mathematics

MDPI

Article

A Distributed Blocking Flowshop Scheduling with Setup Times Using Multi-Factory Collaboration Iterated Greedy Algorithm

Chenyao Zhang [1], Yuyan Han [1,*], Yuting Wang [1,*], Junqing Li [2] and Kaizhou Gao [3]

[1] School of Computer Science, Liaocheng University, Liaocheng 252059, China
[2] School of Computer Science, Shandong Normal University, Jinan 252000, China
[3] Macau Institute of Systems Engineering, Macau University of Science and Technology, Macau 999078, China
* Correspondence: hanyuyan@lcu-cs.com (Y.H.); wangyuting@lcu-cs.com (Y.W.); Tel.: +86-188-6497-4734 (Y.H.); +86-156-6635-1136 (Y.W.)

Abstract: As multi-factory production models are more widespread in modern manufacturing systems, a distributed blocking flowshop scheduling problem (DBFSP) is studied in which no buffer between adjacent machines and setup time constraints are considered. To address the above problem, a mixed integer linear programming (MILP) model is first constructed, and its correctness is verified. Then, an iterated greedy-algorithm-blending multi-factory collaboration mechanism (mIG) is presented to optimize the makespan criterion. In the mIG algorithm, a rapid evaluation method is designed to reduce the time complexity, and two different iterative processes are selected by a certain probability. In addition, collaborative interactions between cross-factory and inner-factory are considered to further improve the exploitation and exploration of mIG. Finally, the 270 tests showed that the average makespan and RPI values of mIG are 1.93% and 78.35% better than the five comparison algorithms on average, respectively. Therefore, mIG is more suitable to solve the studied DBFSP_SDST.

Keywords: blocking; iterated greedy algorithm; distributed flowshop scheduling; multi-factory collaborative strategy; makespan

MSC: 93B28

Citation: Zhang, C.; Han, Y.; Wang, Y.; Li, J.; Gao, K. A Distributed Blocking Flowshop Scheduling with Setup Times Using Multi-Factory Collaboration Iterated Greedy Algorithm. *Mathematics* **2023**, *11*, 581. https://doi.org/10.3390/math11030581

Academic Editor: Ioannis G. Tsoulos

Received: 15 December 2022
Revised: 19 January 2023
Accepted: 19 January 2023
Published: 22 January 2023

1. Introduction

Industrial intellectualization and informatization are the frontier trends of manufacturing development. Manufacturing is the mainstay of the real economy and the lifeblood of the national economy, and its development is an essential reflection of a country's comprehensive national power. Smart manufacturing is the main research content of the manufacturing system at this stage. In the manufacturing industry, the flowshop scheduling problem (FSP) has been a popular topic of research and is of great practical importance. In FSP, jobs are processed on a series of machines according to a fixed process flow. The ultimate goal is to find the optimal scheduling sequence with optimal value(s) of the single (multiple) objective function(s). As we all know, in the context of globalization, the collaborative production mode between companies is becoming more and more common. The traditional centralized production methods are no longer able to meet market demands. Thus, the centralized manufacturing model is gradually shifted to a distributed manufacturing model [1], which can break geographical restrictions and make full use of the resources of multiple enterprises or factories to achieve a rational allocation, optimal combination, and sharing of resources [2]. Due to the above advantages of the distributed model, researchers have applied the distribution constraint to FSP and proposed the distributed permutation flowshop scheduling problem (DPFSP).

Many works on the DPFSP have been done. Naderi and Ruiz first constructed a MILP model and adopted heuristic of construction, and a variable neighborhood descent

method to address this problem [3]. Liu and Gao presented a hybrid variable neighborhood search by combining with the electromagnetism mechanism to optimize the makespan criterion [4]. Since then, a number of constructive algorithms have emerged, i.e., the improved variable neighborhood descent (VND) algorithm [5], the taboo search (TS) algorithm [6], the estimation distribution algorithm (EDA) [7], the scatter search (SS) algorithms [8], and the bounded search iterated greedy (BSIG) algorithm [9]. In addition, Komaki and Malakooti [10] presented a variable neighborhood search (VNS) to solve the DPFSP with a no-wait constraint. In recent years, new scheduling algorithms, i.e., two stage iterated greedy algorithms containing different local search operators [11] and a cooperative co-evolutionary algorithm (CCEA) [12] have been developed to optimize DPFSP and successfully applied to the distributed robotic scheduling problem [13]. To optimize the total flowtime value of DPFSP, Fernandez-Viagas et al. discussed some properties of DPFSP and proposed eighteen construction heuristics to obtain a solution with high quality [14].

Recently, researchers have also taken sequence-dependent setup times (SDSTs) into account in DPFSP, called DPFSP_SDST, and have done some work on DPFSP_SDST. To address this problem, an IG with restart strategy (IGR) is presented [15]. The experimental results have demonstrated that IGR has the best performance among all the compared algorithms, i.e., chemical reaction optimization, differential evolution, evolutionary algorithm, etc. Han et al. designed an iterated greedy (NIG) algorithm that includes swapping of single jobs and job blocks [16]. Furthermore, it shows better performance compared to advanced algorithms. Li et al. also extended the DPFSP by considering a heterogeneous machine with unrelated parallel (forming DHHFSP_SDST) [17]. Next, to further design a good algorithm, the three heuristics based on problem specifics and a discrete artificial bee colony (DABC) algorithm were employed to solve DPFSP_SDST [18]. The study in [19] proposed two mathematical models of DPFSP_SDST, i.e., constraint planning (CP) and MILP. The authors also presented an evolution strategy algorithm based on a self-adaptive mechanism to quickly provide the quality of solutions. In [20], the authors considered DPFSP_SDST with assembly constraints and presented a hyper-heuristic algorithm based on genetic programming.

The above scheduling problems assume that the buffers are infinite between any adjacent machines. However, due to cost constraints, temporary buffers may not be allowed between any adjacent machines. The current machine must be blocked with a job until the next machine is free. In this case, FSP with no buffer is transformed into a blocking flowshop scheduling problem (BFSP). Thus, our article simultaneously considers the above blocking, SDST and distributed constraints, and forms a distributed flowshop scheduling problem based on blocking and sequence-dependent setup times (DBFSP_SDST).

DPFSP with more than two machines has been evidenced in the literature [12] as an NP-hard problem. However, DBFSP_SDST, as an extension of DPFSP, adds blocking and sequence-dependent setup time constraints that are more complex than the permutation flowshop scheduling problem (PFSP). This is because, (1) from a distributed perspective, PFSP is a single-factory problem. One issue that needs to be solved is how to generate the optimal scheduling sequence. However, when it comes to DBFSP_SDST, the following two sub-issues must be taken into account. One is to assign the job to factories in a reasonable way, and the other is to arrange the job sequence for each factory [21,22]. (2) The DBFSP_SDST simultaneously considers blocking and SDST constraints in a distributed manufacturing environment in addition to the constraints listed in PFSP.

Regarding DBFSP, Companys and Ribas initially studied this problem and presented ten constructive heuristics based on typical heuristic rules [21]. Ying and Li constructed a MILP model of DBFSP and developed different hybrid IG algorithms [23]. Zhang et al. designed a discrete differential evolution (DDE) method based on problem features to optimize two different mathematical models [24]. Shao et al. employed a fruit fly optimization algorithm incorporating constructive heuristic initialization and an enhanced local search strategy [25]. Next, a mutation strategy combining crossover and insertion operators is employed to obtain a good solution [26]. Recently, Han et al. considered SDST and blocking

constraints in DPFSP and developed a variable IG (VNIG) algorithm to optimize energy cost [27].

In the present study, the iterated greedy algorithm (IGA) and its modifications have been successfully applied in many discrete scheduling problems. Ruiz and Stützle proposed IGA to address FSP with the makespan criterion for the first time [28]. Next, Lin et al. modified the IGA by improving initialization, local search, and destruction and reconstruction strategies to optimize DPFSP [29]. Pan and Ruiz proposed an effective IG to solve the mixed no-idle FSP [30]. The study in [31] presented an IG based on a reference (IRG) algorithm to effectively solve no-idle DFSP. Huang et al. designed an enhanced IGA to optimize the assembly DPFSP with total flowtime [32]. Mao et al. presented a multi-stage IGA to address DPFSP with a preventive maintenance constraint [33]. For the scheduling problems with the blocking constraint, IGA also shows superiority. Ribas et al. developed an efficient IGA for optimizing the blocking parallel flowshop scheduling problem with a total tardiness criterion [34]. Qin et al. considered an IG algorithm based on double-level mutation (IGDLM) in solving a hybrid BFSP [35]. For the DBFSP with makespan and total flowtime criteria, Chen et al. used some constructive heuristics in the IGA [36] and a population-based IG [21], respectively, to minimize the above two objectives. In addition, Öztop et al. employed four different IG algorithms for the hybrid flowshop scheduling problem to optimize the objective of total flowtime [37].

From the analysis above, it is found that (1) unlike other population-based algorithms, the iterated greedy algorithm (IGA) is an efficient meta-heuristic algorithm with a simple framework that can be coded and replicated easily. (2) Among the intelligence algorithms for DPFSP discussed above, the iterated greedy algorithm (IGA) exhibits advanced performance. The advantages of the IG algorithm are attributed to the simplicity of the algorithm framework, with few parameters, ease of integration, and good reinforcement and local convergence performance. For the DBFSP_SDST, no relevant research has attempted to solve this problem by using improved IGA. Therefore, to make the IGA more appropriate for DBFSP_SDST, this article has made some adjustments according to the problem characteristics and designed a multi-factory collaborative iterated greedy algorithm.

Our main innovations are that (1) a MILP of DBFSP_SDST with makespan is constructed, and the Gurobi solver is adopted to verify the correctness of this model. (2) According to the characteristics of the problem, a new refresh acceleration calculation method based on job insertion is designed to speed up the calculation of the objective, thereby reducing the time complexity of the algorithm. (3) To enrich the diversity of solutions, *iterative process* I and *iterative process* II strategies are selected by a certain probability. (4) A collaborative strategy between cross-factory and inner-factory is presented.

The remaining parts are listed as follows. Section 2 formulates a MILP model of DBFSP-SDST. Section 3 states the specific details of the mIG algorithm. Experimental results and statistical analyses are performed in Section 4. Section 5 summarizes the research on the problem, algorithm, and future directions of research.

2. Problem-Specific Characteristics

The DBFSP_SDST considered in this article can be characterized as follows. Assume that $F(F > 2)$ identical factories exist. For each factory, J jobs have been processed on M machines. All factories should meet the restrictions in the MILP. The constraints are as follows: (1) A job has been processed continuously in only one factory. (2) Each job should be processed on one machine at a time according to the scheduled order. (3) Each machine can process only a job at a time. (4) No buffer exists between adjacent machines. The current machine must be blocked with a job until the next machine is free. (5) On each machine, the sequence-dependent setup time is taken into account. In addition, the first job on the machine needs to be set with an initial setup time. (6) Jobs cannot be interrupted during processing. Based on the above constraints, the optimization objective of DBFSP_SDST is makespan (unit: seconds).

2.1. Mathematical Model

Notations:

F Number of factories.

J Number of jobs.

M Number of machines in each factory.

j, j' Index of jobs, $j, j' \in \{0, 1, \cdots, J\}$, where 0 denotes a dummy job that starts and ends at each factory.

m Index of machines.

$p_{j,m}$ Processing time of job j on machine m.

$s_{j,j',m}$ Setup time of adjacent job j and job j' on machine m. $s_{0,j,m}$ is a predetermined value when j is the initial job on machine m.

h A positive large number.

Decision Variables:

$C_{j,m}$ Completion time of job j on machine m.

$D_{j,m}$ Departure time of job j on machine m.

$x_{j,j'}$ A decision variable using binary coding, 1 if job j' is a direct successor of job j, 0 otherwise.

