	0.81
	mean	**3628**	**5.01**	3939.13	10.99	3825.53	7.69	3805.07	6.77	3812.6	7.46	3946.53	10.37
	100 × 5	**1173**	**2.63**	1359	15.86	1228	7.74	1232	9.68	1208	4.61	1282	9.29
	100 × 8	**1244**	**1.7**	1521	22.27	1308	8.43	1399	14.85	1286	4.97	1338	7.56
	100 × 10	**1400**	**3.93**	1646	17.57	1565	13.11	1458	7.69	1562	13.6	1714	22.43
	200 × 5	**2418**	15.16	2784	15.14	2715	**13.25**	2705	14.76	2794	15.55	2984	23.41
	200 × 8	**1406**	12.94	1525	8.46	1516	8.39	1448	**5.34**	1525	8.46	1655	17.71
	200 × 10	**1343**	16.76	1619	20.55	1566	18.59	1472	**10.77**	1539	16.76	1617	20.4
	300 × 5	**1668**	**8.35**	1911	14.57	1857	12.84	1823	10.39	1847	12.07	1908	14.39
$f = 5$	300 × 8	**3355**	**5.39**	3671	9.42	3552	6.61	3547	7.4	3569	7.28	3746	11.65
	300 × 10	**3185**	**3.1**	3538	11.08	3337	5.91	3378	6.88	3288	4.53	3384	6.25
	400 × 5	**2049**	**7.73**	2403	17.28	2338	15	2305	13.28	2360	16.15	2345	14.45
	400 × 8	**4506**	4.77	4555	1.09	4541	1.02	4512	**0.91**	4622	2.57	4937	9.57
	400 × 10	**5078**	15.76	6047	19.08	5808	15.09	5676	**12.79**	5945	18.26	6129	20.7
	500 × 5	**5023**	**0.58**	5258	4.68	5072	1.52	5212	4.06	5042	0.86	5078	1.09
	500 × 8	**5767**	**20.53**	7068	22.56	6969	21.72	6932	21	7193	26.68	7345	27.36
	500 × 10	**5927**	12.15	6737	13.67	6530	11.66	6443	**10.41**	6738	14.66	6939	17.07
	mean	**3036.13**	**8.77**	3442.8	14.22	3326.8	10.72	3302.8	10.01	3367.87	11.13	3493.4	14.89

by the CIG algorithm are 58 out of 60 test instances in Table 6 and 56 out of 60 test instances in Table 7. The DDE, EA, MN_IG, IG, and DPSO algorithms can get 0, 1, 1, 1, 0 best values, respectively, and the number of all these algorithms is significantly less than that of the CIG algorithm. For RPI values, the CIG algorithm obtains the maximum number of the best results, followed by IG, EA, DPSO, MN_IG and DDE algorithms. About the mean value of instances in each factory, the proposed algorithm can get 6/6 minimum best values and 6/6 minimum best RPI values in Tables 6 and 7. EA, DDE, MN_IG, and IG algorithms obtain 0/6 minimum best values and 0/6 minimum best RPI values, respectively. From the overall performance shown in Tables 6 and 7, the proposed CIG algorithm is better than other comparison algorithms. This indicates that the proposed strategies are effective in solving DHHFSP with blocking constraints.

Table 7
Experimental results of the best and RPI values for all test algorithms when$\omega = 10$.

