An effective iterative greedy algorithm for distributed blocking flowshop scheduling problem with balanced energy costs criterion

Xue Han, Yuyan Han, Biao Zhang, Haoxiang Qin, Junqing Li, Yiping Liu, Dunwei Gong

Abstract

With the increase in production levels, a pattern of industrial production has shifted from a single factory to multiple factories, resulting in a distributed production model. The distributed flowshop scheduling problem (DFSP) is of great research significance as a frequent pattern in real production activities. In this paper, according to real-world scenarios, we have added blocking constraints and sequence-dependent setup times (SDST) to the DFSP and proposed a distributed blocking flowshop scheduling problem with sequence-dependent setup times (DBFSP_SDST). In a distributed environment, the allocation of resources and utilization have become an urgent problem to be solved. In addition, scheduling problems related to resource conservation have also attracted increasing attention. Therefore, we study DBFSP_SDST and consider minimizing the energy consumption cost of the critical factory (critical factory is the factory with maximum energy consumption cost) under resource balance. To tackle this problem, an effective iterated greedy algorithm based on a learning-based variable neighborhood search algorithm (VNIG) is proposed. In VNIG, an efficient construction heuristic is well designed. Two different local searches based on the characteristics of the proposed problem are developed to enhance the local exploitation by neighborhood searching. A learning-based selection variable neighborhood search strategy is designed to avoid the solution trapping in local optima. By conducting extensive simulation experiments, the proposed VNIG shows superior performance compared with artificial chemical reaction optimization (CRO, 2017), the discrete artificial bee colony algorithm (DABC, 2018), the iterative greedy algorithm with a variable neighborhood search scheme (IGR, 2021), and the evolution strategy approach (ES, 2022).

1. Introduction

With the rapid growth of manufacturing, the production model of multiple factory operations has become the norm. Moreover, the emergence of globalization has also prompted factories to show multiregional, multimode characteristics [1]. Product processing is no longer limited to a factory but several factories, which makes processing much more efficient. The distributed permutation flowshop scheduling problem (DPPSP) has become the choice of many enterprises due to its simple structure and easy operation and has attracted much attention from scholars [2]. The characteristics of the DPFSP are as follows: there are n jobs to be processed in f identical factories, in which m uncorrelated machines exist in each factory. Once a job is assigned to a factory, it cannot be assigned to another factory. The job must be processed on all machines of the factory. Based on the above characteristics, there are two key subproblems in the DPFSP to be solved: how to assign jobs to multiple factories and how to determine the scheduling sequence in each factory.

In the traditional DPFSP, factories are assumed to have infinite buffers. However, in real production activities, there are sometimes no buffers between neighboring machines due to the limitations of factory storage devices [3]. In this situation, a job must wait on the current machine until the next machine is available, which causes the blocking of jobs. For example, in the chemical industry, partially processed jobs (i.e., physical or chemical materials) are held in machines because there is no intermediate storage. In the case of the iron and steel industry, the blocking of ingot in the soaking pit will increase the extra consumption of energy since the blocked ingot requires a
high temperature [3]. In this paper, the blocking constraint is considered in the DPFSP, resulting in a new problem, i.e., the distributed blocking flowshop scheduling problem (DBFSP). Except for the above blocking constraint, the setup time of adjacent jobs is also an important factor in real-world scenarios [4], such as fixtures or tools of machines that may need to be changed when different jobs are processed on the same machine. The abovementioned setup time is related to the operation being processed and the previous operation in the sequence, generally called sequence-dependent setup times (SDST) [5,6]. According to the literature [7,8], both blocking constraints and SDST have been considered simultaneously in PFSP, and it is evident that the study of these two constraints on the flowshop scheduling problem is of practical importance. Therefore, based on the above analysis, combined with the distributed production environment, we investigate the distributed blocking flowshop scheduling problem with sequence-dependent setup time (DBFSP_SDST).

In the literature, most research on the above distributed flowshop scheduling problems only considers economic indicators, such as makespan, tardiness time or earliness time, and relatively few focus on environmental protection or energy consumption or energy consumption cost indicators from a sustainable manufacturing point of view. In practical production, idle machines, setup of machines, and blocking of jobs in each factory often lead to energy consumption costs [3]. For distributed production models, the factories may be located in different regions, which leads to the imbalanced regions energy consumption costs. In the report of the Academician Conference of the Chinese Academy of Sciences in 2021, professor Zhongli Ding, Academician of the Chinese Academy of Sciences, pointed out that if different enterprises in a certain industry cannot coordinate and progress together, it will inevitably lead to cost savings for “doing not act as an enterprise”, resulting in the phenomenon of “bad money drives out good money”. Furthermore, efforts have also been made on minimizing the total energy consumption costs by many scholars [9,10]. Based on the above analysis, we find that it is more reasonable to optimize the total energy consumption costs of the critical factory than the total energy consumption of the critical factory.

Thus, it is necessary to expend much effort to ensure the energy costs balance between different regions and reduce the energy consumption costs of each factory. In view of this, in this paper, the objective is to minimize the total energy consumption costs of each critical factory by assigning jobs to the factories and reasonably scheduling sequence in each factory.

For flowshop scheduling problems, the iterated greedy algorithm (IG) shows great performance [11,12]. Compared with some swarm intelligence algorithms, IG has the characteristics of a simple frame, easy operations, and a powerful neighborhood search capability for deeper mining of solutions [13]. For the DBFSP_SDST with balanced energy costs criterion, there is no relevant literature to propose corresponding algorithms to solve this problem. Therefore, an expanded algorithm based on the IG algorithm is proposed in this paper.

This study has the following two novelties.

Based on the scheduling problem subject to blocking constraints. This study formulates a distributed blocking flowshop scheduling problem with setup times that optimize the balanced energy costs criterion for the first time. The formulation of the above scheduling problem can better reflect real-world applications, thus, compared to those in previous work, that have more practical significance.

Following that, an IG algorithm with variable neighborhoods (VNIG) is designed to solve the DBFSP_SDST. The proposed algorithm has the following trifold features. The first is that the initialization solution is obtained by a variant of MM and NEH2_en. The second is that three different local search strategies are proposed based on the properties of DBFSP_SDST with balanced energy costs criterion. Third, a variable neighborhood search strategy is added to avoid becoming trapped in local optima. The performances of the presented initialization, three local search, and variable neighborhood search strategies are empirically evaluated. The experimental results demonstrate that the proposed strategies can effectively tackle DBFSP_SDST with balanced energy costs criterion by obtaining a good scheduling sequence.

The remainder of this paper is organized as follows. Section 2 describes the literature related to energy-balance DBFSP_SDST. In Section 3, we explain and give examples of DBFSP_SDST with balanced energy costs criterion. The algorithms and innovations proposed in this paper are listed in Section 4. Section 5 gives the experimental results and analysis of the algorithms. In the last section, we conclude the paper and give an outlook on future research directions.

2. Literature review

Compared with the research on the classical DPFSP, little research has been done on the DPFSP_SDST with blocking constraints and balanced energy costs criterion. Most studies have focused on optimizing the DPFSP or DPFSP_SDST with the makespan criterion. The following first reviews the DPFSP and then the DPFSP_SDST. Next, the blocking constraint in the flowshop scheduling problems is described. Finally, the characteristics of our addressed problem are presented.

The DPFSP has emerged in response to globalization and the continuous development of the manufacturing industry. The target of studying DPFSP is to save production materials and improve production efficiency. Since the emergence of the DPFSP, many scholars have developed a series of innovative algorithms to solve it. Heuristics and metaheuristics are studied by Hatami and Ruiz [14]. Naderi and Ruiz proposed a scatter search (SS) method to minimize makespan [15]. For the same objective, Bargouei et al. designed an effectively improved chemical reaction optimization algorithm (CRO) [16]. Fernandez-Viagas et al. employed a bounded-search iterated greedy algorithm based on the specific structure of the DPFSP [17]. Ruiz and Pan proposed an improved iterative greedy algorithm that showed excellent performance in solving the DPFSP [18]. In addition to minimizing makespan, scholars have also studied the DPFSP with other objectives. Meng and Pan proposed three heuristics to optimize the customer satisfaction objective, i.e., the neighborhood descent, the artificial bee colony, and the iterative greedy methods [19]. For the DPFSP with the total flow time criterion, Fernandez-Viagas et al. proposed an iterative improvement algorithm [20]. Subsequently, Pan et al. designed three constructive heuristics and four metaheuristics based on the characteristics of the problem [21]. Recently, Zhang et al. proposed an innovative three-dimensional matrix-cube-based estimation of distribution algorithm (MCEDA) [22].