Objective:

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

Constraints:

$$\sum_{j'=0, j' \neq j}^{J} x_{j,j'} = 1, \forall j \in \{1, 2, \cdots, J\} \tag{2}$$

$$\sum_{j=0, j \neq j'}^{J} x_{j,j'} = 1, \forall j' \in \{1, 2, \cdots, J\} \tag{3}$$

$$\sum_{j'=1}^{J} x_{0,j'} \leq F \tag{4}$$

$$\sum_{j=1}^{J} x_{j,0} \leq F \tag{5}$$

$$\sum_{j'=1}^{J} x_{0,j'} = \sum_{j=1}^{J} x_{j,0} \tag{6}$$

$$D_{j,m} \geq C_{j,m}, \forall j \in \{1, 2, \cdots, J\}, \forall m \in \{1, 2, \cdots, M\} \tag{7}$$

$$C_{j,m} - p_{j,m} = D_{j,m-1}, \forall j \in \{1, 2, \cdots, J\}, \forall m \in \{1, 2, \cdots, M\} \tag{8}$$

$$C_{j',m} - p_{j',m} \geq D_{j,m} + s_{j,j',m} + (x_{j,j'} - 1) \cdot h, \forall j, j' \in \{1, 2, \cdots, J\}, j \neq j', \forall m \in \{1, 2, \cdots, M\} \tag{9}$$

$$C_{j,m} - p_{j,m} \geq s_{0,j,m} + (x_{0,j} - 1) \cdot h, \forall j \in \{1, 2, \cdots, J\}, \forall m \{1, 2, \cdots, M\} \tag{10}$$

$$C_{max} \geq c_{j,M}, \forall j \in \{1, 2, \cdots, J\} \tag{11}$$

Equation (1) is the makespan objective. Constraints (2) and (3) ensure that each job in the scheduling sequence can only have one immediate predecessor and successor, respectively. Constraints (4) and (5) assure that the dummy job has an immediate successor and predecessor, respectively. The dummy job must have an equal number of immediate predecessors and successors, which is assured by Constraint (6). Each job on each machine must have a departure time that is equal to or more than its completion time, as required by Constraint (7). According to Constraint (8), the departure time of each job from the previous machine is equal to the time that started processing on the current machine. Constraint (9) is that the start time of job j' on machine m is larger than the sum of the departure time of job j on machine m and the setup time $s_{j,j',m}$. Constraint (10) considers the initial setup

time $s_{0,j,m}$ of the first job on machine m. Constraint (11) defines the makespan. No more than 2F dummy jobs are used for sequence-based variables. The job sequence within each factory starts and ends with dummy jobs.

2.2. Example Instance

The described problem is clearly reflected by considering the example having five jobs ($J = 5$), two machines ($M = 2$), and two factories ($F = 2$). Table 1 gives the processing times for the five jobs, and the SDSTs are shown in Table 2. Processing time and SDST are in seconds. One possible solution is denoted as: $x_{0,1} = 1, x_{1,4} = 1, x_{4,0} = 1, x_{0,5} = 1, x_{5,3} = 1, x_{3,2} = 1, x_{2,0} = 1$; the rest decision variables are equal to 0. The solution corresponds to a sequence {0, 1, 4, 0, 5, 3, 2, 0}, where the dummy job 0 divides it into two sequences {1, 4} and {5, 3, 2}. It means that factory 1 processes jobs 1 and 4, and factory 2 processes jobs 5, 3, and 2. The makespan is 57, and the scheduling Gantt chart as shown in Figure 1.

Table 1. Processing times $p_{j,m}$ of jobs.

$p_{j,m}$	J_1	J_2	J_3	J_4	J_5
M_1	11	3	11	12	9
M_2	25	3	13	5	17

Table 2. The SDSTs $s_{j,j',1}$ and $s_{j,j',2}$ of jobs.

$s_{j,j',1}$	J_1	J_2	J_3	J_4	J_5	$s_{j,j',2}$	J_1	J_2	J_3	J_4	J_5
	7	14	6	21	5		24	1	22	12	10
J_1	-	11	16	10	20	J_1	-	13	18	3	20
J_2	12	-	12	9	23	J_2	8	-	20	19	1
J_3	0	5	-	23	16	J_3	16	3	-	18	23
J_4	4	3	11	-	0	J_4	20	22	15	-	17
J_5	15	23	6	2	-	J_5	9	13	7	5	-

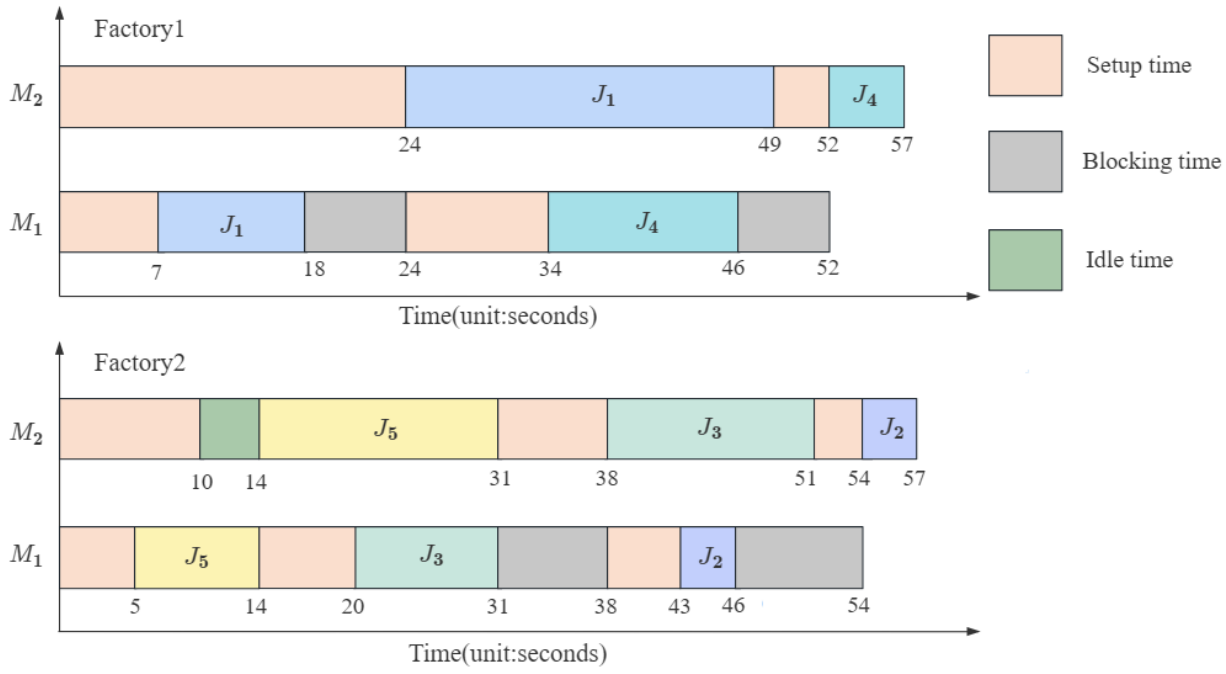


Figure 1. The Gantt chart of the example instance.

2.3. Improved Rapid Evaluation Criteria

A type of acceleration method inspired by Taillard [38] was proposed to save computational effort by combining the characteristics of the problem under study. In the rapid evaluation process, forward and backward calculation methods are adopted. The forward calculation is as follows. (1) Compute the leave time of the first job on the first machine, the second machine, and up to the last machine, respectively. (2) Similarly, the departure times on each machine for the second job, the third job, and until the last job are calculated. See Figure 2a. The backward calculation is as follows. (1) Calculate the departure time of the last job on the last machine, on the penultimate machine, and up to the first machine, respectively. (2) Similarly, the departure times on each machine for the penultimate job, the antepenultimate job, and up to the first job are calculated. See Figure 2b.

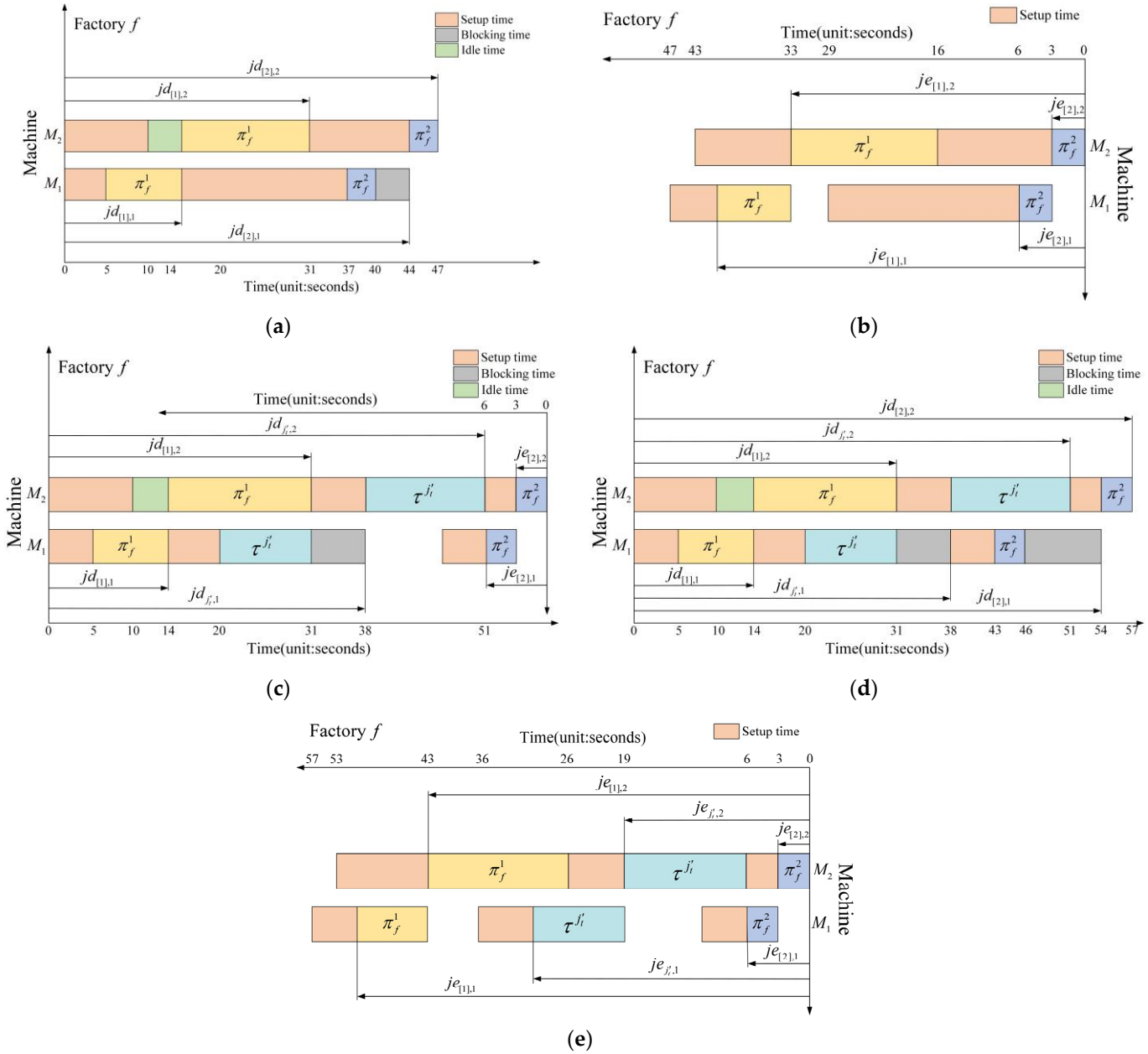


Figure 2. Rapid evaluation criteria. (**a**) Calculate the time $jd_{[j],m}$. (**b**) Calculate the time $je_{[j],m}$. (**c**) Insert job $\tau^{j'_t}$ into position 2. (**d**) Recalculate $jd_{[j],m}$ of the job after position 2. (**e**) Recalculate $je_{[j],m}$ of the job before position 2 and calculate $je_{j'_t,m}$.