f	$n \times s$	CIG		DDE[2018]		EA[2018]		IG[2019]		DPSO[2019]		MN_IG[2020]	
		best	RPI	best	RPI	best	RPI	best	RPI	best	RPI	best	RPI
	100 × 5	**2762**	**0.11**	2928	6.01	2789	1.44	2806	3.15	2772	1.37	2932	6.15
	100 × 8	**2644**	**0.42**	2838	7.34	2678	2.66	2690	3.82	2654	1.44	2752	4.08
	100 × 10	**2684**	**0.59**	3062	16.83	2768	4.73	2841	8.98	2729	3.22	2819	5.03
	200 × 5	**1765**	**2.47**	2033	15.18	1952	11.88	1968	12.86	1995	14.71	2163	22.55
	200 × 8	**2984**	**2.02**	3553	19.07	3346	14	3352	13.84	3402	15.63	3665	22.82
	200 × 10	**5301**	**1.99**	6821	28.67	6207	18.67	6250	20.62	6149	17.56	6476	22.17
	300 × 5	**7617**	**0.29**	8148	6.97	7853	3.66	7851	4	7786	2.58	8076	6.03
***f* = 2**	300 × 8	**7607**	**0.7**	8254	8.51	7911	4.76	7921	4.78	7767	2.94	8178	7.51
	300 × 10	**8124**	**1.06**	8577	5.88	8289	2.79	8352	3.27	8184	1.43	8342	2.68
	400 × 5	**10,277**	**0.91**	10,825	5.33	10,698	4.54	10,634	4.18	10,499	3.11	10,986	6.9
	400 × 8	**9959**	0.72	10,125	1.67	10,011	1.01	10,004	1.06	9977	**0.69**	10,139	1.81
	400 × 10	**9975**	1.16	10,178	2.04	10,106	1.61	10,060	1.27	10,027	**0.86**	10,176	2.02
	500 × 5	**12,498**	**0.48**	13,156	5.26	12,872	3.51	12,813	3.41	12,663	1.8	13,045	4.38
	500 × 8	**12,995**	**2.48**	16,323	25.61	15,906	22.91	15,465	20.28	15,296	18.8	16,846	29.63
	500 × 10	**13,391**	**2.26**	16,515	23.33	16,350	22.91	15,870	19.5	15,588	18.06	17,011	27.03
	mean	**7372.2**	**1.18**	8222.4	11.85	7982.4	8.07	7925.13	8.33	7832.53	6.95	8240.4	11.39
	100 × 5	**1746**	**1.38**	1955	11.97	1792	3.66	1855	6.84	1760	1.96	1816	4.01
	100 × 8	**1834**	**2.07**	2125	15.87	1933	7.6	1938	7.57	1956	8.26	2106	14.83
	100 × 10	**1977**	**1.75**	2491	26	2201	13.36	2221	16.08	2212	14.33	2293	15.98
	200 × 5	**3043**	**1.23**	3247	6.7	3118	3.22	3161	4.34	3066	1.99	3112	2.27
	200 × 8	**3888**	**3.41**	4902	26.08	4607	20.36	4543	18.18	4601	20.32	4931	26.83
	200 × 10	**3880**	**3.71**	4976	28.25	4434	15.9	4368	13.85	4399	15.09	4925	26.93
	300 × 5	**5419**	1.72	5505	1.59	5460	1.27	5433	**1.17**	5484	1.56	5724	5.63
***f* = 3**	300 × 8	**5660**	3.35	5967	5.42	5781	2.53	5741	**2.18**	5782	3.64	6164	8.9
	300 × 10	**5348**	**8.02**	6219	16.29	6014	13.35	5878	11.47	6013	13.69	6303	17.86
	400 × 5	**6737**	2.29	6862	1.86	6786	1.14	6796	1.25	6761	**0.81**	6844	1.59
	400 × 8	**7651**	**14.6**	9523	24.47	9319	22.61	9027	18.8	9126	21.6	9603	25.51
	400 × 10	**4161**	**14.4**	4915	18.12	4901	18.42	4841	17	4915	18.12	5287	27.06
	500 × 5	8585	1.31	8590	0.25	8588	**0.24**	**8569**	0.25	8610	0.82	8652	0.97
	500 × 8	**8239**	1.78	8436	2.39	8338	1.85	8347	1.57	8310	**1.28**	8459	2.67
	500 × 10	**8421**	2.85	8692	3.22	8566	2.32	8555	2.13	8510	**1.81**	8745	3.85