In real production activities of the DPFSP, many constraints limit the production of products, e.g., sequence-dependent setup time and blocking constraints. Regarding the sequence-dependent setup time, since the machine may generate operations with replacement parts, maintenance, etc., before processing a job, some extra time will be generated. The setup time is related to the job being machined as well as to the previous one. To solve this problem, Parthasarathy et al. [23] proposed an experimental evaluation of heuristics for scheduling in a real-life flowshop with sequence-dependent setup times (SDST) of jobs. Following that, to optimize the problems with SDST, some excellent algorithms, such as ant colony optimization techniques [24], variable domain constructive heuristics [25], constructive heuristics [26], and enhanced migratory bird optimization algorithms [27], have been proposed. Recently, Huang
et al. proposed an improved iterative greedy algorithm [28] and an effective discrete bee colony algorithm [29] to solve the DPFSP with SDST. The DPFSP can be classified into two categories concerning the size of buffers (either infinite or no buffers). The former does not result in job blocking since it has enough intermediate buffers to store uncompleted jobs. Here, the term "blocking" means maintaining a limited capacity of in-process inventories due to finite intermediate buffers. Once blocking occurs, it affects the overall production efficiency of the sequence and increases energy consumption. Therefore, determining job scheduling under blocking constraints becomes very important. To solve the blocking constraint, many researchers have proposed intelligent optimization algorithms. For the DPFSP_SDST, blocking time often arises in each factory due to no storage space between adjacent machines. In this paper, the blocking constraint is considered in DPFSP_SDST, and a new problem, called the distributed blocking flowshop scheduling problem with setup times (DBFSP_SDST), is formed. To solve the above DBFSP_SDST, Zhang et al. proposed a novel hybrid discrete differential evolution (DDE) algorithm for the DBFSP [30]. Next, a hybrid enhanced discrete fruit fly optimization algorithm (HEDFOA) was proposed by Shao [31]. Later, an ensemble discrete differential evolution [32] and an iterated greedy (IG) algorithm [33] were proposed.

Due to the gradual emphasis on resource conservation and environmental protection, research with the goal of green manufacturing has received increasing attention. Scholars have mostly presented their research on energy consumption objectives [32–42]. In recent years, for the energy-efficiency DPFP, Wang et al. proposed a knowledge-based cooperative algorithm (KCA) [34]. Wang and Li studied the multiobjective whale swarm algorithm (MOWSA) [43]. Then, the collaborative optimization algorithm (COA) [44], the innovative 3D matrix cube distribution estimation algorithm (MCEDA) [22], the genetic programming hyperheuristic (GP-HH) algorithm [45], and the improved NSGAII algorithm (INSGAII) [46] were continuously designed and proposed to minimize the energy consumption objective in different research fields. For the distributed models, there are differences in the level of development between regions. To narrow the differences between factories and improve the utilization rate of factories, the resource balance between factories was considered for the first time.

The DBFSP_SDST with balanced energy costs criterion is first considered in this paper; therefore, some literature related to the DPFSP (see Table 1) must be described. Table 1 includes the characteristics and the main contributions of the DPFSP. For the DPFSP with makespan criteria, a improved iterated greedy algorithm [16] and CRO algorithm [18] were proposed. The work in [20] studied the DPFSP with makespan criteria and proposed an enhanced discrete differential evolution algorithm and an iterated greedy algorithm, an effective initialization method, and an enhanced construction method. Wang et al. considered the DPFSP with energy consumption and makespan criteria. The work in [35] addressed mixed no-idle DPFSP and considered sequence-dependent setup times. In [36], an effective discrete artificial bee colony algorithm is proposed to solve the DPFSP with total flowtime minimization.

In summary, for the above literature, we found that research on the DPFSP problem is a current hot topic. DPFSP is a frequent pattern in real production activities and has great research significance. However, the situation faced in real production activities is more complex [47]. To better approach real production activities, we consider the abovementioned constraints related to energy consumption cost indicators in the DPFSP and propose a simple and effective IG algorithm with a variable neighborhood (VNI) for solving the above DBFSP_SDST with balanced energy costs criterion.

### 3. DBFSP_SDST with balanced energy costs criterion

In this section, the mathematical model of the DBFSP_SDST with balanced energy costs criterion is described, and we assume that there are \( J \) jobs to be scheduled in \( F \) factories that have \( M \) machines. In addition, some restrictions may appear in the model:

1. A job can only be processed on one machine at a time.
2. Machines in a factory can only process one job at a time.
3. Each job must be scheduled in process order and cannot change the factory.
4. A job has to be blocked in the current machine until the downstream one is available.
5. An anticipatory and sequence-dependent job setup time is considered on each machine, and an initial setup time is needed if job \( j \) is the first job on a machine.
6. The total energy consumption costs are the sum of the processing energy consumption cost, setup energy consumption cost, and standby energy consumption cost.

In this paper, the objective is to minimize the total energy consumption costs of each critical factory and balance the resources by assigning jobs to the factories and reasonably scheduling the job sequence in each factory. The calculation method and the related notations, decision variables, objective, and constraints in the proposed mathematical model are given below.

#### Notations:

- \( F \): The number of factories.
- \( f \): Index of factories, \( f \in \{1, 2, \ldots, F\} \).
- \( M \): The number of machines in each factory.
- \( m \): Index of machines.
- \( J \): The number of jobs.
- \( j \): The number of jobs.
- \( J \)' Index of jobs, \( j, f \in \{0, 1, \ldots, J\} \), where 0 is the index of the dummy job, which represents the start and end of the job sequence in a factory.

#### Table 1

Review of the works on DPFSP.

<table>
<thead>
<tr>
<th>Authors</th>
<th>Setting</th>
<th>Objective/(Minimizing)</th>
<th>Solution approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barghouti et al. [16]</td>
<td>DPFSP</td>
<td>Makespan</td>
<td>Improved artificial chemical reaction optimization</td>
</tr>
<tr>
<td>Ruiz et al. [18]</td>
<td>DPFSP</td>
<td>Makespan</td>
<td>Effective iterated greedy methods</td>
</tr>
<tr>
<td>Huang et al. [29]</td>
<td>DPFSP</td>
<td>Setup times, constraint and makespan</td>
<td>An iterated greedy algorithm with a restart scheme</td>
</tr>
<tr>
<td>Zhao et al. [32]</td>
<td>DBFSP</td>
<td>Makespan</td>
<td>An ensemble discrete differential evolution algorithm</td>
</tr>
<tr>
<td>Chen et al. [33]</td>
<td>DBFSP</td>
<td>Makespan</td>
<td>An iterated greedy algorithm, an effective initialization method, and an enhanced construction method</td>
</tr>
<tr>
<td>Wang et al. [34]</td>
<td>DPFSP</td>
<td>Energy-efficient and makespan</td>
<td>A knowledge-based cooperative algorithm</td>
</tr>
<tr>
<td>Rossi et al. [35]</td>
<td>Mixed No-idle DPFSP</td>
<td>Sequence-Dependent setup times constraint</td>
<td>A novel constructive heuristic and iterated greedy algorithms</td>
</tr>
<tr>
<td>Pan et al. [36]</td>
<td>DPFSP</td>
<td>Total flow time</td>
<td>A discrete artificial bee colony algorithm</td>
</tr>
<tr>
<td>This paper</td>
<td>DBFSP</td>
<td>Setup times constraint and Energy-balance</td>
<td>An effective iterative greedy with variable neighborhood search strategy</td>
</tr>
</tbody>
</table>
\( p_{j,m} \): Processing time of job \( j \) on machine \( m \).
\( s_{j,f,m} \): Setup time from job \( j \) to job \( f' \) on machine \( m \). An initial setup time \( s_{0,j,m} \) is needed if job \( j \) is the first job on machine \( m \).

\( E_{\text{Process}} \): The energy consumption per unit time of machine \( m \) when the machine processes job \( j \).

\( E_{\text{Setup}} \): The energy consumption per unit time of machine \( m \) when the machine stays in the setup state from job \( j \) to job \( f' \).

\( E_{\text{Idle}} \): The energy consumption per unit time of machine \( m \) when the machine stays in the idle or blocked state.

\( E_{\text{Cj}} \): The cost per unit of energy consumption in factory \( f \).

\( h \): Sufficiently large positive number.

**Decision variables:**

\( C_{j,m} \): The completion time of job \( j \) on machine \( m \).

\( D_{j,m} \): The departure time of job \( j \) on machine \( m \).

\( w_{j,f} \): Binary decision variable, 1 if job \( j \) is assigned to factory \( f \), 0 otherwise.

\( x_{j,f,f'} \): Binary decision variable, 1 if both jobs \( j \) and \( f' \) are assigned to factory \( f \), and job \( j \) is an immediate successor of job \( f \) in factory \( f' \), 0 otherwise.

\( TPE_f \): Total energy consumption costs of all machines in factory \( f \) when they stay in the processing state.

\( TSE_f \): Total energy consumption costs of all machines in factory \( f \) when they stay in the setup state.

\( TIE_f \): Total energy consumption costs of all machines in factory \( f \) when they stay in the idle or blocked state.

\( E_{\text{MAX}} \): Total energy consumption costs of all machines in each critical factory.