Assume that the factory f consists of η_f jobs processed according to the sequence $\pi_f = \left\{\pi_f^1, \pi_f^2, \cdots, \pi_f^j, \cdots \pi_f^{\eta_f}\right\}$, where the job in the factory f is represented as π_f^j $j \in \left\{1, 2, \cdots, \eta_f\right\}$. $[j]$ denotes the index of the jth job. In the forward calculation, $jc_{[j],m}$ and $jd_{[j],m}$ denote the completion time and the leave time, respectively, of π_f^j on m. In the backward calculation, $js_{[j],m}$ and $je_{[j],m}$ denote the completion time and leave time, respectively, of π_f^j on m.

Refresh accelerated calculation for inserting job:

An attempt is made to insert η_s jobs $\tau^{j'_1}, \tau^{j'_2}, \cdots, \tau^{j'_t}, \cdots, \tau^{j'_{\eta_s}}$, $j'_t \in \{1, 2, \cdots, \eta_s\}$ sequentially into the job sequence π_f to minimize the makespan of the factory f.

Step 1: Set $t = 1$ and consider the insertion of the job $\tau^{j'_t}$.

Step 2: Forward calculate $jd_{[j],m}$ for job π_f^j on machine m according to Equations (12)–(14). Please see Figure 2a.

$$jc_{[j],0} = 0, j = 1, 2, \cdots, \eta_f \tag{12}$$

$$jc_{[j],m} = \begin{cases} max(s_{0,[j],m}, jc_{[j],m-1}) + p_{[j],m}, j = 1, m = 1, 2, \cdots, M \\ max(jd_{[j-1],m} + s_{[j-1],[j],m}, jc_{[j],m-1}) + p_{[j],m}, j = 2, 3, \cdots, \eta_f, m = 1, 2, \cdots, M \end{cases} \tag{13}$$

$$jd_{[j],m} = \begin{cases} jc_{[j],m+1} - p_{[j],m+1}, j = 1, 2, \cdots, \eta_f, m = 1, 2, \cdots, M-1 \\ jc_{[j],m}, j = 1, 2, \cdots, \eta_f, m = M \end{cases} \tag{14}$$

Step 3: Backward calculate $je_{[j],m}$ for job π_f^j on machine m according to Equations (15)–(17). Please see Figure 2b.

$$js_{[j],M+1} = 0, j = \eta_f, \eta_f - 1, \cdots, 1 \tag{15}$$

$$js_{[j],m} = \begin{cases} js_{[j],m+1} + p_{[j],m}, j = \eta_f, m = M, M-1, \cdots, 1 \\ max\left(je_{[j+1],m} + s_{[j],[j+1]}, js_{[j],m+1}\right) + p_{[j],m}, j = \eta_f - 1, \eta_f - 2, \cdots, 1, m = M, M-1, \cdots, 1 \end{cases} \tag{16}$$

$$je_{[j],m} = \begin{cases} js_{[j],m-1} - p_{[j],m-1}, j = \eta_f, \eta_f - 1, \cdots, 1, m = M, M-1, \cdots, 2 \\ js_{[j],m}, j = \eta_f, \eta_f - 1, \cdots, 1, m = 1 \end{cases} \tag{17}$$

Step 4: The job sequence π_f has a set of $\eta_f + 1$ positions. The job can be tested in these positions. Suppose that the qth position is inserted by job $\tau^{j'_t}$, where $q = 1, 2, \ldots, \eta_f + 1$. Then, $jd_{j'_t,m}$ can be calculated by using Equations (18) and (19), as shown in Figure 2c.

$$jd_{j'_t,0} = 0 \tag{18}$$

$$jd_{j'_t,m} = \begin{cases} max\left(s_{0,j'_t,m}, jd_{j'_t,m-1}\right) + p_{j'_t,m}, q = 1, m = 1, 2, \cdots, M \\ max\left(jd_{[q-1],m} + s_{[q-1],j'_t,m}, jd_{j'_t,m-1}\right) + p_{j'_t,m}, q = 2, \cdots, \eta_f + 1, m = 1, 2, \cdots, M \end{cases} \tag{19}$$

Step 5: From Equation (20), the makespan of factory f, $C_{\max}(j'_t, q)$, can be calculated after inserting job $\tau^{j'_t}$ into the qth position of job sequence π_f, as shown in Figure 2c.

$$C_{max}(j'_t, q) = \begin{cases} \underset{m=1,2,\cdots,M}{max}\left(jd_{j'_t,m} + s_{j'_t,[q],m} + je_{[q],m}\right), q = 1, 2, \ldots, \eta_f \\ jd_{j'_t,M}, q = \eta_f + 1 \end{cases} \tag{20}$$

Step 6: Repeat steps 3 and 4 until all positions have been considered. It is assumed that position q^{best} is the best position at which job $\tau^{j'_t}$ can be inserted.

Step 7: After job $\tau^{j'_t}$ is inserted into position q^{best}, the $jd_{[j],m}$ ($je_{[j],m}$) of the job before (after) position q^{best} is unchanged. Therefore, we only need to recalculate $jd_{[j],m}$ of the job after position q^{best}, according to Equations (12)–(14), and $je_{[j],m}$ of the job before position q^{best}, according to Equations (15)–(17). It is also necessary to calculate $je_{j'_t,m}$ of job $\tau^{j'_t}$, as shown in Figure 2d,e.

Step 8: Set $t = t + 1$, $\eta_f = \eta_f + 1$.

Step 9: Repeat steps 4, 5, 6, 7, and 8 until all η_s jobs have been considered.

With the above steps, we find that the computational complexity of inserting the jobs into the sequence is reduced from $O\left(m\left(\eta_s\eta_f^2 + \sum_{t=1}^{\eta_s}\left(2t\eta_f + t^2\right)\right)\right) \approx O(mn^2)$ to $O\left(m\left((2\eta_s+1)\eta_f + \sum_{t=1}^{\eta_s}(2t-1)\right)\right) \approx O(mn)$. The computational cost savings are substantial when dealing with large-scale problems.

3. Proposed IG Algorithm for DBFSP_SDST

First, unlike other population-based algorithms, the iterated greedy algorithm focuses on the iteration of one solution and has a strong local search capability due to its greedy strategy. It has the advantages that it is a simple framework, has a small number of parameters, and is easy to encode and replicate. Considering the multi-factory feature of DBFSP_SDST and the diversity of solutions from a global perspective, we make some modifications to the IGA, such as designing iterative processes I and II to increase the diversity of solutions and focusing on the cooperation between global search and local search. Thus, we propose a multi-factory collaborative iterated greedy algorithm, i.e., mIG to solve DBFSP_SDST.

3.1. Algorithm Description

Figure 3 shows the flow chart of mIG. It is well-known that an initialization solution with high quality can enhance the convergence of the algorithm. Thus, we first design an enhanced NEH heuristic, $Refresh\_NEH\_en$, to initialize the solution by using refresh accelerated calculation (see line 1 of Algorithm 1). Then, we adopt a multi-neighborhood structures search based on the variable neighborhood descent ($mVND$) method to improve the quality of the initialization solution described above (see line 2 of Algorithm 1). Considering the multiple factories characteristic of DBFSP_SDST and enhancing the diversity of solutions from a global perspective, we also design two iterative stages, called *iterative process* I and *iterative process* II, and each iterative process is adopted with a certain probability (see lines 4–8 of Algorithm 1). After performing the above search strategy, a simulated annealing acceptance criterion is adopted to enhance the diversity of solutions. If the performance of the current new solution is not better than the original one, the original one is still retained using the following criterion, $r \leq \exp\{-(C_{\max}(\pi^{current}) - C_{\max}(\pi^{origin}))/T\}$, $T = \lambda T, \lambda \in (0,1), r \in (0,1)$. Furthermore, the proposed refresh accelerated calculation for inserting job method is adopted throughout the algorithm.

3.2. Solution Representation

Regarding the solution encoding of DBFSP_SDST, a solution is represented by adopting a discrete integer encoding. That is, a solution π can be expressed, $\left\{\pi_1, \pi_2, \cdots, \pi_f, \cdots, \pi_F\right\}$, with each π_f consisting of $\left\{\pi_f^1, \pi_f^2, \cdots, \pi_f^j, \cdots \pi_f^{\eta_f}\right\}$, in which π_f refers to the job sequence of factory f, and η_f refers to the number of jobs in factory f. The specific example can be found in Section 2, in which a solution can be expressed as $\pi = \{\pi_1, \pi_2\}$, where $\pi_1 = \{1, 4\}$, $\pi_2 = \{5, 3, 2\}$, $\eta_1 = 2$, and $\eta_2 = 3$. This means that factory 1 processes jobs 1 and 4 in the order $1 \rightarrow 4$. Similarly, factory 2 processes jobs 2, 3, and 5 in the order $5 \rightarrow 3 \rightarrow 2$.

Algorithm 1: The proposed mIG

Input: ρ is the probability value.
01: $\pi = Refresh\_NEH\_en$
02: $\pi^{temp} = mVND(\pi)$
03: **while** (the current CPU time <terminate time) **do**
04: **if** $rand(0,1) > \rho$
05: $\pi = iterative\ process\ \mathrm{I}(\pi^{temp})$
06: **else**
07: $\pi = iterative\ process\ \mathrm{II}(\pi^{temp})$
08: **end if**
09: $\pi^{best} = AcceptanceCriterion(\pi)$
10: **end while**
Output: $BestSolution$

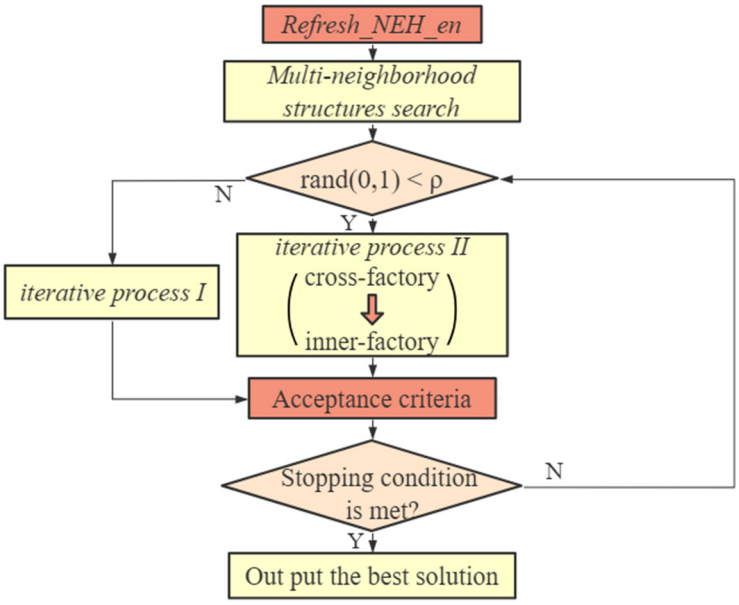


Figure 3. Flow chart of the mIG algorithm.

3.3. Initialization Solution

As mentioned above, an initialization sequence is closely related to the convergence nature of the algorithm. Thus, initialization operations are performed by using the heuristic method. According to the distributed characteristic of DBFSP_SDST, two issues need to be addressed. One is the assignment of jobs into factories, and the other is the arrangement of a reasonable scheduling sequence for each factory. NEH2_en presented by [11] has shown superior performance when optimizing a distributed flowshop scheduling problem and can solve the above two issues. However, NEH2_en has high time complexity due to the objective function needing to be reevaluated when jobs are put into all possible positions of all factories. Considering the problem characteristics and rapid evaluation method of the insertion job designed in Section 2.3, we propose a rapid initialization strategy *Refresh_NEH_en* by using refresh accelerated calculation.