	mean	**5105.93**	**4.26**	5627	12.56	5455.87	8.52	5418.2	8.18	5433.67	8.35	5664.27	12.33
	100 × 5	**1352**	**3.49**	1621	19.9	1431	8.26	1428	8.14	1441	8.89	1563	15.61
	100 × 8	**1585**	**2.51**	1848	16.59	1656	6.96	1610	4.34	1675	7.88	1738	9.65
	100 × 10	**1834**	**2.29**	2214	20.72	2044	13.3	1978	11.27	2032	13.28	2221	21.1
	200 × 5	**2521**	1.82	2604	3.29	2562	2.39	2559	2.31	2531	**1.45**	2590	2.74
	200 × 8	**2884**	**9.88**	3485	20.84	3305	16.61	3219	13.98	3331	16.83	3655	26.73
	200 × 10	**1684**	**6.36**	2099	24.64	1930	15.91	1857	12.71	1939	16.46	2049	21.67
	300 × 5	**1985**	**6.12**	2202	10.93	2134	8.47	2058	4.88	2143	9.31	2210	11.34
***f* = 4**	300 × 8	**2109**	**7.76**	2475	17.35	2334	12.77	2342	12.08	2324	12.28	2420	14.75
	300 × 10	**4189**	**12.43**	5190	23.9	4828	16.65	4721	14.39	4794	16.15	5277	25.97
	400 × 5	**2843**	**14.18**	3425	20.47	3344	18.56	3199	13.55	3363	19.57	3352	17.9
	400 × 8	**5280**	5.45	5612	6.29	5475	4.63	5451	**3.93**	5502	5.56	5610	6.25
	400 × 10	**5106**	**1.11**	5507	7.85	5220	2.98	5353	5.6	5144	1.55	5251	2.84
	500 × 5	**6407**	1.43	6660	3.95	6469	1.39	6653	4.01	6428	**0.8**	6536	2.01
	500 × 8	**6758**	5.22	7002	3.61	6790	1.56	**6758**	**0.88**	6816	1.91	7061	4.48
	500 × 10	**7160**	17.11	8385	17.11	8098	13.92	8165	**15.57**	8171	15.83	8651	20.82
	mean	**3579.8**	**6.48**	4021.93	14.5	3841.33	9.63	3823.4	8.51	3842.27	9.85	4012.27	13.59
	100 × 5	**1013**	**3.95**	1135	12.04	1059	7.49	1037	4.78	1045	4.92	1156	14.12
	100 × 8	**1395**	**4.7**	1734	24.3	1524	11.6	1553	17.6	1524	11.57	1676	20.14
	100 × 10	**1230**	**4.59**	1486	20.81	1348	12.15	1315	9.46	1324	10.53	1439	16.99
	200 × 5	**1177**	**7.37**	1339	13.76	1294	11.02	1250	7.47	1299	11.95	1417	20.39
	200 × 8	**2409**	**6.47**	2809	16.6	2581	8.66	2548	9.4	2583	10.03	2679	11.21
	200 × 10	**2339**	**3.85**	2666	13.98	2437	6.02	2472	7.21	2401	4	2555	9.23
	300 × 5	3492	5	3643	6.93	3423	1.58	**3407**	**1.35**	3456	2.77	3614	6.08
***f* = 5**	300 × 8	3172	3.15	3518	10.98	**3170**	**2.09**	3386	9.85	3240	3.34	3264	2.97
	300 × 10	**3508**	6.15	3838	9.41	3642	5.08	3565	**2.61**	3664	6.28	3751	6.93
	400 × 5	**4976**	17.49	5655	13.65	5611	**13.02**	5579	13.3	5780	16.16	6122	23.03
	400 × 8	4275	1.85	4571	7.02	**4271**	**1.15**	4494	7.15	4331	2.4	4387	2.72
	400 × 10	**4646**	19.55	5528	18.98	5382	16.76	5208	**14.46**	5376	17.71	5737	23.48
	500 × 5	**5332**	5.82	5484	2.85	5442	**2.28**	5444	3.22	5655	6.65	5786	8.51
	500 × 8	**2767**	**7.4**	3115	12.58	2986	9.4	2951	7.42	3004	9.39	3103	12.14
	500 × 10	**5275**	6.54	5541	5.04	5473	4.32	5399	**3.21**	5489	5.07	5762	9.23
	mean	**3133.73**	**6.93**	3470.8	12.6	3309.53	7.51	3307.2	7.9	3344.73	8.18	3496.53	12.48