**Objective:**

Minimize \( (E_{\text{MAX}}) \) \hspace{1cm} (1)

**Constraints:**

\[ w_{0,f} = 1, \forall f \in \{1,2,\ldots,F\} \] \hspace{1cm} (2)

\[ \sum_{f=1}^{F} w_{j,f} = 1, \forall j \in \{1,2,\ldots,J\} \] \hspace{1cm} (3)

\[ \sum_{j=0}^{J} x_{j,f,f} = w_{j,f}, \forall j \in \{1,2,\ldots,J\}, \forall f \in \{1,2,\ldots,F\} \] \hspace{1cm} (4)

\[ \sum_{j=0}^{J} x_{j,f,f'} = w_{j,f}, \forall j \in \{1,2,\ldots,J\}, \forall f \in \{1,2,\ldots,F\} \] \hspace{1cm} (5)

\[ \sum_{f'=0}^{F} x_{0,f'} = 1, \forall f \in \{1,2,\ldots,F\} \] \hspace{1cm} (6)

\[ \sum_{f'=0}^{F} x_{f',0} = 1, \forall f \in \{1,2,\ldots,F\} \] \hspace{1cm} (7)

\[ C_{j,m} = p_{j,m} + \delta_j, \forall j \in \{1,2,\ldots,J\}, \forall m \in \{1,2,\ldots,M\} \] \hspace{1cm} (8)

\[ D_{j,m} \geq C_{j,m}, \forall j \in \{1,2,\ldots,J\}, \forall m \in \{1,2,\ldots,M\} \] \hspace{1cm} (9)

\[ C_{j,m} = p_{j,m} = D_{j,m-1}, \forall j \in \{1,2,\ldots,J\}, \forall m \in \{2,3,\ldots,M\} \] \hspace{1cm} (10)

\[ C_{j,m} = p_{j,m} = D_{j,m-1} + s_{j,m} + \sum_{f=1}^{F} x_{j,f,f} - 1 \] \hspace{1cm} (11)

\[ C_{j,m} = p_{j,m} = s_{0,j,m} + \sum_{f=1}^{F} x_{0,f} - 1 \] \hspace{1cm} (12)

\[ TPE_f = E_{\text{Cj}} \cdot \sum_{m=1}^{M} \sum_{j=1}^{J} (E_{\text{Process}} \cdot p_{j,m} \cdot w_{j,f}) \cdot \forall f \in \{1,2,\ldots,F\} \] \hspace{1cm} (13)

\[ TSE_f = E_{\text{Cj}} \cdot \sum_{m=1}^{M} \sum_{j=1}^{J} \left( \sum_{f'=1}^{F} E_{\text{Setup}} \cdot s_{j,f,f'} \cdot x_{j,f,f'} \right) \] \hspace{1cm} (14)

\[ TIE_f = E_{\text{Cj}} \cdot \sum_{m=1}^{M} (E_{\text{Idle}} \cdot \sum_{j=1}^{J} (C_{j,m} - x_{j,0,0}) - \sum_{f,m} (p_{j,m} \cdot w_{j,f})) \] \hspace{1cm} (15)

\[ E_{\text{MAX}} \geq TPE_f + TSE_f + TIE_f, \forall f \in \{1,2,\ldots,F\} \] \hspace{1cm} (16)

Eq. (1) minimizes the energy consumption costs of each critical factory, which can balance the energy consumption costs of each factory. Constraint (2) defines that each factory contains a dummy job that represents the start and end of the job sequence in a factory. Constraint (3) guarantees that each job can only be assigned to one factory for processing. Constraints (4), (5), (6), and (7) ensure that each job must have only one immediate predecessor and successor. Constraint (8) enforces that the start processing time of each job on each machine must be greater than or equal to 0. Constraint (9) ensures that the departure time of each job on each machine must be greater than or equal to its completion time. Constraint (10) defines that the start processing time of each job in a machine is equal to its departure time on the previous machine. For job \( j \) and its immediate successor \( f' \) on machine \( m \) in factory \( f \), the start processing time of job \( j \) on machine \( m \) is not less than the departure time of job \( f \) on machine \( m \) plus the setup time \( s_{j,f,m} \), which is ensured by constraint (11). For the first job on machine \( m \) in factory \( f \), the start processing time must be greater than or equal to the initial setup time \( s_{0,j,m} \) and is considered by constraint (12). Constraints (13), (14), and (15) calculate the total processing energy consumption costs of each factory, the total setup energy consumption costs of each factory, and the total standby energy consumption costs of each factory, respectively. Constraint (16) defines the total energy consumption costs of all machines in each critical factory. Our model contains \( F \) job sequences, each starts from a dummy job and ends with another dummy job and represents the job scheduling in a factory.

To clearly understand the processing of calculating \( E_{\text{MAX}} \), we give the equations of the above model to calculate \( E_{\text{MAX}} \) in the case of heuristics and metaheuristics. Suppose factory \( f \) includes \( \delta_f \) jobs that are processed according to job sequence \( \pi_f = \{ \pi_{f,1}, \pi_{f,2}, \ldots, \pi_{f,\delta_f} \} \), where \( \pi_{f,j} \in \{1,2,\ldots,\delta_f\} \), is the job included in factory \( f \). Denote \( \{f, j\} \) as the job index of the \( j \)th job to be processed in factory \( f \). For factory \( f, f = 1,2,\ldots,F \), the completion time and departure time of each job on each machine are calculated according to Eqs. (17)–(19). The energy consumption costs \( TPE_f, TSE_f, \) and \( TIE_f \) of factory \( f \) are calculated according to Eqs. (20)–(22). Eq. (23) gives the expression of \( E_{\text{MAX}} \).

\[ C_{f,j,0} = 0, j = 1,2,\ldots, \delta_f \] \hspace{1cm} (17)
shown in Tables 4 and 5. We adopt the Gurobi solver to solve the above example, and obtain the optimal schedule sequence. The job sequence in each factory is (7, 6, 9), (1, 5) and (4, 3, 8, 2), respectively. The Gantt chart of the above scheduling sequences is given in Fig. 1. The optimization objective of this paper is to minimize the energy consumption costs of the critical job. The energy consumption costs of each factory is calculated as the sum of processing energy cost, setup energy cost, and standby energy cost (in the following, we will refer to the sum of blocking time and idle time as standby time).

For ease of understanding, we give a scheduling case of DBFS. The corresponding energy consumption costs having 9 jobs and 3 factories with 3 machines per factory. The processing time \( p_{j,m} \) and the corresponding energy consumption \( \text{EC}_{p,m} \) of job \( j \) on machine \( m \) are shown in Tables 2 and 3, respectively. The cost per unit of energy consumption in factory \( f \) is \( \text{EC}_{f,m} \) and the corresponding energy consumption \( \text{EC}_{f,m} \) are shown in Tables 4a and 5.

We adopt the Gurobi solver to solve the above example, and obtain the optimal schedule sequence. The job sequence in each factory is (7, 6, 9), (1, 5) and (4, 3, 8, 2), respectively. The Gantt chart of the above scheduling sequences is given in Fig. 1. The optimization objective of this paper is to minimize the energy consumption costs of the critical job. The energy consumption costs of each factory is calculated as the sum of processing energy cost, setup energy cost, and standby energy cost (in the following, we will refer to the sum of blocking time and idle time as standby time).

From the Gantt chart, we can see that the maximum completion time of the sequence is 40. We set the end time of the last machine in each factory to be the end time of all machines. The energy consumption costs \( \text{TPE}_f, \text{TSE}_f \) and \( \text{TIE}_f \) in factory 1 are 260, 77, and 112, respectively. The energy consumption costs \( \text{TPE}_2, \text{TSE}_2 \) and \( \text{TIE}_2 \) in factory 2 are 105, 90, and 237, respectively. The energy consumption costs \( \text{TPE}_3, \text{TSE}_3 \) and \( \text{TIE}_3 \) in factory 3 are 124, 196, and 96, respectively. The total energy consumption costs of factory 1, factory 2 and factory 3 are 449, 432, and 416, respectively, in which the critical factory is the factory with the maximum energy consumption cost, that is, \( \text{EMAX} = 449 \).
4. Iterative greedy algorithm with a variable neighborhood search strategy

For DBFSP_SDST, two problems should be considered simultaneously: how to allocate the sequence to the factories and how to arrange the sequence of jobs within each factor. Therefore, considering the characteristics of DBFSP_SDST with balanced energy consumption costs criterion, we propose a simple iterative greedy algorithm with variable neighborhood search (VNIG). The VNIG is a variant of the IG algorithm. Within the while loop, we design two different local search strategies to adjust the optimization solution. In addition, a learning-based variable neighborhood strategy is designed to avoid the solution falling into local optima.

4.1. Initial solution (MME_en)

NEH2_en has good performance when solving the distributed permutation flowshop scheduling problem [16]. MME is used to generate a good initialization solution for the blocking flowshop scheduling problem [3]. Inspired by the above two heuristics, we proposed the MME_en heuristic, which is the combination of NEH2_en and MME, to solve the distributed blocking flowshop scheduling problem.

The MM algorithm is used to generate a heuristic solution, \( \pi = [\pi^1, \pi^2, \ldots, \pi^k, \ldots, \pi^n] \), by minimizing the critical path length, where \( n \) is the number of jobs. Let \( \varphi = \{1, 2, \ldots, n\} \) be the set of initial jobs. First, the job with the shortest processing time on the first machine is chosen as \( \pi^1 \), and the job with the shortest processing time on the last machine is chosen as \( \pi^n \).
The remaining jobs, \( \{\pi^2, \ldots, \pi^k, \ldots, \pi^{n-1}\} \) are obtained using function (24).

\[
\pi^k = \arg\min_{\phi, \pi} \left( \phi \times \sum_{i=1}^{m-1} |p_i| - p_{(k-1),i+1} + (1 - \phi) \times \sum_{j=k}^{m} p_{j,i} \right),
\]

\[k = 2, 3, \ldots, n - 1\]

where \([k]\) is the job index of the \(k\)th job of the sequence \(\pi\). \(\phi\) is a random number between 0 and 1. After obtaining \(\pi = [\pi^1, \pi^2, \ldots, \pi^n]\) yielded by the MM heuristic, the jobs in \(\pi\) are assigned to factories according to the following process. Put first \(f\) jobs of the sequence, \(\pi\), into each factory one by one. The above operations ensure the uniformity of the allocated quantity. For the remaining jobs of the sequence, we take out the jobs one by one and try to test them in all positions of all factories, and the position \(\text{pos}_s\) in factory \(F_s\), with the minimal objective is selected. At this time, we can guarantee that the energy consumption cost of each factory is as low as possible, but we cannot guarantee that the number of jobs in all factories is the same. Thus, in our paper, we attempt to optimize the scheduling sequences of each factory and search for the optimal solution with minimal energy consumption costs. After insertion, remove a job from \(\text{pos}_s - 1\) or \(\text{pos}_s + 1\) of factory \(F_s\), then try to insert it at all locations in \(F_s\) until you find the lowest energy cost \(E'_{F_s}\). Algorithm 1 gives the steps of MME_en.