Algorithm 2 shows the procedure of *Refresh_NEH_en* in detail. First, the total processing time P_j is calculated for every job on all machines (see line 1), and a scheduling sequence τ is obtained according to the descending of P_j (see line 2). Second, we take each job from the sequence τ and put it into each factory one by one (see lines 3–5), which ensures uniform allocation. The remaining jobs are removed one after another and put into all positions in all factories; finally the best position is selected (see lines 6–12). After finishing the insertion operator, we remove a job at position $pos_{f*} - 1$ or $pos_{f*} + 1$ from

π_{f*}, attempt it at all positions in π_{f*}, and select the position with minimal makespan (see lines 13–16).

Algorithm 2: *Refresh_NEH_en*

Input: an initial solution $\pi = \phi$.
01: $P_j = \sum_{m=1}^{M} p_{j,m}, j = 1, 2, \cdots, J$
02: $\tau = \{\tau^1, \tau^2, \cdots, \tau^J\}$ (Sort jobs according to decreasing P_j)
03: **for** $j = 1$ **to** F **do** %% uniformly allocate the jobs to the factories
04: Take job τ^j from the set of jobs and assign it in π_j
05: **end for**
06: **for** $j = F + 1$ **to** J **do**
07: **for** $f = 1$ **to** F **do**
08: Insert τ^j in all positions in π_f and calculate the corresponding makespan by using refresh accelerated calculation
09: $C_f = \min_{pos_f=1}^{\eta_f+1} C_f^{pos_f}$ and $pos_f = \arg(\min_{pos_f=1}^{\eta_f+1} C_f^{pos_f})$
10: **end for**
11: $pos_{f*} = arg(min_{f=1}^{F} C_f)$ %%pos_{f*} is the best position of factory with minimal makespan
12: Insert τ^j into position pos_{f^*} of π_{f^*}
13: Randomly select a job j' from $pos_{f^*} - 1$ or $pos_{f^*} + 1$ of π_{f^*}
14: Measure job j' in all positions using refresh accelerated calculation
15: Insert job j' in the position with minimum makespan
16: **end for**
Output: the initial solution π

3.4. Multi-Neighborhood Structures Search

According to the distributed characteristic of DBFSP_SDST, a variable neighborhood descent based on the multiple neighborhood structures search ($mVND$) method is adopted to further disturb the current solution. Multiple parallel isomorphic factories exist in DBFSP_SDST; we consider using cross-factory and inner-factory neighborhood search to explore the global solution. In addition, a critical factory that is the one with maximum makespan decides the final makespan value of DBFSP_SDST. In view of this, two neighborhood structures search operators based on a critical factory and non-critical factory, i.e., $Critical\_cross\_swap1(\pi)$ and $Critical\_inner\_insert(\pi_{f_{critical}})$, are designed.

$Critical\_cross\_swap1(\pi)$ accomplishes the interaction between the critical factory and secondary factory, called a cross-factory interaction, where the secondary factory is the one with the second highest makespan. The details are as follows. First, a critical factory is found (if there are more than one critical factory, one will be chosen randomly) according to the current solution π. Second, a job is chosen from the critical factory. Third, another job is selected from the secondary factory. Next, the above two selected jobs are swapped and evaluated. If the objective value of the critical factory is reduced, the current solution will be updated.

$Critical\_inner\_insert(\pi_{f_{critical}})$ accomplishes the interaction within the critical factory, called inner-factory interaction. First, select a random job in the critical factory. Second, try the selected job in all positions of $\pi_{f_{critical}}$, and select the best position.

Algorithm 3 gives the pseudocode of the *multi-neighborhood structures search* algorithm.

3.5. Two Iterative Processes

As mentioned above, IGA is an efficient meta-heuristic algorithm with a simple framework. Because its structure is easy to reproduce, many good strategies can be ported to its framework to further improve the performance of IGA. In addition, considering the multiple factories characteristic of DBFSP_SDST and enhancing the diversity of solutions from a global perspective, two iterative processes are designed, called *iterative process* I and *iterative process* II, and each iterative process is adopted by a certain probability.

Algorithm 3: $mVND(\pi)$

Input: π is the initial solution.
01: Find a critical factory $f_{critical}$ and secondary factory $f_{secondary}$ and record their scheduling sequences $\pi_{f_{critical}}$ and $\pi_{f_{secondary}}$, respectively.
02: $p_{max} = 2$ and $p = 1$
03: **do** {
04: **if** $p = 1$
05: $\pi^{temp} = Critical\_cross\_swap1(\pi)$
06: **else**
07: $\pi^{temp} = Critical\_inner\_insert(\pi_{f_{critical}})$
08: **end if**
09: **if** $C_{\max}$ is improved
10: $\pi = \pi^{temp}$
11: $p = 1$
12: **else**
13: $p = p + 1$
14: **end if**
15: } **while**$(p \leq p_{max})$
16: **end while**
Output: π

The *iterative process* I (see Algorithm 4) adopts $vDestruction\_Reconstruction(\pi)$ (see Algorithm 5) and $Critical\_cross\_swap1(\pi)$ (see Section 3.4) operators to disturb the current solution. The traditional destruction and reconstruction operators [11] are improved according to the distributed characteristics, abbreviated as $vDestruction\_Reconstruction(\pi)$. The details are as follows. First, initialize a parameter, d, using the random function $randbetween(2, 6)$ and use it to generate an integer between 2 and 6. Second, a sequence π_R with d jobs is obtained, in which $d/2$ jobs are extracted from the critical factory, and the rest are randomly selected from the non-critical factories (see lines 2–9). At the same time, the above d jobs are sequentially removed from the original sequence. Third, adopt the jump reconstruction operator [39] to insert d jobs in all possible positions and finally select the best position (see lines 13–21). It should be noted that (1) the difference between jump reconstruction and the traditional reconstruction is that the former adopts a jumpy insertion when the insertion cannot improve the quality of the solution, which can accelerate insertion speed and reduce the time complexity; (2) a refresh accelerated calculation is adopted when performing the above insertion operator and calculation function value. The proposed $vDestruction\_Reconstruction(\pi)$ can further explore the deep neighborhood of the solution, increasing the diversity of solutions to prevent falling into the local optimum. Algorithm 5 displays the procedure of $vDestruction\_Reconstruction(\pi)$ in detail.

The *iterative process* II (see Algorithm 6) accomplishes the interaction of cross-factory and inner-factory. Since the completion time of the critical factory directly affects the optimal solution of the whole scheduling, it is necessary to appropriately schedule the critical factory. Combined with the distributed characteristic of DBFSP_SDST, the cross-factory and inner-factory strategies are designed, respectively. In this way, the development and exploration of the proposed algorithm can be balanced by the cooperation of the two strategies.

Multiple search strategies can improve the diversity of solutions. Therefore, in the cross-factory strategy, four disturbing operators are designed, i.e., $vDestruction\_Reconstruction(\pi)$, $Critical\_cross\_swap1(\pi)$, $Critical\_min\_swap(\pi)$, and $Critical\_cross\_swap2(\pi)$, to improve the opportunity of obtaining potential solutions. To further increase the search efficiency of mIG, the above four strategies will be adaptively selected (see Algorithm 6). In the inner-factory strategy, an operator $Critical\_inner\_swap(\pi_{f_{critical}})$ is proposed to optimize the sequence within the factory.

Algorithm 4: *iterative process* I

Input: π is the current primary solution; J is the total number of jobs in π.
01: Find a critical factory $f_{critical}$ and secondary factory $f_{secondary}$ and record their scheduling sequence $\pi_{f_{critical}}$ and $\pi_{f_{secondary}}$, respectively.
02: $\pi = vDestruction\_Reconstruction(\pi)$ %% Algorithm 5
03: **for** $cnt = 1$ **to** $J/2$ **do**
04: $\pi^{temp} = Critical\_cross\_swap1(\pi)$ %% subSection 3.4
05: **if** C_{max} is improved
06: $\pi = \pi^{temp}$
07: **end if**
08: **end for**
Output: π

Algorithm 5: $vDestruction\_Reconstruction(\pi)$

Input: π is the current primary solution; d is the number of removed jobs from π, π_R=$\varnothing$
01: Find a critical factory $f_{critical}$ and record its scheduling sequence $\pi_{f_{critical}}$
/* Destruction */
02: $d = randbetween(2, 6)$
03: **for** $cnt = 1$ **to** $d/2$ **do**
04: Select a random job j from $\pi_{f_{critical}}$
05: $\pi_R \leftarrow j$ and $\pi_{f_{critical}} = \pi_{f_{critical}} \backslash j$
06: **end for**
07: **while** $|\pi_R| < d$ **do** %%$|\pi_R|$ refers to the number of jobs in π_R
08: Randomly select a job j from $\pi_f(\pi_f \neq \pi_{f_{critical}})$ %%π_f is the sequence of factory f
09: $\pi_R \leftarrow j$ and $\pi_f = \pi_f \backslash j$
10: **end while**
/* Reconstruction based on jumpy insertion and refresh accelerated calculation */
11: **for** $j = 1$ **to** d **do**
12: **for** $f = 1$ **to** F **do**
13: pos =0 and $K = 1$
14: **while** $pos \leq \left|\pi_f\right|$ **do**
15: Measure job j at position pos of π_f^{temp} using refresh accelerated calculation
16: **if** $C_{\max}$ is improved
17: Insert job j at pos of π_f^{temp}, and $K = 1$
18: **else**
19: $K = K + 1$
20: **end if**
21: $pos = pos + K$
22: **end while**
23: **end for**
24: **end for**
Output: π

Except for $Critical\_cross\_swap1(\pi)$ and $vDestruction\_Reconstruction(\pi)$, which are stated in Sections 3.4 and 3.5, respectively, $Critical\_cross\_swap2(\pi)$, $Critical\_min\_swap(\pi)$, and $Critical\_inner\_swap(\pi_{f_{critical}})$ are as follows.

$Critical\_min\_swap(\pi)$ is the interaction between the two factories with maximal and minimal makespan. First, select two jobs from each of the two factories mentioned above. Second, the above two selected jobs are swapped and evaluated. If the objective value of the critical factory is reduced, the current solution will be updated.

$Critical\_cross\_swap2(\pi)$ performs the $Critical\_cross\_swap1(\pi)$ operation twice to explore the space more deeply and facilitate the improvement of the quality of the solution.

$Critical\_inner\_swap(S_s)$: Select two jobs at random from the sequence of critical factory. Next, swap the two selected jobs. This will get a new solution and apply this swap

when a sequence of smaller makespan solutions is produced. If the objective value of the critical factory is reduced, the current solution will be updated.

In the self-adaptive strategy, two lists are defined, i.e., *List* and *BestList*. *List* contains sixty search strategies that are randomly selected from the above four strategies. *BestList* is initialized to empty. Each value of parameter R represents one of the four strategies, $R \in (1, 2, 3, 4)$. During the iteration, if the solution is improved, the corresponding strategy is saved to the *BestList*. Last, by using the strategies in the *BestList* to update *List* by parameter ω, ω determines how many strategies in *BestList* are available to update *List* (see lines 17–22). The details of *iterative process* II, including the self-adaptive strategy, are described in Algorithm 6.