To further evaluate the performance of the proposed algorithm, a multifactor ANOVA analysis is utilized to illustrate whether the results obtained by CIG are indeed different from the compared algorithms. The type of the algorithms and jobs, factories, and stages are considered as the factors. Fig. 6 shows that the Means plots and 95% least-significant difference (LSD) confidence intervals for all compared algorithms when$\omega = 5$. Fig. 6(a) illustrates the means plots of the algorithms. Fig. 6 (b) shows the interactions of the algorithms and the number of factories (f). Fig. 6(c) gives the interactions of the algorithms and the number of the stages and jobs (s and n). In addition, to further illustrate the difference among these algorithms, we randomly select six examples of different scales to draw the box plots. The details are shown in Fig. 7 (a-f).

To enrich the statistical results, we carried out the Friedman test on

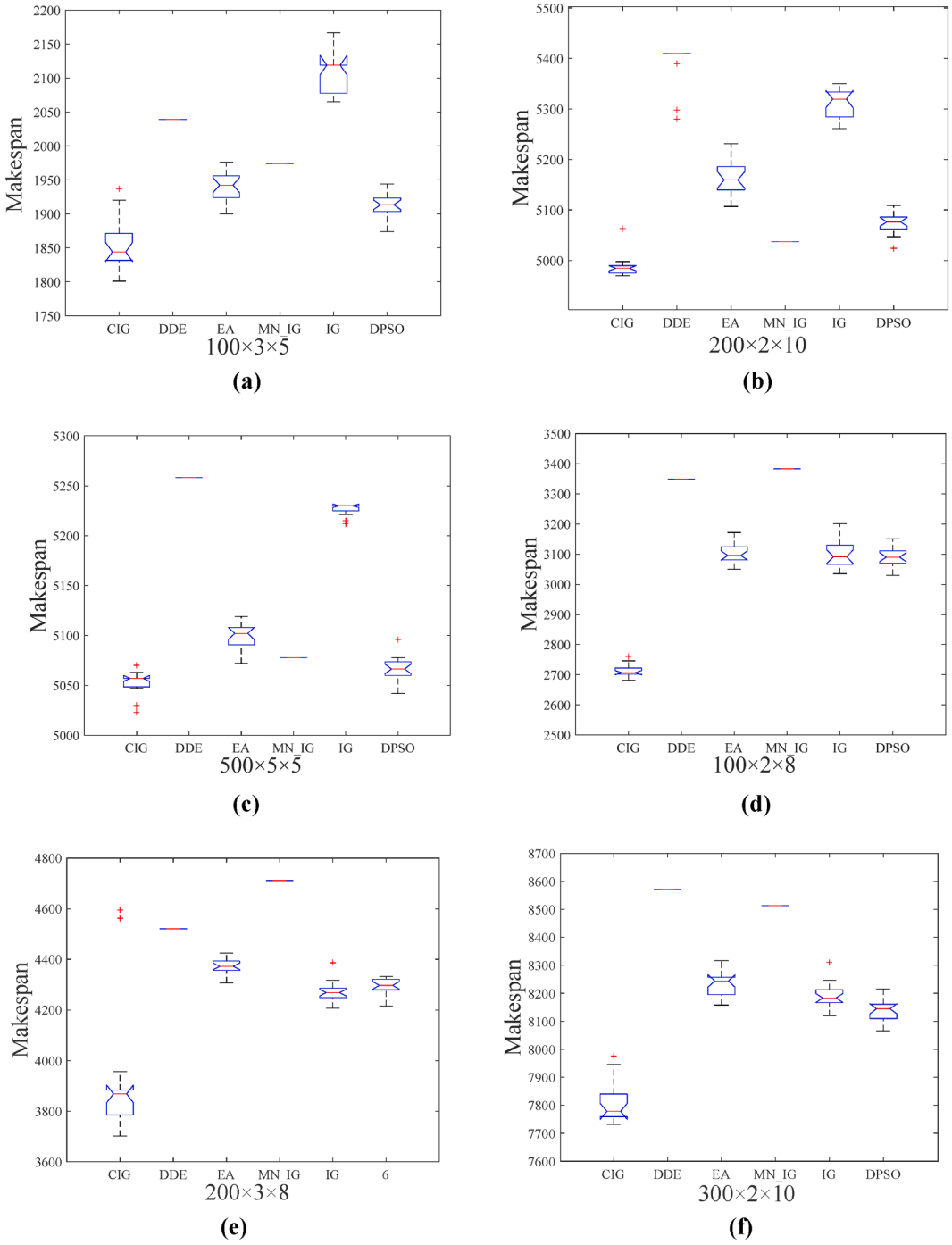


Fig. 7. The box plots of all compared algorithms.

Table 8
Experimental results of the Friedman test (α = 0.05).

Algorithms	Ranks	N	Mean	Std. Deviation	Min	Max
CIG	**1.05**	150	**1.91**	**1.84**	**0.00**	**14.66**
DDE	5.47	150	16.70	5.95	6.13	24.83
EA	3.13	150	9.31	4.70	1.16	18.38
MN_IG	4.64	150	12.77	7.66	4.83	26.14
IG	4.13	150	11.47	4.94	1.99	20.32
DPSO	2.58	150	8.17	5.27	0.47	18.42
p-value	0.00					

the experimental results. The Friedman test is a kind of non-parametric test, and the first step is to assume that all algorithms are null hypotheses. Judge whether the result rejects the hypothesis. If it rejects, it indicates that the two algorithms have statistically significant differences; otherwise, there is no significant difference between any algorithms. The Friedman test is illustrated in Table 8. It contains the following elements: the performance ranks (Ranks), the test number (N), the mean value (Mean), the standard deviation (Std. Deviation), minimum and maximum value (Min and Max).

From Fig. 6(a), it is clear that the proposed CIG algorithm is significantly different from all of the compared algorithms. The CIG is the