<table>
<thead>
<tr>
<th>Algorithm 1 MME_en</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> (\pi, n ; \pi = ) an empty scheduling sequence, and (n) is the number of jobs</td>
</tr>
<tr>
<td>01: (\pi = [\pi^1, \pi^2, \ldots, \pi^n] \rightarrow ) MM strategy</td>
</tr>
<tr>
<td>02: for (i = 1 ) to (f)</td>
</tr>
<tr>
<td>03: Take job (\pi^i) from (\pi) and assign it to factory (F_i)</td>
</tr>
<tr>
<td>04: endfor</td>
</tr>
<tr>
<td>05: for (i = f + 1 ) to (n)</td>
</tr>
<tr>
<td>06: for (i = 1 ) to (f)</td>
</tr>
<tr>
<td>07: Test (\pi^i) in all possible positions in (\pi_i \rightarrow \pi_i ) is the job sequence of factory (F_i)</td>
</tr>
<tr>
<td>08: (E_i = ) the lowest energy cost of factory (F_i) obtained (\pi_i)</td>
</tr>
<tr>
<td>09: (\text{pos}_s = ) the position with (E_i)</td>
</tr>
<tr>
<td>10: endfor</td>
</tr>
<tr>
<td>11: (\pi = \arg\min_{\pi} (E_i))</td>
</tr>
<tr>
<td>12: Insert (\pi^i) at the position (\text{pos}_s) of the sequence (\pi)</td>
</tr>
<tr>
<td>13: Extract a job from position (\text{pos}_s - 1) or (\text{pos}_s + 1) of (\pi)</td>
</tr>
<tr>
<td>14: Insert the selected job in all possible positions of (\pi)</td>
</tr>
<tr>
<td>15: Put the selected job to the best positions with minimal (E'_{F_s}) of (\pi)</td>
</tr>
<tr>
<td>16: endfor</td>
</tr>
<tr>
<td><strong>Output:</strong> (E_{\text{MAX}}, \pi)</td>
</tr>
</tbody>
</table>

4.2. Effective local search strategies

Neighborhood-based local search has an essential role in improving the quality of solutions. Insert and swap operations are widely used as local search strategies because of their simplicity and efficiency. When we solve the problem with the continuous expansion of the flowshop scheduling, the advantages of the iterative improvement strategy based on insertion may be gradually declined compared to the one based on swap operation. This is because the insertion operation needs to move a series of jobs. The time complexity of multilayer loops is very high, i.e., \(O(n^3)\). However, the time complexity of the swap operation is \(O(n^2)\) [48]. If the termination time is the same, the number of insert operations will be smaller than that of swap operations, which will reduce the number of algorithm iterations and make it difficult to further seek a potential solution. Obviously, if the optimization objective is further enhanced, the insert operator will spend too much execution time. Therefore, in this paper, to reduce the time complexity, local perturbation strategies based on swaps are adopted to improve the local search ability of the solution.

Based on the distributed feature of DBFSP_SDST, two local search strategies based on swap permutation are proposed, i.e., critical and other factories job swapping (Exter_CriticalFactory_Swap) [29] and random factory job block swapping (LS_JBS). The first one randomly selects two jobs from critical and other factories to perform the swap operator. The last one is to swap the job block with minimal energy consumption cost and a random job block and avoid destroying the job block. Based on the above description, the proposed local search strategies can be given as follows.

(1) Local search based on critical and other factory job swapping

The energy consumption cost of the critical factory has a direct influence on the total energy consumption costs. Thus, in this paper, we propose a local search based on single job swapping within the two factories, named Exter_CriticalFactory_Swap. Exter_CriticalFactory_Swap is a swap operation of critical and other factory. Two jobs, \(j_{1}\) and \(j_{2}\) are randomly selected from a critical factory and random factory, respectively, and swap the two jobs. (See Algorithm 2)

<table>
<thead>
<tr>
<th>Algorithm 2 Exter_CriticalFactory_Swap</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> (\text{Crit} = 0, F_s, n_{\text{max}}) is an initial solution</td>
</tr>
<tr>
<td>01: Find a critical factory (F_{s1}) with (\text{EMAX}) and a random factory (F_s) selected from ({F_1, F_2, \ldots, F_s} / F_{s1})</td>
</tr>
<tr>
<td>02: While (\text{Crit} &lt; n) do</td>
</tr>
<tr>
<td>03: (n_{\text{max}} = \infty)</td>
</tr>
<tr>
<td>04: job(<em>{1}) = a job randomly selected from (F</em>{s1})</td>
</tr>
<tr>
<td>05: job(_{2}) = a job randomly selected from (F_s)</td>
</tr>
<tr>
<td>06: Swap job (j_{1}) and (j_{2})</td>
</tr>
<tr>
<td>07: (E_{\text{MAX}}) = the energy consumption cost of (F_{s1}) after swapping</td>
</tr>
<tr>
<td>08: if (E_{\text{MAX}} &lt; \text{EMAX})</td>
</tr>
<tr>
<td>09: (\text{EMAX} = \text{EMAX} *)</td>
</tr>
<tr>
<td>10: Find a critical factory (F_{s1}) with (E'_{\text{MAX}})</td>
</tr>
<tr>
<td>11: Random select a factory (F_s) from ({F_1, F_2, \ldots, F_s})</td>
</tr>
<tr>
<td>12: (\text{Crit} = 0)</td>
</tr>
<tr>
<td>13: else</td>
</tr>
<tr>
<td>14: (\pi = n_{\text{max}}) and (\text{Crit} = \text{Crit} + 1)</td>
</tr>
<tr>
<td>15: endwhile</td>
</tr>
<tr>
<td><strong>Output:</strong> (E_{\text{MAX}}, \pi)</td>
</tr>
</tbody>
</table>
In Algorithm 3, define the length \( d \) of the job block, randomly select the factory \( F_i \) from \( \{ F_1, F_2, \ldots, F_n \} \), and record the energy consumption cost \( E_k \) of \( F_i \) (\( E_k = TPE_k + TSE_k + TIE_k \)). When the number of jobs \( n \) in \( F_i \) is less than or equal to \( 3 \times d \), a single job exchange (see lines 3–5) is carried out. Otherwise, job block \( \text{block}_1 \) and job block \( \text{block}_2 \) (\( \text{block}_1 \cap \text{block}_2 = \emptyset \)) are randomly selected from sequence \( \pi_0 \). Swap \( \text{block}_1 \) and \( \text{block}_2 \) (see lines 6–9). Finally, the acceptance criterion is executed (see lines 10–16).

### Algorithm 3 LS_JBS

**Input**: \( d, \text{count} = 0 \), \( \pi_0 \) (\( \pi_0 \) is the sequence of \( F_i \))

1. \( F_i \) = random selected a factory from \( \{ F_1, F_2, \ldots, F_n \} \), \( \pi_i = \pi_0 \), \( E_k \) is the energy consumption cost of \( F_k \)
2. while (\( \text{count} < \text{Factory number} \))
   3. if \( \pi_i \Rightarrow \text{next} \) do
   4. \( \text{job}_i, \text{job}_j \) = randomly selected a job from \( F_i \) respectively // \( \text{job}_i \) \& \( \text{job}_j \)
   5. swap \( \text{job}_i \) and \( \text{job}_j \). \( E_k \) is the energy consumption cost of \( F_k \) after swapping
   6. else
   7. \( \text{block}_1, \text{block}_2 \) = randomly selected a job from \( \pi_i \) respectively // \( \text{block}_1 \cap \text{block}_2 = \emptyset \)
   8. swap \( \text{block}_1 \) and \( \text{block}_2 \). \( E_k \) is the energy consumption cost of \( F_k \) after swapping
   9. endif
10. if \( \pi_i < \text{count} + d \) do
11. \( \pi_i = \pi_{i+1} \)
12. end if
13. else
14. \( \pi_i = \pi_0 \)
15. \( \text{count} = \text{count} + 1 \)
16. end if
17. endwhile

**Output**: \( \pi_i, \pi_0 \)

### 4.3. Improved variable neighborhood search strategies

In the local search, the solution easily falls into the local optima because of the slight disturbance. In view of this, if the quality of the solution is not improved after \( \text{num} \) generations, we will execute an improved variable neighborhood search (IVNS, for short) strategy to avoid the solution trapping into local optima. Furthermore, to enhance the diversity or globality of the solution, we first use destruction and reconstruction strategies to disturb the current solution. After this, we try to adopt different operators to produce a promising solution, which can improve the quality of the solution. In this paper, one of the three strategies, i.e., Exter_CriticalFactory_Swap (shown in Algorithm 2), Insert_K, and Insert_D, is chosen by the learning-based selection strategy in the IVNS strategy shown in Algorithm 4.