Algorithm 6: *iterative process* II

Input: the current solution π, counter c, cnt, i
01: Find a critical factory $f_{critical}$ and secondary factory $f_{secondary}$ and a factory with minimal makespan f_{min}. Record their scheduling sequence $\pi_{f_{critical}}$, $\pi_{f_{secondary}}$, and $\pi_{f_{min}}$, respectively.
/* cross-factory */
02: **for** $c = 1$ **to** $|List|$ **do** %% $|List|$ is the length of *List*
03: $R = randbetween(1, 4)$
04: **switch** (R)
05: **case 1:** $\pi^{temp} = vDestruction\_Reconstruction(\pi)$ %% Section 3.5
06: break;
07: **case 2:** $\pi^{temp} = Critical\_min\_swap(\pi)$
08: break;
09: **case 3:** $\pi^{temp} = Critical\_cross\_swap1(\pi)$ %% Section 3.4
10: break;
11: **case 4:** $\pi^{temp} = Critical\_cross\_swap2(\pi)$
12: break;
13: **if** $C_{\max}$ is improved
14: $\pi = \pi^{temp}$
15: Record the R value in *BestList*
16: **end for**
17: **for** $i = 1$ **to** $\min\{\omega \times |List|, |BestList|\}$
18: $List[i]$=$BestList[i]$
19: **end for**
20: **for** $i = \min\{\omega \times |List|, |BestList|\} + 1$ **to** $|List|$ **do**
21: $List[i]$=$randbetween(1, 4)$
22: **end for**
/* inner-factory */
23: **for** $cnt = 1$ **to** $J/2$ **do**
24: $\pi^{temp} = Critical\_inner\_swap(\pi_{f_{critical}})$
25: **if** $C_{\max}$ is improved
26: $\pi = \pi^{temp}$
27: **end if**
28: **end for**
Output: π

3.6. The Computational Complexity of mIG

In mIG, we suppose that there are n jobs, f factories, and m machines. Each factory contains $\frac{n}{f}$ jobs. The computational complexity of the mIG algorithm includes initialization, multi-neighborhood structures search, *iterative process* I, and *iterative process* II. First, the time complexity of *Refresh_NEH_en* is $O\left(mn + n\log_2 n + f + (n-f)\left(mn + 2\frac{n}{f} + m\frac{n}{f}\right)\right) \approx O\left(n^2\right)$. Second, the time complexity of the multi-neighborhood structures search is calculated as $O\left(\frac{n}{f} + \frac{n}{f} \times \frac{n}{f} \times m\right) \approx O\left(n^2\right)$. In addition, assume that the number of iterations of *iterative process* I and *iterative process* II are k_1 and k_2, respectively. For *iterative process* I, the

complexity is $O\left(k_1 \times \frac{n}{2} \times m \times \frac{n}{f} \times 2\right) \approx O(n^2)$. For *iterative process* II, the complexity is $O\left(k_2 \times \frac{n}{2} \times m \times \frac{n}{f}\right) \approx O(n^2)$. In summary, the complexity of the whole mIG is $O(n^2)$.

4. Numerical Experiment and Analysis

This section gives the experimental design and analysis to demonstrate the effectiveness of mIG. The experiments are run on a PC with Intel(R) Core (TM) i7 CPU @ 2.90 GHz processor and 8 GB of RAM. For the proposed MILP model, the Gurobi 9.1.2 solver is adopted. For all the compared algorithms, C++ in the Visual Studio 2019 environment is used for coding and runs on the Release x64 platform. In the algorithm test, to ensure fairness, the maximum CPU elapsed time is adopted as the stopping criterion. In addition, it is considered that the algorithm has practical significance only when it can solve the problem in an acceptable time. Therefore, the termination condition is set as $TimeLimit = 5 \times J \times M$ milliseconds in this article. J and M indicate the total number of jobs and machines in the test instance, respectively. Each instance is run 5 times independently.

4.1. Test Data and Performance Metric

The experimental data used in this article can be referred to in [15]. This article test 270 instances with $F \times M \times J \times Factor$, where $F(F \in \{2,3,4,5,6,7\})$ is the number of factories, $M(M \in \{5,8,10\})$ is the number of machines, $J(J \in \{100,200,300,400,500\})$ is the number of jobs, and $Factor$ ($Factor \in \{25,50,100\}$) is the influence factor value that is used to generate different instances for the same scale size problem. From the above analysis, $6 \times 3 \times 5 \times 3 = 270$ combinations are obtained. Processing times for each job are evenly distributed within $[1,99)$. The setup times of each job relative to the other jobs are calculated by the equation $(1 + rand()\%99) \times Factor/100$, where $rand()$ used to generate a random integer.

We adopt the relative percentage increase (RPI) as an evaluation indicator. RPI estimates the difference between the makespan obtained by an algorithm and the optimal makespan found so far. The equation to calculate the RPI is shown below:

$$RPI = \frac{M_i - M_{best}}{M_{best}} \times 100\% \tag{21}$$

where M_{best} is the minimal makespan found by all compared algorithms of 5 independent running for a test instance. M_i refers to the average makespan obtained by the ith algorithm of 5 independent running for a test instance. i belongs to ES [40], DABC [18], IGR [15], EA [14], and DDE [24]. Because there are 3 different instances for each scale instance, the average PRI is calculated for 3 different instances, called ARPI. Obviously, the smaller the RPI or ARPI, the better result the algorithm obtained.

4.2. Correctness Verification of MILP

The correctness of the presented MILP model is verified by using 8 small-scale instances. The model is written in Python on the Gurobi solver. In the exact solver, the maximum termination criterion is set to 3600 s [41,42]. The termination criterion of mIG is set to $TimeLimit = 5 \times J \times M$. Set each instance to run 5 times independently to reduce the randomness of mIG. Table 3 lists the respective makespan and running time of MILP and mIG. Among them, the makespan represents the best value found in the termination time. In addition, F_J_M denotes the numbers of factories, jobs, and machines, respectively.

Table 3. Result for the MILP model.

F_J_M	MILP		mIG	
	Makespan	**Time (s)**	**Makespan**	**Time (s)**
2_2_2	**115**	0.00	**115**	0.02
2_5_2	**135**	0.02	**135**	0.05
2_8_2	**198**	0.14	**198**	0.08
2_10_2	**214**	2.43	**214**	0.10
2_12_2	**243**	23.04	**243**	0.12
2_20_2	**424**	3600	**424**	0.20
2_35_2	763	3600	**742**	0.35
2_40_2	879	3600	**844**	0.40

Best values are indicated in bold.

Table 3 shows that the optimal solution can be found by MILP and takes less time when the instance size is small, i.e., 2_2_2, 2_5_2, 2_8_2, 2_10_2, and 2_12_2 instances. Within the termination time, the values of the makespan obtained by Gurobi are good for 6 (6/8) instances, suggesting that the MILP is correct and can find optimal solutions in small-scale instances. As the scale of the instances continues to grow, i.e., 2_35_2 and 2_40_2 instances, MILP cannot generate a good solution even if the run time is extended to 3600 s. However, mIG can obtain the best solution in a shorter time for all instances. Thus, mIG has better capacity to solve large-scale and complicated instances of DBFSP SDST than MILP.

4.3. Parameter Calibration

In the proposed mIG, two key parameters should be calibrated. One is the threshold value of two iterative processes, ρ, and the other is the proportion of $|BestList|$ to $|List|$, ω. To obtain a more intuitive sensitivity of the two parameters, the Taguchi method of design of experiment (DOE) is used to determine the best combination of parameter values. For each parameter, the four levels illustrated in Table 4 are considered, and 16 ($4 \times 4 = 16$) parameter combinations are listed in Table 5. To fairly investigate the sensitivity of these two parameters, three different instances are randomly selected (*F_J_M*), i.e., 2_100_5, 4_300_5, 7_500_10. For each instance, 16 combinations are run independently five times and obtain the average RPI values (see Table 5). Factor-level trends for each parameter are shown in Figure 4. Table 6 indicates the level of significance of the two parameters. The largest influence on the algorithm is exerted by the parameter ρ, followed by ω.

Table 4. Parameter level factor.

Parameters	**Parameter Level**			
	1	**2**	**3**	**4**
ρ	0	0.1	0.2	0.3
ω	0.6	0.7	0.8	0.9

From Tables 4–6 and Figure 4, the parameter ρ has the greatest influence on the experimental results. It directly affects the global and local search balance of the two iterative processes. As can be seen in Figure 4, when $\rho = 0$, *iterative process* I is invoked completely; *iterative process* II is not involved. At this time, the value of ARPI (1.282) is higher, suggesting that the IGA with only the *iterative process* I strategy easily falls into a local optimum. However, when $\rho = 0.1$, the average RPI (1.221) is better than that of $\rho = 0.2$ and $\rho = 0.3$. This can further illustrate the validity of our proposed *iterative process* II to increase the diversity of solutions and avoid local optima.

For the parameter ω, it determines how many strategies in *BestList* are available to update *List*. If the value is too small, it suggests that few good search strategies in *BestList* are used to update *List*, which may influence the convergence of the algorithm.

On the contrary, if the value is too large, the diversity of strategies in *List* may be reduced. Thus, the performance of mIG is tested under the values of ω being 0.6, 0.7, 0.8, and 0.9, respectively. Based on the experimental results of Table 5 and Figure 4, the value of ω is set 0.7.

Table 5. Orthogonal array and ARPI value.

Experiment Number	Parameters		Response (ARPI)
	ρ	ω	
1	0	0.6	1.27
2	0	0.7	1.33
3	0	0.8	1.23
4	0	0.9	1.30
5	0.1	0.6	1.39
6	0.1	0.7	1.08
7	0.1	0.8	1.15
8	0.1	0.9	1.26
9	0.2	0.6	1.27
10	0.2	0.7	1.34
11	0.2	0.8	1.39
12	0.2	0.9	1.34
13	0.3	0.6	1.30
14	0.3	0.7	1.25
15	0.3	0.8	1.28
16	0.3	0.9	1.31

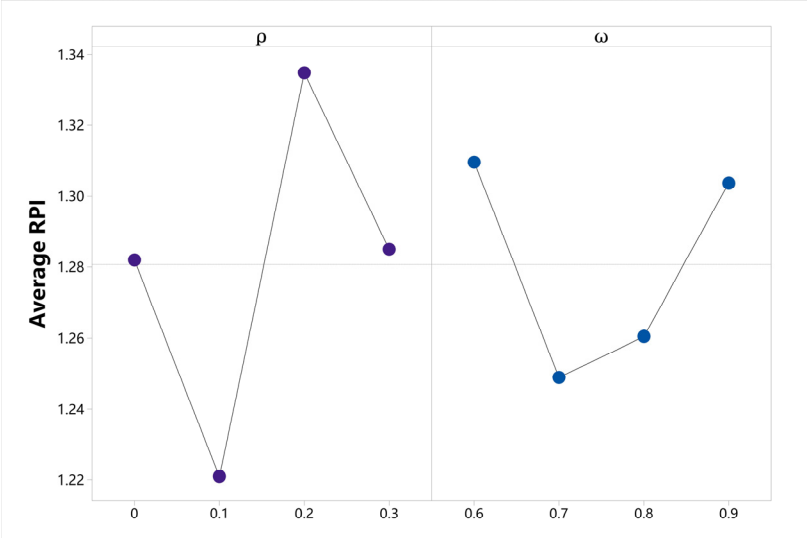


Figure 4. The trend of the parameter level.

Table 6. The average RPI response values.

Level	ρ	ω
1	1.282	1.310
2	1.221	1.249
3	1.335	1.260
4	1.285	1.304
Delta	**0.114**	0.061
Rank	**1**	2

Best values are indicated in bold.

4.4. Evaluation of the Proposed Problem-Specific mVND Operator

In this section, the proposed *mVND* strategy is investigated to demonstrate its contribution. mIG_NV refers to the mIG without *mVND*. All instances were tested in the same experimental environment, and each instance was repeatedly run 5 times, with $TimeLimit = 5 \times J \times M$ milliseconds as the same termination time. ANOVA will be used to evaluate the RPI values of all instances as experimental results. From the results shown in Figure 5, the value of RPI yielded by mIG with *mVND* is lower than that of mIG_NV. This suggests that the proposed multi-neighborhood structures search based on variable neighborhood descent can increase the diversity of mIG and provide more opportunities to generate potential solutions. In addition, the reason why *mVND* has good performance is due t the designed neighborhood search strategies for cross-factory and inner-factory, which provide advantages for exploring the global solution.