best; IG, DPSO, EA, DDE, and MN_IG follow sequentially. They all show better performance in solving the DHHFSP with blocking constraints. In Fig. 6(b), with the increase of the factories, the performance gap between the proposed algorithm and other algorithms is gradually narrowing. However, there are still some regions that are significantly different from the comparison algorithms. In terms of the stages and jobs, as shown in Fig. 6(c), the proposed algorithm is significantly different from other comparison algorithms. In the Friedman test, it can be observed that it has the best performance for comparison with the minimum ranks of 1.05. Moreover, compared to other algorithms, the Std. Deviation, Mean, Min, and Max value of the CIG are all minimum. The *p-value* calculated from the experimental results is 0.00. It shows the significant differences among these algorithms. In addition, it also can be seen from Fig. 7 (a-f) that makespan in the box plot of the CIG algorithm is smaller than those by the DDE, EA, MN_IG, IG, and DPSO. In 6 randomly selected examples, the proposed algorithm shows significant differences from other algorithms. On the whole view, the CIG algorithm performs best.

The reason for the CIG algorithm's superior performance can be explained as follows: Firstly, the proposed *NEH_F_asc* strategy can improve the product productivity by prioritizing the production of jobs with short processing time. Blocking conditions for many small jobs are effectively reduced. Secondly, the neighborhood search strategy can quickly adjust the job sequence to search for a better solution and increase the global search ability. Thirdly, the local intensification strategy enhances the local search capability of the algorithm. The solution of a single factory can be explored more deeply by this strategy. Furthermore, the cross-factory and inner-factory search strategies can maintain a balance between exploration and exploitation. After completing one iteration, the quality of the solution will be further improved. To sum up, according to the experimental results of all algorithms, the proposed CIG algorithm is the most suitable to solve the DHHFSP with blocking constraints for minimizing the makespan.