In Algorithm 4, \( \text{cnt} \) records the number of cycles of IVNS, \( R_0 \), \( R_1 \), and \( R_2 \) record the number of executions of Exter_CriticalFactory_Swap (shown in Algorithm 2), Insert_K, and Insert_D, respectively. \( PL \) is the number of jobs that are removed and reinserted in destruction and reconstruction. The first step is to perform destruction and reconstruction strategies to disturb the current solution. For the first \( z \) times of IVNS, the random strategy is adopted (see lines 21–30). If the solution is improved, the corresponding \( R_i = R_i + 1 \). After applying IVNS several times, the strategy will be selected according to the value of the probability \( P_i \).

#### Algorithm 4 IVNS

**Input**: \( X, \text{cnt}, R_0, R_1, R_2, PL, E \) // \( E \) is the energy consumption cost of \( R \)

1. \( x^* = \text{Destruction} (\pi_*, PL) \)
2. if (\( \text{cnt} < z \)) // \( z \) is a threshold value
3. \( R = \text{randomly obtain value from 0-2} \)
4. \( \text{cnt} = \text{cnt} + 1 \)
5. if \( R = 0 \) do
6. \( \pi \leftarrow \text{Exter_CriticalFactory_Swap} (\pi^*) \)
7. if \( E \) is improve
8. \( R_0 = R_0 + 1 \)
9. else \( R = 1 \) do
10. \( \pi \leftarrow \text{Insert_K} (\pi^*) \)
11. if \( E \) is improve
12. \( R_1 = R_1 + 1 \)
13. else \( R = 2 \) do
14. \( \pi \leftarrow \text{Insert_D} (\pi^*, i) \)
15. if \( E \) is improve
16. \( R_2 = R_2 + 1 \)
17. end if
18. end if
19. end if
20. end while
21. end if
22. \( \pi \leftarrow \text{randomly select a job from} \ F_i \)
23. if \( (r < \pi_i) \)
24. \( R = 0 \)
25. \( R_0 = R_0 + 1 \)
26. else if \( (r < \pi_i) \)
27. \( R = 1 \)
28. \( R_1 = R_1 + 1 \)
29. else if \( (r < \pi_i) \)
30. \( R = 2 \)
31. \( R_2 = R_2 + 1 \)
32. end if
33. end if
34. end if
35. endwhile

**Output**: \( \pi^* \)

In the IG algorithm, the destructive and reconstruction operations are used to largely disturb the current solution and effectively avoid the algorithm from falling into a local optimum. In our algorithm, the destructive and reconstruction (DR, for short) strategies are applied in IVNS as the first step. The framework of DR is shown in Algorithm 5.

#### Algorithm 5 Destruction and reconstruction

**Input**: \( X, PL \) // \( PL \) is the number of jobs that be removed and reinserted in DR

**Define**: \( \pi^* = \pi_0 \), \( \text{Cut} = 0 \) // \( \text{Cut} \) is a counter

1. while \( \text{Cut} < \text{num} \) do // Destruction
2. \( F_i = \text{random select factory from} \ \{ F_1, F_2, \ldots, F_n \} \)
3. if \( (\pi > 1) \) // \( \pi \) = the number of jobs in \( F_i \)
4. \( \text{job} = \text{random select job from} \ F_i \)
5. \( \text{Put} \ \text{job} \ \text{into} \ \pi^* \)
6. \( \text{Delete} \ \text{job} \ \text{from} \ \pi^* \)
7. \( \text{Cut} = \text{Cut} + 1 \)
8. end while

**Output**: \( \pi^* \)

The DR strategy is divided into two parts: destruction and reconstruction. For the destruction, factory \( F_i \) is randomly selected from \( \{ F_1, F_2, \ldots, F_n \} \). If factory \( F_i \) has more than one job, a job will be randomly selected from \( F_i \) and put into the empty collection \( \pi_{temp} \), and correspondingly, the selected job is deleted from the original sequence. The above operation is performed \( PL \).
times so that there are PL jobs in set π temp (see lines 1–9). For the reconstruction operation, we remove job j from π temp one by one, try to insert it at all positions of all factories, and find the best position posi with the minimum energy consumption cost (see lines 12–16). The above step is repeated until all the jobs in sequence π temp have been removed.

Insert K (π) refers to the insert operation in a factory. The factory Fk is randomly selected from the set {F1, F2, . . . , Fn}, and the job sequence of Fk is πk. Calculate the sum of the energy consumption cost of each job processing at all machines in the selected factory, and the formula is JobE[πk,i]. We believe that the movement of a job with large processing energy consumption cost is more likely to have an impact on the quality of the solution. Therefore, the job with the maximum energy consumption cost is selected from the factory, denoted as Maxjob. Next, Maxjob is inserted into all positions of the solution from all the factories, and the position with the minimal value of total energy consumption cost is selected.

Algorithm 6 Insert based on critical job (Insert K)

Input: πi
01: Fk ← random selected factory from (F1, F2, . . . , Fn)
02: for j = 0 to n do //n is the job number of πi allocated in Fk
03: calculated JobE[πk,i] = ∑(πk,i,Fk,j) E[Cj]
04: endfor
05: Find the job, Maxjob, with maximum energy cost from πk
06: Test insert Maxjob in all possible positions of all factories
07: Put Maxjob in the best position i with the lowest energy cost θi

Output: EMAX, πi

Insert D (π, l) states a job block dispersal insertion method to increase the diversity of the solution. Compared with the job block insertion operation, Insert D (π, l) makes it easier to jump out of the local optimum. The job block with maximum energy consumption cost is obtained from a random factory. Next, each job in the block is reinserted in the best position of all the factories one by one. In Algorithm 7, first, define the length l of the job block. Then, randomly select the factory Fk in {F1, F2, . . . , Fn}, find the block of jobs with maximum energy consumption cost from Fk, and store it in the array Block. In lines 3–7, the jobs are taken from Block and reinserted at all positions in all the factories. The position with the minimal total energy consumption cost is found. The above steps are repeated until all the jobs in the job block are inserted.

Algorithm 7 Job block dispersal insertion strategy (Insert D)

Input: n, l // l is the number of the job block
01: Fk ← random select factory from (F1, F2, . . . , Fn)
02: Block ← the job block with most energy cost in l
03: for i = 0 to l do //l is the number of compared algorithms
04: Get the job Block[i] in the Block
05: Test insert Block[i] in all possible positions of all factories
06: Insert job Block[i] in the position resulting with the lowest energy cost Ei
07: endfor

Output: EMAX = max EM, πi

5. Experiments and analysis

5.1. Experiment settings

In this section, we validate the performance of the VNIG. Because there are few algorithms for solving DBFSP_SDST with balanced energy cost criterion. Therefore, we selected some algorithms that were used to solve the DFSP that are closely related to the DBFSP_SDST. The compared algorithms are artificial chemical reaction optimization (CRO) [16], the discrete artificial bee colony algorithm (DABC) [36], the iterative greedy algorithm with a restart scheme (IGR) [28], and an evolution strategy approach (ES) [49]. In our experiments, the following comparison was performed:

1. Verification of the MILP model
2. Sensitivity study on three parameters
3. Comparison of the proposed MME_en and NEH2_en.
4. Validate the effectiveness of IVNS
5. Comparison results between the VNIG algorithm and the existing four compared algorithms.

In this paper, we choose 90 instances, where the number of jobs n comes from the set {100, 200, 300, 400, 500}, the number of machines m comes from the set {5, 8, 10}, and the number of factories f is {2, 3, 4, 5, 6, 7}. Thus, 90 different combinations can be obtained by combining n, m, and f. The processing data are generated as follows. The values of setup time and processing time are in the range of [1, 99], and we set the energy consumption per unit of processing, setup, and standby as in the range of [4, 6], [1, 3], [1, 2], respectively. The cost per unit of energy consumption in each factory is in the range of [1, 10].

For the termination criterion of all the compared algorithms, the same maximal elapsed CPU time of TimeLimit = t × n × m milliseconds, where n represents the number of jobs, m refers to the number of machines, and t is equal to 2 and 3. Thus, the computation time can be adjusted for different sizes of instances and the value of t. To evaluate the performance of the proposed algorithm, we choose the energy consumption cost objective and the relative percentage increase (RPI) as evaluation indicators. The formula for calculating RPI is as follows.

\[
RPI = \frac{M_i - M_{best}}{M_{best}} \times 100
\]  

where \(M_i\) is the average energy consumption cost value of the ith algorithm and \(M_{best}\) is the optimal value obtained by all the algorithms. A smaller value of RPI means a better performance.

We also used normalization to process the results. Normalization reduces the values to [0, 1] in equal proportions, aiming to make the differences between the values clearer. The normalization formula is as follows.

\[
y = \frac{x - \min(x_i)}{\max(x_i) - \min(x_i)}, \quad 1 < i < \sigma
\]  

where \(y\) is the value after normalization and \(x\) is the value that needs to be processed. \(\sigma\) is the number of compared algorithms. \(\min(x_i)\) and \(\max(x_i)\) denote the minimum and maximum of the values to be processed, respectively.