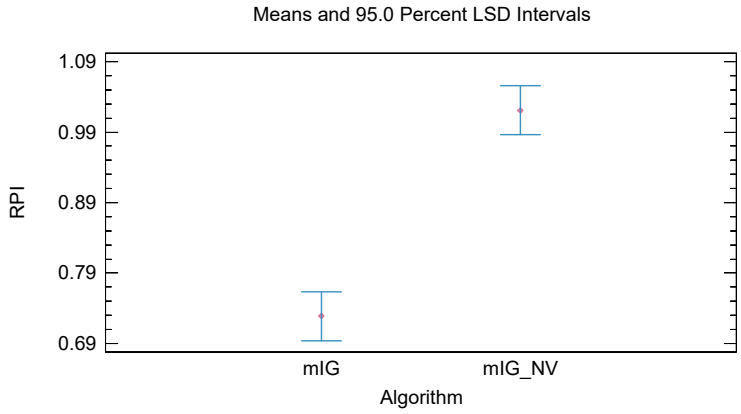


Figure 5. Confidence interval for mIG and mIG_NV.

4.5. Evaluation of mIG with Other Efficient Algorithms

This section compares the mIG algorithm with five intelligent optimization algorithms for solving DFSP, i.e., ES [40] and DDE [24], which are used to solving DBFSP, DABC [18], IGR [15], and EA [14], which are used to solving DPFSP. For fairness of comparison, all algorithms are carefully implemented according to the characteristics of the problem under the same termination conditions. The termination condition of all algorithms is set as $TimeLimit = 5 \times J \times M$ milliseconds. The mIG without refresh accelerated calculation, called mIG0, is also compared. In Table 7, *J_M* represents the scale with J jobs and M machines. In addition, we calculated the percentage values using equation $\left(P_{Comparing} - P_{mIG}\right) / P_{Cmparing} \times 100\%$, where $P_{Comparing}$ and P_{mIG} refer to the values of Avg or ARPI obtained by the comparing algorithm and mIG, respectively. The calculated percentages represent how much better mIG is than other algorithms, and the data are marked in bold. For Avg, in different size instances of $F = 2$, the percentages of mIG superior to EA, DDE, DABC, IGR, ES, and mIG0 are 1.11%, 1.64%, 4.25%, 2.04%, and 1.10%, respectively. Similarly, for $F = 3, 4, 5, 6, 7$, the percentages are better than the six comparing algorithms. For ARPI, the percentages of mIG superior to the comparison algorithms, EA, DDE, DABC, IGR, ES, and mIG0 are 66.67%, 76.47%, 86.37%, 80.56%, 70.98%, and 50.88%, respectively. It is obviously the case that when $F = 3, 4, 5, 6, 7$, the percentages of mIG are still better than other algorithms.

Table 7. Average makespan and RPI values of ES, DABC, IGR, EA, DDE, mIG0, and mIG.

Factory	*J_M*	Time (s)	Algorithms													
			EA		DDE		DABC		IGR		ES		mIG0		mIG	
			Avg	ARPI	Avg	ARPI	Avg	ARPI	Avg	ARPI	Avg	ARPI	Avg	ARPI	Avg	ARPI
$F = 2$	100_5	2.5	4464	1.80	4506	2.63	4478	2.05	4572	4.20	4537	3.44	4497	2.44	**4416**	**0.62**
	100_8	4	5030	2.01	5116	3.78	5060	2.62	5138	4.20	5113	3.68	5079	2.90	**4976**	**0.90**
	100_10	5	5170	1.45	5241	2.93	5184	1.77	5268	3.44	5255	3.12	5211	2.26	**5139**	**0.83**
	200_5	5	8775	1.83	8839	2.57	8833	2.54	8907	3.34	8797	2.12	8710	1.13	**8688**	**0.75**
	200_8	8	9726	1.34	9844	2.54	9799	2.11	9891	3.05	9761	1.69	9701	1.05	**9647**	**0.47**
	200_10	10	10,057	1.06	10,134	1.85	10,099	1.50	10,193	2.44	10,113	1.59	10,070	1.14	**10,003**	**0.45**
	300_5	7.5	13,106	1.99	13,182	2.59	13,388	4.14	13,221	2.87	13,054	1.59	12,939	0.66	**12,920**	**0.50**
	300_8	12	14,373	1.57	14,388	1.68	14,595	3.09	14,453	2.15	14,352	1.40	14,267	0.81	**14,245**	**0.61**
	300_10	15	14,929	1.80	15,005	2.36	15,154	3.28	15,070	2.79	14,905	1.58	14,781	0.77	**14,771**	**0.68**
	400_5	10	17,195	1.60	17,300	2.15	18,135	6.95	17,371	2.58	17,251	1.84	17,073	0.81	**17,041**	**0.61**
	400_8	16	18,864	1.77	18,935	2.15	19,483	5.03	18,989	2.45	18,766	1.20	18,665	0.67	**18,608**	**0.35**
	400_10	20	19,771	1.68	19,868	2.15	20,300	4.30	19,953	2.57	19,697	1.26	19,602	0.74	**19,525**	**0.35**
	500_5	12.5	21,294	2.12	21,374	2.52	23,006	10.13	21,435	2.82	21,293	2.09	20,995	0.66	**20,921**	**0.28**
	500_8	20	23,561	1.73	23,608	1.96	24,675	6.50	23,662	2.19	23,439	1.18	23,291	0.53	**23,270**	**0.44**
	500_10	25	24,559	1.42	24,660	1.84	25,590	5.61	24,733	2.12	24,505	1.14	24,349	0.52	**24,348**	**0.51**
	Mean	-	14,058	1.68	14,133	2.38	14,518	4.11	14,190	2.88	14,056	1.93	13,949	1.14	**13,901**	**0.56**
Percentage		-	1.11%	66.67%	1.64%	76.47%	4.25%	86.37%	2.04%	80.56%	1.10%	70.98%	0.03%	50.88%	-	-
$F = 3$	100_5	2.5	3072	3.24	3122	4.85	3060	2.82	3146	5.69	3117	4.68	3074	3.23	**3001**	**0.88**
	100_8	4	3425	2.19	3464	3.38	3419	2.08	3488	4.08	3464	3.40	3425	2.16	**3364**	**0.37**
	100_10	5	3630	2.54	3650	3.14	3619	2.26	3675	3.83	3675	3.81	3628	2.43	**3562**	**0.63**
	200_5	5	5852	2.21	5874	2.65	5854	2.18	5925	3.50	5875	2.55	5828	1.72	**5766**	**0.60**
	200_8	8	6539	2.44	6615	3.62	6523	2.14	6633	3.91	6548	2.58	6506	1.85	**6428**	**0.59**
	200_10	10	6883	1.91	6968	3.18	6879	1.83	6991	3.51	6927	2.52	6881	1.84	**6793**	**0.49**
	300_5	7.5	8793	2.19	8850	2.87	8876	3.04	8875	3.14	8778	1.95	8731	1.40	**8671**	**0.66**
	300_8	12	9734	1.66	9759	1.97	9776	2.07	9793	2.27	9697	1.26	9658	0.84	**9625**	**0.45**
	300_10	15	10,070	1.54	10,122	2.04	10,102	1.83	10,157	2.39	10,065	1.44	10,026	1.04	**9971**	**0.47**
	400_5	10	11,705	1.93	11,749	2.34	11,941	3.84	11,807	2.84	11,671	1.56	11,596	0.92	**11,529**	**0.30**
	400_8	16	12,812	1.91	12,832	2.07	12,985	3.25	12,884	2.47	12,761	1.46	12,700	1.00	**12,642**	**0.51**
	400_10	20	13,263	1.69	13,313	2.12	13,408	2.78	13,353	2.40	13,242	1.50	13,180	1.03	**13,104**	**0.40**
	500_5	12.5	14,443	2.11	14,467	2.27	14,983	5.81	14,518	2.62	14,353	1.48	14,284	1.00	**14,231**	**0.53**
	500_8	20	15,879	2.13	15,898	2.31	16,271	4.54	15,953	2.64	15,795	1.53	15,725	1.05	**15,649**	**0.53**
	500_10	25	16,482	1.52	16,541	1.94	16,818	3.49	16,588	2.21	16,442	1.17	16,391	0.89	**16,304**	**0.31**
	Mean	-	9505	2.08	9548	2.72	9634	2.93	9586	3.17	9494	2.19	9442	1.49	**9376**	**0.51**
Percentage		-	1.36%	75.48%	1.80%	81.25%	2.68%	82.59%	2.19%	83.91%	1.24%	76.71%	0.70%	65.77%	-	-

Table 7. *Cont.*

Factory	*J_M*	Time (s)	Algorithms													
			EA		DDE		DABC		IGR		ES		mIG0		mIG	
			Avg	ARPI	Avg	ARPI	Avg	ARPI	Avg	ARPI	Avg	ARPI	Avg	ARPI	Avg	ARPI
$F = 4$	100_5	2.5	2353	3.38	2387	4.98	2325	2.14	2397	5.38	2387	4.84	2337	2.67	**2301**	**1.09**
	100_8	4	2674	3.46	2702	4.57	2653	2.61	2696	4.32	2703	4.56	2648	2.47	**2607**	**0.87**
	100_10	5	2824	3.04	2851	4.11	2809	2.50	2860	4.35	2850	4.01	2797	2.05	**2758**	**0.66**
	200_5	5	4473	2.62	4473	2.60	4454	2.32	4514	3.52	4486	2.87	4462	2.26	**4389**	**0.57**
	200_8	8	5040	2.51	5092	3.51	5030	2.26	5093	3.55	5054	2.76	5021	2.08	**4953**	**0.66**
	200_10	10	5253	1.86	5290	2.52	5251	1.87	5332	3.34	5280	2.42	5244	1.73	**5183**	**0.53**
	300_5	7.5	6691	2.58	6713	2.91	6697	2.61	6732	3.17	6687	2.47	6651	1.90	**6574**	**0.64**
	300_8	12	7373	1.62	7407	2.16	7391	1.77	7429	2.44	7378	1.65	7356	1.28	**7299**	**0.44**
	300_10	15	7703	1.69	7752	2.35	7710	1.78	7773	2.62	7727	1.98	7690	1.49	**7615**	**0.45**
	400_5	10	8755	2.13	8769	2.29	8835	2.98	8794	2.57	8707	1.57	8672	1.12	**8609**	**0.33**
	400_8	16	9681	2.08	9720	2.49	9730	2.56	9760	2.90	9642	1.62	9635	1.52	**9556**	**0.65**
	400_10	20	10,162	1.84	10,201	2.24	10,210	2.27	10,227	2.51	10,142	1.58	10,133	1.52	**10,030**	**0.44**
	500_5	12.5	10,797	2.01	10,835	2.42	11,025	4.04	10,866	2.69	10,756	1.58	10,715	1.14	**10,629**	**0.30**
	500_8	20	11,995	1.47	12,041	1.85	12,178	2.91	12,076	2.14	11,974	1.19	11,957	1.09	**11,867**	**0.26**
	500_10	25	12,477	1.59	12,489	1.71	12,598	2.55	12,527	2.01	12,427	1.16	12,404	0.95	**12,339**	**0.40**
	Mean	-	7217	2.26	7248	2.85	7260	2.48	7272	3.17	7213	2.42	7182	1.68	**7114**	**0.55**
Percentage		-	1.43%	75.66%	1.85%	80.70%	2.01%	77.82%	2.17%	82.65%	1.37%	77.27%	0.95%	67.26%	-	-
$F = 5$	100_5	2.5	1928	4.61	1945	5.50	1902	3.13	1946	5.61	1934	4.95	1894	2.71	**1862**	**1.00**
	100_8	4	2182	3.77	2189	4.10	2152	2.35	2200	4.62	2198	4.52	2150	2.22	**2125**	**1.07**
	100_10	5	2329	3.45	2357	4.67	2305	2.28	2359	4.73	2356	4.64	2303	2.24	**2270**	**0.80**
	200_5	5	3650	3.09	3656	3.30	3630	2.50	3675	3.79	3646	2.99	3617	2.05	**3568**	**0.69**
	200_8	8	4050	2.76	4070	3.33	4024	2.11	4073	3.35	4057	2.96	4026	2.14	**3960**	**0.49**
	200_10	10	4289	2.48	4313	3.08	4263	1.87	4329	3.42	4311	2.98	4272	2.09	**4207**	**0.46**
	300_5	7.5	5297	2.32	5302	2.43	5309	2.49	5323	2.81	5304	2.39	5269	1.70	**5207**	**0.45**
	300_8	12	5949	2.68	5966	3.00	5913	2.06	5987	3.33	5938	2.47	5907	1.90	**5826**	**0.45**
	300_10	15	6239	2.12	6293	3.04	6229	1.90	6311	3.36	6244	2.19	6225	1.85	**6132**	**0.29**
	400_5	10	7030	5.97	7059	2.41	7080	2.67	7080	2.71	7007	1.67	6991	1.38	**6928**	**0.41**
	400_8	16	7807	2.06	7826	2.31	7816	2.17	7845	2.56	7778	1.68	7757	1.40	**7687**	**0.42**
	400_10	20	8153	1.57	8193	2.09	8188	1.97	8217	2.38	8141	1.37	8130	1.22	**8056**	**0.26**
	500_5	12.5	8781	2.27	8785	2.32	8895	3.54	8814	2.63	8738	1.79	8689	1.21	**8636**	**0.48**
	500_8	20	9669	1.72	9714	2.27	9743	2.42	9735	2.48	9652	1.48	9643	1.38	**9555**	**0.41**
	500_10	25	10,139	1.91	10,183	2.37	10,156	2.06	10,198	2.53	10,110	1.61	10,084	1.36	**10,008**	**0.48**
	Mean	-	5833	2.85	5857	3.08	5840	2.37	5873	3.35	5828	2.65	5797	1.79	**5735**	**0.54**
Percentage		-	1.68%	81.05%	2.08%	82.47%	1.80%	54.43%	2.35%	83.88%	1.60%	79.62%	1.07%	69.83%	-	-