6. Conclusions

This paper presents an effective CIG algorithm to solve the DHHFSP with blocking constraints for minimizing the makespan. To the best of our knowledge, this is the first work that solves such a meaningful problem. For solving this problem, a mathematical model of the DHHFSP with blocking constraints is developed. Next, we design the *NEH_F_asc* initialization strategy combined with the *NEH_F_des* to reduce the impact of blocking as much as possible in the early stage of the algorithm. Later, a neighborhood search strategy with two search operators is presented to improve the global search ability of the CIG algorithm and quickly adjust the job sequence in different factories. Moreover, the local intensification strategy based on the swap operation is utilized to explore the neighborhood of the solution further and improve the overall quality of the solution. However, the CIG algorithm proposed in this paper does not consider the adjustment of the number of jobs in the factory after the factory is allocated to the jobs. This is a limitation of this algorithm. In later research, we will consider this factor and continue to improve CIG algorithm.

In the future, many flow shop scheduling problems with heterogeneous factories and blocking constraints can be studied. Other constraints, such as the machine breakdowns, the setup time, the job assembly, etc., can be considered. In addition, the multiobjective DHHFSP is an interesting research topic. Furthermore, we would like to improve the performance of the CIG algorithm. It may be a good choice to introduce reinforcement learning or deep learning into the algorithm. According to the real-world production needs, we may consider applying the CIG algorithm to the real production shop scheduling problem.

CRediT authorship contribution statement

Hao-Xiang Qin: Writing-origianl draft. **Yu-Yan Han:** Writing-review & editing. **Yi-Ping Liu:** Supervision. **Jun-Qing Li:** Supervision. **Quan-Ke Pan:** Supervision. **Xue-Han:** Supervision.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was jointly supported by the National Natural Science Foundation of China under grant numbers 61803192, 62173216, 62106073, 61973203, 61966012, We are grateful for Youth Innovation Talent Introduction and Education Program support received from Shandong province colleges, universities and Natural Science Foundation of Hunan Province of China under grant number 2021JJ40116, and the Fundamental Research Funds for the Central Universities(Grant NO. HNU: 531118010537).

Reference

Bargaoui, H., Belkahla, D. O., & Ghédira, K. (2017). A novel chemical reaction optimization for the distributed permutation flowshop scheduling problem with makespan criterion. *Computers & Industrial Engineering, 111*, 239–250.

Cai, J.-C., Zhou, R., & Lei, D.-M. (2020). Dynamic shuffled frog-leaping algorithm for distributed hybrid flow shop scheduling with multiprocessor tasks. *Engineering Applications of Artificial Intelligence, 90*, 1–13.

Choi, H. S., Kim, J. S., & Lee, D. H. (2011). Real-time scheduling for reentrant hybrid flow shops: A decision tree based mechanism and its application to a TFT-LCD line. *Expert Systems with Applications, 38*, 3514–3521.

Deng, J., & Wang, L. (2016). A competitive memetic algorithm for multiobjective distributed permutation flow shop scheduling problem. *Swarm & Evolutionary Computation, 32*, 107–112.

Feng, X., Zheng, F., & Xu, Y. (2016). Robust scheduling of a two-stage hybrid flow shop with uncertain interval processing times. *International Journal of Production Research, 54*, 3706–3717.

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

Fernandez-Viagas, F., & J. m.. (2020). Design of a testbed for hybrid flow shop scheduling with identical machines Victor. *Computers & Industrial Engineering, 141*, 1–11.

Fu, Y.-P., Tian, G.-D., et al. (2019). Stochastic multiobjective modelling and optimization of an energy-conscious distributed permutation flow shop scheduling problem with the total tardiness constraint. *Journal of Cleaner Production, 226*, 515–525.

Han, W., Deng, Q., Gong, G., et al. (2020). Multiobjective evolutionary algorithms with heuristic decoding for hybrid flow shop scheduling problem with worker constraint. *Expert Systems with Applications, 168*, 1–17.

Han, Y.-Y., Gong, D.-W., Jin, Y.-C., et al. (2019). Evolutionary Multiobjective Blocking Lot-Streaming Flow Shop Scheduling with Machine Breakdowns. *IEEE Transactions on Cybernetics, 49*, 184–197.

Han, Y.-Y., Li, J.-Q., Sang, H.-Y., et al. (2020). Discrete evolutionary multiobjective optimization for energy-efficient blocking flow shop scheduling with setup time. *Applied Soft Computing, 93*, 1–15.