In this study, all the algorithms adopt the same maximal elapsed CPU time with the unit of milliseconds as the termination criterion. All the experiments should be conducted and compared under the same or stricter conditions. Variety algorithms are implemented on different PCs, and the implementation settings will be different. In this circumstance, using execution time as the stopping criterion will no longer be reliable since execution time may be affected by the operating system and other applications running during the experiments. Thus, all the algorithms are written in Visual C++ 2019, and the same library functions are adopted in this study to make a fair comparison. For their implementations, all the algorithms are realized on a PC with Pentium (R) Dual 2.9 GHz and 8 G memory, in which the operating system is Microsoft Windows 7 X 64. In addition, the same background running environment is employed, the background processes that may occupy system resources are closed, and no other programs are executed in parallel while implementing an algorithm.
5.2. Verification of the MILP model

In this section, we select eight instances with small sizes to verify the effectiveness of MILP using the Gurobi solver [48]. In Table 6, F, J, M represents the numbers of factories, jobs and machines. We set the running time to 1000 s and 3600 s to ensure sufficient time to search for the solution. Table 6 gives the values of the gap, running time, and energy consumption cost obtained by Gurobi. In addition, the values of energy consumption cost and running time obtained by our VNIG algorithm are listed in Table 6. Gap = 0 means that the optimal solution is found for the problem. For a minimization model, Gap is computed as (ObjVal - ObjBound)/ObjVal, where ObjVal is the objective value for the current solution, and ObjBound is the lower bound that gives a bound of the best possible objective. Thus, if the gap is not equal to 0, it does not mean that no optimal solution is found.

As seen from Table 6, the optimal solution will be found by Gurobi when the size of instance is small, i.e., 3_6_3, 3_7_3, 3_9_3, and 3_10_3 instances. For 4 out of 10 instances, the values of the energy consumption cost obtained by Gurobi and VNIG are the same. However, the computation time of VNIG is far less than that of Gurobi. For 5 out of 10 instances, the values of the energy consumption cost yielded by Gurobi are good, suggesting that the Gurobi solver can obtain better solutions in small-scale examples than those of VNIG. However, with the increasing scale of the instances, the solutions obtained by VNIG are gradually better than those obtained by the Gurobi solver, and it takes less time. Therefore, we believe that the VNIG solver is more suitable than the Gurobi solver for solving large-scale and complicated instances.

5.3. Parameters calibration

To demonstrate the effectiveness of the proposed algorithm, sensitivity analyses of the five parameters are first conducted. The five parameters are \( d \) (the length of the job block swapping), \( l \) (the length of the job block insertion), num (the execution number of the unimproved solution), PL (the number of jobs that are removed and reinserted in destruction and reconstruction), and \( z \) (the threshold value used for the random selection strategy in Algorithm 5). To better determine the values of the parameters, we adopt the Taguchi experimental method to calibrate them.

Table 7 shows the five factor levels of each parameter, i.e., \( d \in \{1, 2, 3, 5, 7\} \), \( l \in \{1, 3, 4, 6, 9\} \), num \( \in \{1, 3, 5, 8, 10\} \), PL \( \in \{2, 3, 4, 5, 9\} \), and \( z \in \{10, 30, 50, 70, 90\} \). We obtained 25 combinations by the orthogonal table listed in Table 8. For the sake of fairness, we select eight instances with different sizes, e.g., 100 × 2, 200 × 8, 3, 200 × 10, 6, 300 × 5 × 4, 300 × 8 × 7, 400 × 10 × 4, 500 × 5 × 2 and 500 × 8 × 5. Each instance is run 20 times independently under the same conditions, and the corresponding RPI values are obtained. Subsequently, the mean RPI values of the five instances with different combinations of parameters are integrated. According to the obtained RPI, the trend of the factor level is plotted in Fig. 2.

Table 9 lists the significance level of each parameter according to the average RPI value of each scale instance, where Delta measures the size of the effect by taking the difference between the maximum and minimum average RPI of the four factors. A larger value of Delta generally indicates a more significant influence. In addition, a smaller Rank value means a larger difference for different values of the parameter.

As seen from Table 9, the parameter PL has the greatest impact on the algorithm, followed by \( d, l, \text{num} \) and \( z \). From Fig. 2, the parameter PL has the best effect when the value is 2. As the number of broken jobs increases, the efficiency of the algorithm becomes low. We believe that too much job block destruction may lead to the destruction of excellent job sequences, which degrades the performance of the algorithm. When \( d = 2 \), the performance of VNIG is great. With the increase in job block length, the algorithm becomes progressively less effective. The
reason may be that the current solution in a factory may be an approximated optimal solution before disturbing; thus, a slight disturbing should be done. However, if the length of the job block is large, the disturbance to the sequence will be very large by swapping the job block, which leads to the generation of a bad solution. For the length $l$ of the job block insertion, the algorithm obtains the minimal mean RPI value when the value of $l$ is 4. The reason may be that if the length of the job block is small, the disturbance to the sequence will be small, which leads to poor diversity. When $\text{num} = 5$, the VNIG shows great performance. We believe that an $\text{num}$ value that is too small will make the local search strategy not fully work, and an $\text{num}$ value that is too large will not jump out in time when the solution falls into a local optimum, which will take more time. Therefore, it is reasonable that the value of $\text{num}$ is 5. Parameter $z$ has a small impact on the algorithm. The performance of the algorithm is good when $z = 30$. According to the above experimental results and analysis, we set the parameters as $d = 2$, $l = 4$, $\text{num} = 5$, $PL = 2$ and $z = 30$.

In addition, we calibrated the parameters of the four compared algorithms using the Taguchi experimental method. The calibration results are shown in the supplementary data.

5.4. Comparison of the results of NEH2_en and MME_en

NEH2_en proposed by Ruiz and Pan can generate high-quality solutions [18] and show better performance than NEH. Therefore, based on this, we combined the MM and NEH2_en to propose an efficiency MME_en strategy. To evaluate the performance of the proposed initialization strategy, we equip the developed algorithm with NEH2_en and MME_en heuristics. The experimental results are shown in Fig. 3 when $t = 2$ and 3.

From Fig. 3, the RPI value of MME_en is lower than that of NEH2_en, which indicates that the former is more efficient in seeking promising solutions for the DBFSP. The results suggest that the MM heuristic has a better performance in optimizing the makespan of BFSP than other heuristics. The reason is that MM selects the job based on the blocking time on these machines by utilizing the shortest critical path, which can effectively deal with the blocking constraint.

5.5. Validation of the effectiveness of the variable neighborhood search strategy based on the learning method

To comprehensively evaluate the performance of the proposed improved learning-based variable neighborhood search strategy, we test the performance of learning-based IVNS and IVNIG with only applying Exter_CriticalFactory_Swap, Insert_K, and Insert_D...
strategies. For the sake of convenience, the IVNS with only applying Exter_CriticalFactory_Swap, Insert_K, and Insert_D strategies is named as EC_swap, Insert_K, and Insert_D strategies, respectively. They run the same test instances under the same conditions.

From Fig. 4(a), we observe that the difference in the performance between EC_Swap and Insert_D is not significant except that the performance of Insert_K was slightly worse, and the performance of IVNS is best among the above compared operators. Except for that, we test the scenario with jobs of different sizes. From Fig. 4(b), EC_Swap shows great performance when the numbers of jobs are 100, 300, 400 and 500. Insert_K can obtain the great solutions when the number of jobs is 200, and the performance of Insert_D is excellent when the number of jobs is 400 and 500. However, considering the above three operators in the variable neighborhood search strategy based on the learning method can obtain superior results in most situations. The reason might be that the different operators can generate more promising solutions, which can improve the exploitation of an algorithm by disturbing the current solution.

5.6. Comparison results between the VNIG algorithm and the five compared algorithms

In this section, the five algorithms, i.e., CRO [16], DABC [36], IGR [28], ES [49] (the parameters of the comparison algorithm were calibrated according to the method in Section 5.3.), and the proposed VNIG is compared for t = 2 and t = 3 (t is the variable used to adjust the running time; the larger the value of t is, the longer the running time of the algorithm), respectively. The experimental results are listed in Tables 10 and 11, respectively, where a row represents the results obtained by different algorithms in terms of the average and RPI values of the energy consumption cost objective. In addition, Figs. 5 and 6 display the maximum and minimum values for 90 instances obtained by all the compared algorithms. The confidence intervals and the evolutionary curve of all the compared algorithms are given.
in Figs. 7 and 8 to show their identification and convergence, respectively.