Table 7. *Cont.*

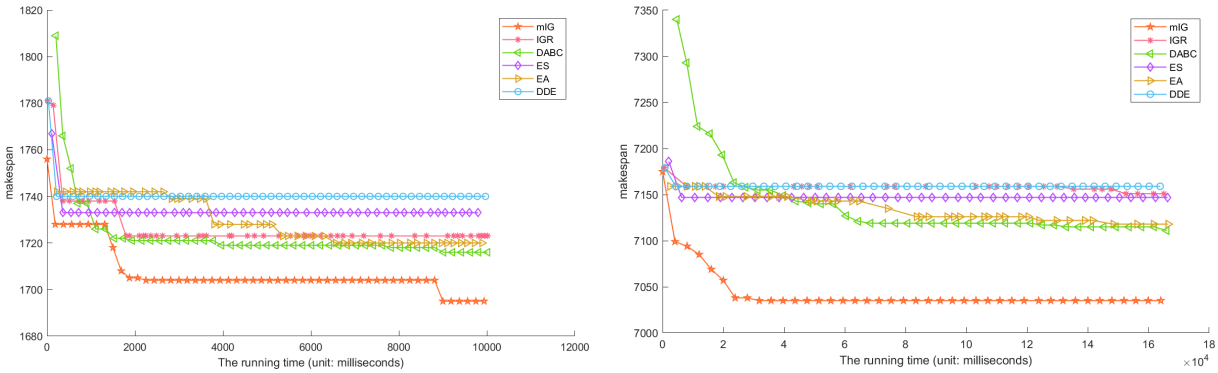
Factory	*J_M*	Time (s)	Algorithms													
			EA		DDE		DABC		IGR		ES		mIG0		mIG	
			Avg	ARPI	Avg	ARPI	Avg	ARPI	Avg	ARPI	Avg	ARPI	Avg	ARPI	Avg	ARPI
	100_5	2.5	1632	5.14	1637	5.41	1587	2.26	1636	5.42	1632	5.11	1593	2.64	**1564**	**0.75**
	100_8	4	1851	3.84	1851	3.87	1811	1.61	1858	4.21	1861	4.38	1818	2.01	**1795**	**0.75**
	100_10	5	2036	4.32	2053	5.22	1997	2.34	2047	4.90	2048	4.93	2005	2.75	**1970**	**1.01**
	200_5	5	3223	3.23	3081	3.46	3049	2.32	3091	3.84	3072	3.23	3046	2.21	**2998**	**0.62**
	200_8	8	3434	3.60	3456	4.24	3401	2.58	3477	4.87	3447	3.97	3403	2.55	**3352**	**1.06**
	200_10	10	3638	2.92	3659	3.58	3598	1.79	3670	3.89	3650	3.29	3609	2.10	**3553**	**0.51**
	300_5	7.5	4458	2.95	4473	3.28	4446	2.62	4489	3.65	4442	2.56	4427	2.18	**4363**	**0.63**
$F = 6$	300_8	12	5033	2.44	5051	2.89	5025	2.27	5063	3.11	5033	2.43	5012	1.94	**4937**	**0.45**
	300_10	15	5242	2.03	5267	2.46	5233	1.83	5290	2.92	5248	2.12	5228	1.69	**5166**	**0.52**
	400_5	10	5908	2.52	5928	2.89	5929	2.76	5939	3.10	5892	2.19	5855	1.57	**5811**	**0.70**
	400_8	16	6600	2.19	6615	2.42	6575	1.78	6636	2.74	6580	1.84	6567	1.65	**6498**	**0.49**
	400_10	20	6917	2.21	6945	2.68	6915	2.18	6953	2.80	6904	2.00	6873	1.55	**6809**	**0.56**
	500_5	12.5	7300	2.19	7302	2.21	7393	3.36	7326	2.53	7291	2.01	7260	1.53	**7198**	**0.59**
	500_8	20	8144	1.94	8155	2.09	8184	2.37	8172	2.31	8125	1.68	8096	1.29	**8040**	**0.50**
	500_10	25	8487	2.05	8519	2.48	8502	2.19	8526	2.55	8444	1.51	8421	1.25	**8353**	**0.37**
	Mean	-	4927	2.90	4933	3.28	4910	2.28	4945	3.52	4911	2.88	4881	1.93	**4827**	**0.63**
Percentage		-	2.02%	78.28%	2.15%	80.79%	1.69%	72.37%	2.39%	82.10%	1.71%	78.13%	1.10%	67.36%	-	-
	100_5	2.5	1406	4.28	1409	4.44	1381	2.39	1413	4.80	1418	5.17	1375	1.98	**1359**	**0.83**
	100_8	4	1630	4.50	1642	5.28	1591	2.03	1636	4.91	1631	4.63	1594	2.19	**1572**	**0.82**
	100_10	5	1781	3.89	1795	4.69	1743	1.71	1785	4.10	1790	4.43	1746	1.90	**1728**	**0.89**
	200_5	5	2670	3.50	2672	3.65	2640	2.26	2685	4.08	2655	3.02	2639	2.29	**2601**	**0.81**
	200_8	8	3001	3.29	3012	3.72	2972	2.27	3020	3.99	3015	3.76	2972	2.24	**2924**	**0.64**
	200_10	10	3201	2.93	3212	3.33	3153	1.42	3208	3.17	3206	3.08	3168	1.84	**3126**	**0.54**
	300_5	7.5	3868	2.64	3880	2.94	3861	2.43	3899	3.45	3877	2.88	3842	1.94	**3789**	**0.50**
$F = 7$	300_8	12	4329	2.78	4340	3.02	4315	2.35	4352	3.31	4315	2.40	4294	1.89	**4232**	**0.41**
	300_10	15	4573	2.43	4592	2.93	4552	1.93	4603	3.13	4559	2.15	4540	1.67	**4483**	**0.38**
	400_5	10	5120	2.65	5131	2.88	5135	2.82	5136	2.99	5091	2.05	5072	1.64	**5024**	**0.59**
	400_8	16	5725	2.43	5745	2.80	5716	2.19	5753	2.94	5706	2.07	5684	1.67	**5615**	**0.36**
	400_10	20	5970	2.24	5982	2.44	5963	2.10	5990	2.57	5959	2.05	5930	1.53	**5872**	**0.50**
	500_5	12.5	6323	2.31	6359	2.83	6362	2.83	6368	3.00	6285	1.69	6273	1.46	**6213**	**0.39**
	500_8	20	6986	1.88	7005	2.17	7009	2.15	7029	2.49	6965	1.53	6951	1.31	**6888**	**0.34**
	500_10	25	7369	1.90	7411	2.53	7382	2.07	7418	2.61	7344	1.53	7322	1.23	**7264**	**0.37**
	Mean	-	4263	2.91	4279	3.31	4252	2.20	4286	3.44	4254	2.83	4227	1.79	**4179**	**0.56**
Percentage		-	1.97%	80.76%	2.34%	83.08%	1.72%	74.55%	2.50%	83.72%	1.76%	80.21%	1.14%	68.72%	-	-

Best values are indicated in bold.

According to Table 7, (1) for most instances, the average makespan and RPI values obtained by mIG0 are smaller than those of EA, DDE, DABC, IGR, and ES, regardless of the number of factories being 2, 3, 4, 5, 6, and 7, respectively. The results demonstrate that the mIG0 is effective and has the ability to generate makespan. The advantages of mIG0 can be attributed to the fact that the proposed strategies, i.e., *Refresh_NEH_en*, $mVND$, and the two iterative processes, are designed based on the distributed multi-factories character of DBFSP. (2) For all the instances, the average makespan and RPI values obtained by mIG0 are better than those of mIG0, EA, DDE, DABC, IGR, and ES, regardless of the number of factories being 2, 3, 4, 5, 6, and 7, respectively. The results demonstrate that the mIG with refresh accelerated calculation has low time complexity and can have more opportunities to search potential solutions. Therefore, mIG shows superior performance compared with all the other algorithms. The main advantage of mIG relative to mIG0 can be attributed to the fact that the proposed refresh accelerated calculation based on job insertion can speed up the calculation of the objective and reduce the time complexity of the algorithm.

4.6. Evolutionary Curves and Interactions for the Compared Algorithms

This section further verifies the convergence of the algorithms by selecting two different scales, i.e., 100_6_10 and 400_7_10. The evolution curves of the mIG, ES, DABC, IGR, EA, and DDE algorithms are plotted as shown in Figure 6. The termination times for the two scales are $Timelimit = 10 \times J \times M$ and $Timelimit = 50 \times J \times M$, respectively. Different colors and symbols represent the six convergence curves obtained by six algorithms, respectively. The abscissa is the execution time of the algorithm (in milliseconds), and ordinate refers to the values of makespan.



(**a**) (**b**)

Figure 6. The evolutionary curves of compared algorithms. (**a**) 100_6_10. (**b**) 400_7_10.

From Figure 6a, we can observe that ES and DDE have the fastest convergence speed, but their solutions tend toward convergence as time increases. The evolutionary process of DABC and EA lasts for a long time, and the final results obtained are mediocre. The convergence speed of IGR is slightly faster than EA, and its solution is only better than ES and DDE. Obviously, mIG has good convergence and is constantly converging as time increases, and it is superior to other algorithms. Similarly, for the large-scale instance, mIG still has the best convergence, as shown in Figure 6b. The reason why the convergence curve of mIG is lower than those of compared algorithms may be that the proposed strategies, i.e., *Refresh_NEH_en*, *mVND*, and two iterative processes, can generate excellent solutions and effectively improve the convergence.

Although the above experiments have shown the superiority and competitiveness of the proposed mIG, it is necessary to verify whether its superiority is statistically significant. In view of this, a multifactor ANOVA analysis is done and uses different algorithms and the numbers of factories, jobs, and machines as influencing factors, respectively. From Figure 7a, the overall RPI values of all the compared algorithms are significantly different, in

which the proposed mIG algorithm remarkably outperforms the other algorithms, followed by mIG0, EA and ES, DABC, DDE, and IGR. Figure 7c,d shows that the values of RPI obtained by mIG are better than those of compared algorithms, and the mIG can remain stable when the numbers of factories, jobs, and machines increase. The ANOVA analysis plotted is illustrated in Figure 7 and shows the significant difference between mIG and other algorithms.