Huang, J.-P., Pan, Q.-K., & Gao, L. (2020). An effective iterated greedy method for the distributed permutation flowshop scheduling problem with sequence-dependent setup times. *Swarm and Evolutionary Computation, 59*, 1–18.

Li, J.-Q., Sang, H.-Y., Han, Y.-Y., et al. (2018). Efficient multiobjective optimization algorithm for hybrid flow shop scheduling problems with setup energy consumptions. *Journal of Cleaner Production, 181*, 584–598.

Li, J.-Q., Song, M.-X., Wang, L., et al. (2020). Hybrid Artificial Bee Colony Algorithm for a Parallel Batching Distributed Flow-Shop Problem with Deteriorating Jobs. *IEEE Transactions on Cybernetics, 50*, 2425–2439.

Li, Y.-L., Xin, Y.-L., Gao, L., & Meng, L.-L. (2020). An Improved Artificial Bee Colony Algorithm for Distributed Heterogeneous Hybrid Flowshop Scheduling Problem with Sequence-Dependent Setup Times. *Computers & Industrial Engineering, 147*, 1–12.

Lin, J., Wang, Z.-J., & Li, X. (2017). A backtracking search hyper-heuristic for the distributed assembly flow-shop scheduling problem. *Swarm & Evolutionary Computation, 36*, 124–135.

Liu, S.-W., Pei, J., Cheng, H., et al. (2019). Two-stage hybrid flow shop scheduling on parallel batching machines considering a job-dependent deteriorating effect and non-identical job sizes. *Applied Soft Computing, 84*, 1–15.

Long, J., Zheng, Z., Gao, X., & Pardalos, P. M. (2018). Scheduling a realistic hybrid flow shop with stage skipping and adjustable processing time in steel plants. *Applied Soft Computing, 64*, 536–549.

Marichelvam, M. K., Geetha, M., et al. (2019). An improved particle swarm optimization algorithm to solve hybrid flowshop scheduling problems with the effect of human factors – a case study. *Computers & Operations Research, 114*, 1–9.

Pan, Q.-K., Gao, L., & Wang, L. (2020). An Effective Cooperative Co-Evolutionary Algorithm for Distributed Flowshop Group Scheduling Problems. *IEEE Transactions on Cybernetics, 99*, 1–14.

Pan, Q.-K., Wang, L., Li, J.-Q., & Duan, J.-H. (2014). A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimization. *Omega, 45*, 42–56.

Peng, K., Pan, Q.-K., Gao, L., Zhang, B., & Pang, X. (2018). An improved artificial bee colony algorithm for real-world hybrid flowshop rescheduling in steelmaking-refining-continuous casting process. *Computers and Industrial Engineering, 122*, 235–250.

Qin, H.-X., Han, Y.-Y., Chen, Q.-D., et al. (2021a). A Double Level Mutation Iterated Greedy Algorithm for Blocking Hybrid Flow Shop Scheduling. *Control and Decision*, 1–10. https://doi.org/10.13195/j.kzyjc.2021.0607.

Qin, H.-X., Han, Y.-Y., Zhang, B., Meng, L.-L., et al. (2021b). An improved iterated greedy algorithm for the energy-efficient blocking hybrid flow shop scheduling problem. *Swarm and Evolutionary Computation*. https://doi.org/10.1016/j.swevo.2021.100992

Qin, M., Wang, R., Shi, Z., et al. (2019). A Genetic Programming-Based Scheduling Approach for Hybrid Flow Shop With a Batch Processor and Waiting Time Constraint. *IEEE Transactions on Automation Science and Engineering, 99*, 1–12.

Ribas, I., Companys, R., & Tort-Martorell, X. (2011). An iterated greedy algorithm for the flowshop scheduling problem with blocking. *Omega, 39*, 293–301.

Ribas, I., Companys, R., & Tort-Martorell, X. (2015). An efficient Discrete Artificial Bee Colony algorithm for the blocking flow shop problem with total flowtime minimization. *Expert Systems with Applications, 42*, 6155–6167.