### Table 10
Energy consumption costs of the compared algorithms when $t = 2$.

<table>
<thead>
<tr>
<th>Job</th>
<th>m + t</th>
<th>DABC$^{\text{RPI}}$</th>
<th>VNIG</th>
<th>LGR$^{\text{RPI}}$</th>
<th>ES$^{\text{RPI}}$</th>
<th>CRO$^{\text{RPI}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 + 2</td>
<td>107.683.7</td>
<td>9.33</td>
<td>98.812.1</td>
<td>0.32</td>
<td>99.538.8</td>
<td>1.06</td>
</tr>
<tr>
<td>8 + 2</td>
<td>241.840.8</td>
<td>11.99</td>
<td>217.059.2</td>
<td>0.51</td>
<td>218.203.9</td>
<td>1.04</td>
</tr>
<tr>
<td>10 + 2</td>
<td>263.729.6</td>
<td>9.75</td>
<td>241.179.7</td>
<td>0.36</td>
<td>243.571.4</td>
<td>1.36</td>
</tr>
<tr>
<td>5 + 3</td>
<td>97.356.1</td>
<td>10.07</td>
<td>89.076.8</td>
<td>0.71</td>
<td>89.212.7</td>
<td>0.21</td>
</tr>
<tr>
<td>8 + 3</td>
<td>259.707.4</td>
<td>10.72</td>
<td>235.782.3</td>
<td>0.52</td>
<td>237.808.2</td>
<td>1.38</td>
</tr>
<tr>
<td>10 + 3</td>
<td>375.025.8</td>
<td>9.85</td>
<td>342.621</td>
<td>0.36</td>
<td>345.494.1</td>
<td>1.20</td>
</tr>
<tr>
<td>5 + 4</td>
<td>67.717.1</td>
<td>11.00</td>
<td>61.304.9</td>
<td>0.49</td>
<td>62.149.7</td>
<td>1.87</td>
</tr>
<tr>
<td>8 + 4</td>
<td>274.612.6</td>
<td>10.68</td>
<td>248.730.8</td>
<td>0.23</td>
<td>251.984.5</td>
<td>1.56</td>
</tr>
<tr>
<td>10 + 4</td>
<td>466.432.7</td>
<td>10.65</td>
<td>405.018.9</td>
<td>0.38</td>
<td>410.550.4</td>
<td>1.06</td>
</tr>
<tr>
<td>5 + 5</td>
<td>122.858.8</td>
<td>10.48</td>
<td>111.363.8</td>
<td>0.61</td>
<td>111.236.2</td>
<td>2.28</td>
</tr>
<tr>
<td>8 + 5</td>
<td>413.736.2</td>
<td>11.04</td>
<td>374.732.2</td>
<td>0.57</td>
<td>379.404.7</td>
<td>1.83</td>
</tr>
<tr>
<td>10 + 5</td>
<td>220.447.1</td>
<td>11.38</td>
<td>198.995.1</td>
<td>0.54</td>
<td>201.062.5</td>
<td>1.5</td>
</tr>
<tr>
<td>5 + 6</td>
<td>707.593.5</td>
<td>11.29</td>
<td>63.906.5</td>
<td>0.84</td>
<td>64.223.0</td>
<td>1.14</td>
</tr>
<tr>
<td>8 + 6</td>
<td>246.534.2</td>
<td>11.60</td>
<td>221.701.5</td>
<td>0.36</td>
<td>224.615.0</td>
<td>1.68</td>
</tr>
<tr>
<td>10 + 6</td>
<td>218.040.1</td>
<td>10.96</td>
<td>197.194.5</td>
<td>0.35</td>
<td>199.564.2</td>
<td>1.56</td>
</tr>
<tr>
<td>5 + 7</td>
<td>99.387.4</td>
<td>12.41</td>
<td>89.111.1</td>
<td>0.79</td>
<td>89.783</td>
<td>1.55</td>
</tr>
<tr>
<td>8 + 7</td>
<td>741.817.6</td>
<td>10.46</td>
<td>672.502.1</td>
<td>0.14</td>
<td>678.004.2</td>
<td>0.95</td>
</tr>
<tr>
<td>10 + 7</td>
<td>974.158.9</td>
<td>9.68</td>
<td>892.167.2</td>
<td>0.14</td>
<td>901.976.1</td>
<td>0.17</td>
</tr>
<tr>
<td>Mean</td>
<td>291.184.2</td>
<td>10.72</td>
<td>264.539.4</td>
<td>0.44</td>
<td>267.232.8</td>
<td>1.47</td>
</tr>
</tbody>
</table>

### Table 11
Pareto and average energy consumption cost value of the algorithms for 90 different instances. From the

(continued on next page)
results, the VNIG obtains the best results for most instances. The performance of the ES algorithm is second only to VNIG. The IGR algorithm performs worse than VNIG and ES in solving the DBFSP with balanced energy cost. Overall, the VNIG substantially outperforms the compared algorithms for solving the DBFSP_SDST with balanced energy cost. It may be that the proposed variable local search can better explore unknown neighborhoods and prevent the solution from falling into a local optimum.

Figs. 5 and 6 report the minimum and maximum values of 90 instances yielded by all the compared algorithms. Because the energy consumption cost value is large, the gap between the results of different algorithms is small when the results are displayed in a figure. Therefore, to show the gap between algorithms more clearly, we normalize the results using Eq. (26). The purpose is to reduce the values in equal proportion and to clearly show the differences. The minimum and maximum values are shown in Figs. 5–6 when \( t = 2 \) and \( t = 3 \), respectively. From Figs. 5–6, we know that the maximum values among all the algorithms are obtained by DABC, so it can be inferred that this algorithm has the worst performance in solving the DBFSP_SDST with balanced energy cost. CRO shows large fluctuations on some small instances and gradually stabilizes as the size of the instances increases.

Meanwhile, we can conclude that the performance of ES is better than that of IGR. As a whole, the values obtained by the VNIG proposed in this paper are excellent in most cases, and we can consider that the performance of VNIG in solving DBFSP_SDST is the best among the compared algorithms.

To have a clear identification of the experimental results, we give the ANOVA of all the algorithms. As shown in Fig. 7, the means plots and interactions plots with 95% LSD intervals represent the average level and overall performance of the algorithms. From Fig. 7, it can be seen that the VNIG proposed in this paper is better than the compared algorithms. The performance of the ES algorithm is worse than that of VNIG but better than that of IGR. In addition, the performance of CRO and IGR is better than that of DABC. We can assume that the algorithm with outstanding local search capability can show superior performance in solving DBFSP_SDST.

To better demonstrate the convergence of the proposed algorithm, we chose scales of 100 \times 10 \times 3, 200 \times 10 \times 3, 300 \times 10 \times 3, and 500 \times 8 \times 5 as examples and plotted the evolutionary curves. The algorithms are running at \( t = 10 \) when the instance size is small, and the large instance is run at \( t = 20 \). The result is shown in Fig. 8, where the X-axis represents the running time (unit: seconds) and the Y-axis represents the total energy consumption cost. All comparison algorithms are listed by different lines. According to the analysis above, we can conclude that the performance of DABC is poor. Therefore, to better show the evolutionary trend of other algorithms, we have removed DABC in Fig. 8. From the four instances, the convergence of the proposed algorithm VNIG is the most rapid among the four algorithms. The VNIG can obtain an excellent initial solution and obtain the best result among the compared algorithms. From Fig. 8, we observe that VNIG can always get excellent solutions in different sizes of examples, which further demonstrates the effectiveness of the VNIG algorithm in solving the DBFSP_SDST with balanced energy cost.

**Remarks.** As the above experimental results and analysis show, the proposed VNIG algorithm is an effective algorithm for solving the DBFSP_SDST with balanced energy cost. The reasons can be concluded as follows. (1) The MM_en strategy combining the MM algorithm and NEH2_en makes the algorithm obtain an excellent initial solution, and a high-quality initial solution is of great importance for the improvement of the solution. (2) Several local search strategies based on energy consumption cost can enhance the neighborhood search ability to further improve the quality of the solution. (3) The variable neighborhood search strategy based on the learning selection method is the key operation that enables increasing the diversity of VNIG by avoiding the solution from falling into local optima in the large-scale instance.
### Table 11
Energy consumption costs of the compared algorithms when $t = 3$.

<table>
<thead>
<tr>
<th>Job</th>
<th>$m \times t$</th>
<th>DABC$^{2018}$ avg</th>
<th>DABC$^{2018}$ RPI</th>
<th>VNIC avg</th>
<th>VNIC RPI</th>
<th>IGR$^{2020}$ avg</th>
<th>IGR$^{2020}$ RPI</th>
<th>ES$^{2021}$ avg</th>
<th>ES$^{2021}$ RPI</th>
<th>CRG$^{2017}$ avg</th>
<th>CRG$^{2017}$ RPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>5 \times 2</td>
<td>467 826.8</td>
<td>10.41</td>
<td>425 341.6</td>
<td>0.38</td>
<td>431 018</td>
<td>1.72</td>
<td>427 832.8</td>
<td>0.97</td>
<td>428 566.6</td>
<td>1.14</td>
</tr>
<tr>
<td></td>
<td>8 \times 2</td>
<td>625 155.6</td>
<td>11.14</td>
<td>564 759.0</td>
<td>0.41</td>
<td>570 072</td>
<td>1.35</td>
<td>569 608.8</td>
<td>1.27</td>
<td>569 234.7</td>
<td>1.20</td>
</tr>
<tr>
<td></td>
<td>10 \times 2</td>
<td>228 768.8</td>
<td>10.23</td>
<td>207 919.6</td>
<td>0.18</td>
<td>210 942.9</td>
<td>1.63</td>
<td>210 496.3</td>
<td>1.42</td>
<td>210 500.5</td>
<td>1.42</td>
</tr>
<tr>
<td></td>
<td>5 \times 3</td>
<td>132 450.2</td>
<td>14.10</td>
<td>156 668.6</td>
<td>0.50</td>
<td>157 076.8</td>
<td>1.07</td>
<td>157 089.5</td>
<td>1.03</td>
<td>157 057.5</td>
<td>1.55</td>
</tr>
<tr>
<td></td>
<td>8 \times 3</td>
<td>461 279</td>
<td>10.50</td>
<td>418 654.8</td>
<td>0.29</td>
<td>423 876.1</td>
<td>1.54</td>
<td>422 439.9</td>
<td>1.19</td>
<td>422 712.6</td>
<td>1.26</td>
</tr>
<tr>
<td></td>
<td>10 \times 3</td>
<td>458 603.1</td>
<td>8.33</td>
<td>424 254.6</td>
<td>0.21</td>
<td>427 891.7</td>
<td>1.07</td>
<td>426 746.2</td>
<td>0.80</td>
<td>428 246.8</td>
<td>1.16</td>
</tr>
<tr>
<td></td>
<td>5 \times 4</td>
<td>151 309.6</td>
<td>10.47</td>
<td>137 507.2</td>
<td>0.39</td>
<td>138 684.7</td>
<td>1.25</td>
<td>138 037.8</td>
<td>0.78</td>
<td>139 236.8</td>
<td>1.65</td>
</tr>
<tr>
<td></td>
<td>8 \times 4</td>
<td>424 778.5</td>
<td>9.02</td>
<td>391 479</td>
<td>0.47</td>
<td>394 059.1</td>
<td>1.14</td>
<td>393 176</td>
<td>0.91</td>
<td>395 860.5</td>
<td>1.60</td>
</tr>
</tbody>
</table>

**Notes:**
- Energy consumption costs are given in kWh.
- The values for DABC$^{2018}$, VNIC, and IGR$^{2020}$ are average values, while the values for ES$^{2021}$ and CRG$^{2017}$ are average RPI values.
- The RPI values are calculated as $100 \times \frac{ES_{\text{other}}}{ES_{\text{ours}}}$. If $ES_{\text{other}} > ES_{\text{ours}}$, the RPI is negative.

**References:**
5.7. Friedman test

Friedman test is a nonparametric test for the presence of significant differences in multiple overall distributions \[28\]. This test first assumes that multiple paired samples from multiple overall distributions are not significantly different. The hypothesis will be accepted when the value of \( p \) is not less than 0.05. If the hypothesis is rejected, the groups of samples are considered to be significantly different.

We analyze 90 examples with situations of \( t = 2 \) and \( t = 3 \). The results are shown in Tables 12–13. According to the Friedman test (confidence level \( \alpha = 0.05 \)), the \( p \) value is 0.000, which indicates that the compared algorithms are significantly different. By observing the results, the rank values of our proposed algorithm VNI are the smallest (1.00 and 1.01). VNI also had the smallest mean values (0.285 and 0.311) and the smallest minimum values (0.08 and 0.06). Meanwhile, the VNI obtains the best standard deviation values (0.1409 and 0.1569) and maximum values (0.79 and 0.84). From an overall perspective, we can deduce that the VNI is the most stable algorithm with CPU = 2 and CPU = 3. And the VNI demonstrates excellent performance in solving resource-balance DBFSP SDST problems.

Table 11 (continued).

<table>
<thead>
<tr>
<th>Job</th>
<th>m * f</th>
<th>DABC-2018</th>
<th>VNI</th>
<th>IGR-2020</th>
<th>ES-2022</th>
<th>CRO-2017</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>avg</td>
<td>RPI</td>
<td>avg</td>
<td>RPI</td>
<td>avg</td>
</tr>
<tr>
<td>10 × 5</td>
<td>1238 094</td>
<td>9.68</td>
<td>11 131 174</td>
<td>0.21</td>
<td>11 146 014</td>
<td>1.52</td>
</tr>
<tr>
<td>5 × 6</td>
<td>1716 316</td>
<td>11.37</td>
<td>14 574 478</td>
<td>0.24</td>
<td>15 887 46</td>
<td>1.15</td>
</tr>
<tr>
<td>8 × 6</td>
<td>792 043</td>
<td>9.05</td>
<td>725 047 4</td>
<td>0.37</td>
<td>731 225 4</td>
<td>1.23</td>
</tr>
<tr>
<td>10 × 6</td>
<td>374 784 4</td>
<td>9.18</td>
<td>343 837 7</td>
<td>0.16</td>
<td>347 231 1</td>
<td>1.15</td>
</tr>
<tr>
<td>5 × 7</td>
<td>407 196 8</td>
<td>13.63</td>
<td>359 277 7</td>
<td>0.26</td>
<td>363 750 3</td>
<td>1.50</td>
</tr>
<tr>
<td>8 × 7</td>
<td>577 463 9</td>
<td>8.94</td>
<td>531 036 4</td>
<td>0.18</td>
<td>535 927 2</td>
<td>1.11</td>
</tr>
<tr>
<td>10 × 7</td>
<td>777 184 7</td>
<td>9.82</td>
<td>709 367 9</td>
<td>0.23</td>
<td>718 191 4</td>
<td>1.57</td>
</tr>
<tr>
<td>Mean</td>
<td>829 474 3</td>
<td>10.0</td>
<td>758 424 9</td>
<td>0.2</td>
<td>765 193 3</td>
<td>1.1</td>
</tr>
</tbody>
</table>

Table 12 Results achieved by Friedman test (confidence level \( \alpha = 0.05 \)) when \( t = 2 \).

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Ranks</th>
<th>CN</th>
<th>Mean</th>
<th>Std. Deviation</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>VNI</td>
<td>1.00</td>
<td>90</td>
<td>0.285</td>
<td>0.1409</td>
<td>0.08</td>
<td>0.79</td>
</tr>
<tr>
<td>IGR</td>
<td>3.79</td>
<td>90</td>
<td>1.295</td>
<td>0.3357</td>
<td>0.71</td>
<td>2.40</td>
</tr>
<tr>
<td>DABC</td>
<td>5.00</td>
<td>90</td>
<td>10.135</td>
<td>1.1676</td>
<td>7.82</td>
<td>12.95</td>
</tr>
<tr>
<td>CRO</td>
<td>3.09</td>
<td>90</td>
<td>1.150</td>
<td>0.3889</td>
<td>0.61</td>
<td>2.37</td>
</tr>
<tr>
<td>ES</td>
<td>2.12</td>
<td>90</td>
<td>0.896</td>
<td>0.2327</td>
<td>0.35</td>
<td>1.66</td>
</tr>
</tbody>
</table>

Table 13 Results achieved by Friedman test (confidence level \( \alpha = 0.05 \)) when \( t = 3 \).

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Ranks</th>
<th>CN</th>
<th>Mean</th>
<th>Std. Deviation</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>VNI</td>
<td>1.01</td>
<td>90</td>
<td>0.311</td>
<td>0.1569</td>
<td>0.06</td>
<td>0.94</td>
</tr>
<tr>
<td>IGR</td>
<td>3.81</td>
<td>90</td>
<td>1.325</td>
<td>0.3733</td>
<td>0.75</td>
<td>2.71</td>
</tr>
<tr>
<td>DABC</td>
<td>5.00</td>
<td>90</td>
<td>10.008</td>
<td>1.3894</td>
<td>7.63</td>
<td>14.10</td>
</tr>
<tr>
<td>CRO</td>
<td>3.12</td>
<td>90</td>
<td>1.700</td>
<td>1.4348</td>
<td>0.34</td>
<td>5.92</td>
</tr>
<tr>
<td>ES</td>
<td>2.07</td>
<td>90</td>
<td>0.915</td>
<td>0.2802</td>
<td>0.55</td>
<td>1.87</td>
</tr>
</tbody>
</table>

500

![Fig. 7. Confidence intervals for the compared algorithms.](image-url)
6. Conclusions

Most existing algorithms in the literature address the discrete flowshop scheduling problem without considering the blocking constraint. Thus, in this study, we formulate the mathematic model of DBFSP_SDST with balanced energy cost and propose a local search with the learning_based variable neighborhood search strategy that seeds the initial solution using MME_en based on the MM strategy and NEH2_en. In the improved variable neighborhood search strategy (IVNS), three different local search methods are used to perturb the job sequence. In the experimental part, the effectiveness of the proposed strategies is verified. The performance of the proposed algorithm is empirically evaluated on 90 instances of DBFSP_SDST. The proposed MME_en and NEH2_en are compared, and the effectiveness of the learning-based variable neighborhood strategy is verified. In addition, comparison results between the proposed VNIG algorithm and the existing four compared algorithms are reported. Through the above simulation experiments, the proposed VNIG shows superior performance compared with state-of-the-art algorithms.

There are several opportunities for future research on DBFSP_SDST. First, the three local search strategies are randomly selected in this study. It might be desirable to develop a self-adaptive mechanism to select the local search from them to improve the exploration capability of the algorithm. Second, some local search strategies can be developed to further reduce the computational complexity of the algorithm. Third, uncertainties related to machine breakdowns, wrong operations, and changes in due date should also be considered when tackling DBFSP_SDST. Last, to remedy the current unsatisfactory situation regarding the experiment replication and comparison, the termination criterion of an algorithm should be appropriately designed, such as replacing the maximal elapsed CPU time with the number of fitness evaluations, to fulfill the comparisons among heuristics algorithms for the flowshop scheduling problem.

CRediT authorship contribution statement

Xue Han: Conception or design of the work, Acquisition, Analysis, Interpretation of data, Writing – original draft, Writing – review & editing. Yuyan Han: Conception or design of the work, Acquisition, Analysis, Interpretation of data, Writing – original draft, Writing – review & editing. Biao Zhang: Writing – review & editing, Interpretation of data. Haoxiang Qin: Writing – review & editing, Interpretation of data. Junqing Li: Writing – review & editing, Interpretation of data. Yiping Liu: Writing – review & editing, Interpretation of data. Dunwei Gong: Writing – review & editing, Interpretation of data.
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.

Data availability

The authors do not have permission to share data.

Acknowledgments

This work was jointly supported by the National Natural Science Foundation of China under grant numbers 61803192, 62106073, 61973203, 61966012, 61773246, and 71533001. We are grateful for Guangyue Youth Scholar Innovation Talent Program support received from LiaoCheng University, the Youth Innovation Talent Introduction and Education Program support received from Shandong Province Colleges and Universities, the Natural Science Foundation of Hunan Province of China under grant number 2021JJ40116, and the Natural Science Foundation of Shandong Province under grant numbers ZR2021QE195 and ZR2021QY105. Xue Han has approved the final version to be published and agrees to be accountable for all aspects of the work in ensuring that questions related to the accuracy or integrity of any part of the work are appropriately investigated and resolved.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.asoc.2022.109502.

References