Ribas, I., Companys, R., & Tort-Martorell, X. (2019). An Iterated Greedy Algorithm for Solving the Total Tardiness Parallel Blocking Flow Shop Scheduling Problem. *Expert Systems with Applications, 121*, 347–361.

Ruben, R., & Thomas, S. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research, 177*, 2033–2049.

Ruiz, R., Pan, Q.-K., & Naderi, B. (2019). Iterated Greedy methods for the distributed permutation flowshop scheduling problem. *Omega, 83*, 213–222.

Salvador, M. S. (1973). *A Solution to a Special Class of Flow Shop Scheduling Problems* (pp. 83–91). Berlin Heidelberg: Symposium on the Theory of Scheduling and Its Applications. Springer.

Shao, W., Shao, Z., & Pi, D.-C. (2020). Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem. *Knowledge-Based Systems, 194*, 1–17.

Shao, W.-S., Shao, Z.-S., & Pi, D.-C. (2021). An Ant Colony Optimization Behavior-Based MOEA/D for Distributed Heterogeneous Hybrid Flow Shop Scheduling Problem Under Nonidentical Time-of-Use Electricity Tariffs. *IEEE Transactions on Automation Science and Engineering*. https://doi.org/10.1109/TASE.2021.3119353

Shao, Z.-S., Shao, W.-S., & Pi, D.-C. (2020). Effective heuristics and metaheuristics for the distributed fuzzy blocking flow-shop scheduling problem. *Swarm and Evolutionary Computation, 59*, 1–17.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research, 64*, 278–285.

Riahi, V., Mostafa Khorramizadeh, M. A., Newton, H., Sattar, A., et al. (2017). Scatter search for mixed blocking flowshop scheduling. *Expert Systems with Application, 79*, 20–32.

Wang, G.-C., Gao, L., Li, X.-Y., et al. (2020). Energy-efficient distributed permutation flow shop scheduling problem using a multiobjective whale swarm algorithm. *Swarm and Evolutionary Computation, 57*, 1–17.

Wang, J.-J., & Wang, L. (2020). A Bi-Population Cooperative Memetic Algorithm for Distributed Hybrid Flow-Shop Scheduling. *IEEE Transactions on Emerging Topics in Computational Intelligence, 99*, 1–15.

Wang, S., Liu, M., & Chu, C. (2015). A branch-and-bound algorithm for two-stage no-wait hybrid flow-shop scheduling. *International Journal of Production Research, 53*, 1143–1167.

Yu, C., Semeraro, Q., & Matta, A. (2018). A genetic algorithm for the hybrid flow shop scheduling with unrelated machines and machine eligibility. *Computers & Operations Research, 100*, 211–229.

Zhang, B., Pan, Q.-K., Gao, L., et al. (2017). An effective modified migrating birds optimization for hybrid flowshop scheduling problem with lot streaming. *Applied Soft Computing, 52*, 14–27.

Zhang, B., Pan, Q.-K., Gao, L., et al. (2019). A Multiobjective Evolutionary Algorithm Based on Decomposition for Hybrid Flowshop Green Scheduling Problem. *Computers & Industrial Engineering, 136*, 325–344.

Zhang, G., Xing, K., & Cao, F. (2018). Discrete differential evolution algorithm for distributed blocking flowshop scheduling with makespan criterion. *Engineering Applications of Artificial Intelligence, 76*, 96–107.

Zhao, F., Zhao, L., Wang, L., et al. (2020). An Ensemble Discrete Differential Evolution for the Distributed Blocking Flowshop Scheduling with Minimizing Makespan Criterion. *Expert Systems with Applications, 160*, 1–21.

Zheng, J., Wang, L., & Wang, J.-J. (2020). A cooperative coevolution algorithm for multiobjective fuzzy distributed hybrid flow shop. *Knowledge-Based Systems, 194*, 1–11.

Ztop, H., Tasgetiren, M. F., Eliiyi, D. T., et al. (2019). Metaheuristic algorithms for the hybrid flowshop scheduling problem. *Computers & Operations Research, 111*, 177–196.