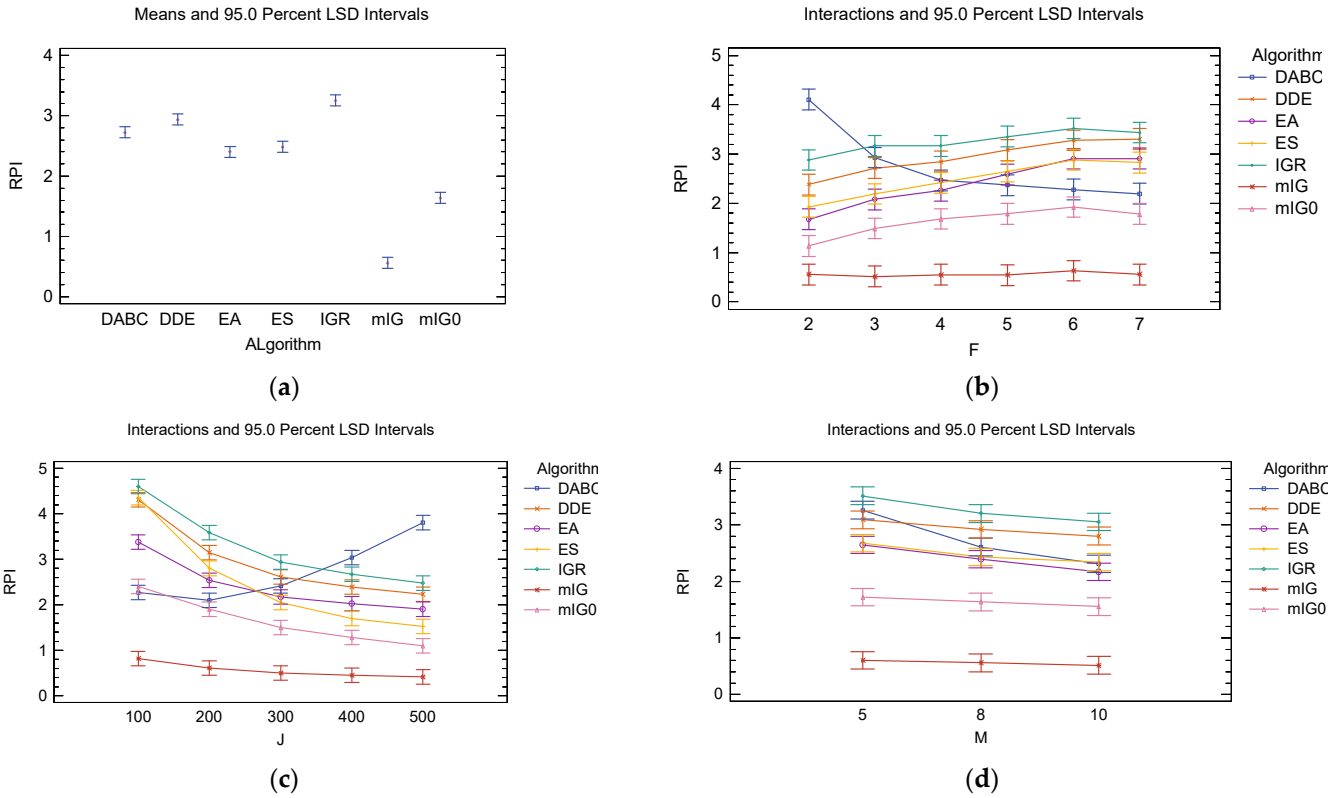


Figure 7. Interactions for ES, DABC, IGR, EA, DDE, mIG0, and mIG. (**a**) Means of all the compared algorithms. (**b**–**d**) are interactions of the numbers of factories, jobs, machines, and compared algorithms, respectively.

4.7. Friedman Tests

The Friedman test can verify whether multiple overall distributions are significantly different. Its original assumption is that all the algorithms involved in the comparison are not significantly different from each other. When a probability *p*-value is smaller than the given 0.05, the original assumption is rejected, and all algorithms are considered to be significantly different. Conversely, the original assumption cannot be rejected. It can be concluded that there are no significant differences between compared algorithms.

Table 8 gives the values of rank (Ranks), the number of test instances (CN), mean of RPI, standard deviation (Std. Deviation), minimum value (Min), and maximum value (Max) of makespan, respectively. The *p*-value obtained by the Friedman test is equal to 0.000, and its confidence level $\alpha = 0.050$. The values of Ranks, Mean, Std. Deviation, Min, and Max obtained by mIG are 1.04, 0.578, 0.2708, 0.11, and 1.53, and they are the smallest among all the compared algorithms. The proposed mIG performs very well in solving the DBFSP_SDST problem in general.

Table 8. Friedman test results (confidence level $\alpha = 0.050$).

Algorithms	Ranks	CN	Mean	Std. Deviation	Min	Max
EA	3.96	270	2.934	1.1850	0.49	6.39
DDE	5.36	270	3.469	1.3126	0.74	7.38
DABC	4.49	270	3.254	1.2517	1.13	11.29
IGR	6.49	270	3.789	1.2246	1.18	7.21
ES	4.24	270	3.014	1.3478	0.61	6.93
mIG0	2.41	270	2.160	0.8912	0.28	4.93
mIG	**1.04**	270	**0.578**	**0.2708**	**0.11**	**1.53**
p-value	0.000					

Best values are indicated in bold.

5. Conclusions and Future Research

There is very little literature about DBFSP_SDST. A MILP model is first constructed for DBFSP_SDST, and this paper uses the Gurobi solver to confirm its accuracy. Then, an efficient mIG algorithm is designed to optimize the above formulated model. For the proposed mIG algorithm, this article has done the following modifications.

1. A refresh acceleration calculation is proposed to reduce the complexity of the algorithm from $O(mn^2)$ to $O(mn)$.
2. A rapid evaluation mechanism, *Refresh_NEH_en*, is designed to reduce the computational complexity of the initialization process.
3. Iterative process I and II strategies are designed, and each iterative process is adopted by a certain probability to enhance the diversity of solutions from a global perspective.
4. According to characteristics of the distributed pattern, cross-factory and inner-factory strategies are presented to allocate the appropriate number and sequence of jobs for each factory, which balance the exploration and exploitation of the proposed mIG algorithm.
5. The proposed mIG algorithm obtains best solutions for a total of 270 instances when comparing to five state-of-the-art algorithms. The average makespan and RPI values of mIG are 1.93% and 78.35% better than the five comparison algorithms on average, respectively. The comprehensive results prove that the proposed mIG contains dual advantages of high quality and efficient solutions, which are more suitable for solving the DBFSP_SDST.

For future research, many issues of SDST-DBFSP need to be addressed urgently. First, multiple objectives should be considered, i.e., makespan, energy consumption, total flowtime, tardiness time [43] and earliness time, and so on. Second, from a practical production perspective, many uncertain factors should be considered, such as machine breakdowns, uncertain processing time, wrong operations, changes in due date, and so on. Last but not least, problem-specific operators or strategies should be designed according to the constraints and characteristics of problems.

Author Contributions: C.Z.: Conceptualization, Methodology, Data curation, Software, Validation, Writing—original draft. Y.H.: Conceptualization, Methodology, Software, Validation, Writing—original draft. Y.W.: Conceptualization, Methodology, Supervision, Writing—original draft. J.L.: Conceptualization, Methodology, Visualization, Investigation. K.G.: Conceptualization, Methodology, Writing—review and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China, grant numbers 61973203, 62106073, 62173216, and 62173356. We are grateful for Guangyue Young Scholar Innovation Team of Liaocheng University under grant number LCUGYTD2022-03.

Data Availability Statement: The data that support the findings of this study are available from the corresponding author upon reasonable request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Wang, L.; Shen, W. *Process Planning and Scheduling for Distributed Manufacturing (Springer Series in Advanced Manufacturing)*; Springer: Berlin/Heidelberg, Germany, 2007.
2. Koen, P.A. *The PDMA Handbook of New Product Development*; Wiley: Hoboken, NJ, USA, 2005.
3. Naderi, B.; Ruiz, R. The Distributed Permutation Flowshop Scheduling Problem. *Comput. Oper. Res.* **2010**, *37*, 754–768. [CrossRef]
4. Liu, H.; Gao, L. A Discrete Electromagnetism-Like Mechanism Algorithm for Solving Distributed Permutation Flowshop Scheduling Problem. In Proceedings of the 2010 International Conference on Manufacturing Automation, Hong Kong, China, 13–15 December 2010; pp. 156–163.
5. Gao, J.; Chen, R.; Deng, W.; Liu, Y. Solving Multi-Factory Flowshop Problems with a Novel Variable Neighbourhood Descent Algorithm. *J. Comput. Inf. Syst.* **2012**, *8*, 2025–2032.
6. Gao, J.; Chen, R.; Deng, W. An Efficient Tabu Search Algorithm for the Distributed Permutation Flowshop Scheduling Problem. *Int. J. Prod. Res.* **2012**, *51*, 641–651. [CrossRef]
7. Wang, S.; Wang, L.; Liu, M.; Xu, Y. An Effective Estimation of Distribution Algorithm for Solving the Distributed Permutation Flow-Shop Scheduling Problem. *Int. J. Prod. Econ.* **2013**, *145*, 387–396. [CrossRef]
8. Naderi, B.; Ruiz, R. A Scatter Search Algorithm for the Distributed Permutation Flowshop Scheduling Problem. *Eur. J. Oper. Res.* **2014**, *239*, 323–334. [CrossRef]
9. Fernandez-Viagas, V.; Framinan, J.M. A Bounded-Search Iterated Greedy Algorithm for the Distributed Permutation Flowshop Scheduling Problem. *Int. J. Prod. Res.* **2015**, *53*, 1111–1123. [CrossRef]
10. Komaki, M.; Malakooti, B. General Variable Neighborhood Search Algorithm to Minimize Makespan of the Distributed No-Wait Flow Shop Scheduling Problem. *Prod. Eng. Res. Devel.* **2017**, *11*, 315–329. [CrossRef]
11. Ruiz, R.; Pan, Q.-K.; Naderi, B. Iterated Greedy Methods for the Distributed Permutation Flowshop Scheduling Problem. *Omega* **2019**, *83*, 213–222. [CrossRef]
12. Pan, Q.-K.; Gao, L.; Wang, L. An Effective Cooperative Co-Evolutionary Algorithm for Distributed Flowshop Group Scheduling Problems. *IEEE Trans. Cybern.* **2022**, *52*, 5999–6012. [CrossRef]
13. Li, W.; Chen, X.; Li, J.; Sang, H.; Han, Y.; Du, S. An Improved Iterated Greedy Algorithm for Distributed Robotic Flowshop Scheduling Withorderconstraints. *Comput. Ind. Eng.* **2022**, *164*, 107907. [CrossRef]
14. Fernandez-Viagas, V.; Perez-Gonzalez, P.; Framinan, J.M. The Distributed Permutation Flow Shop to Minimise the Total Flowtime. *Comput. Ind. Eng.* **2018**, *118*, 464–477. [CrossRef]
15. Huang, J.-P.; Pan, Q.-K.; Gao, L. An Effective Iterated Greedy Method for the Distributed Permutation Flowshop Scheduling Problem with Sequence-Dependent Setup Times. *Swarm Evol. Comput.* **2020**, *59*, 100742. [CrossRef]
16. Han, X.; Han, Y.; Chen, Q.; Li, J.; Sang, H.; Liu, Y.; Pan, Q.; Nojima, Y. Distributed Flow Shop Scheduling with Sequence-Dependent Setup Times Using an Improved Iterated Greedy Algorithm. *Complex Syst. Model. Simul.* **2021**, *1*, 198–217. [CrossRef]
17. Li, Y.; Li, X.; Gao, L.; Zhang, B.; Pan, Q.-K.; Tasgetiren, M.F.; Meng, L. A Discrete Artificial Bee Colony Algorithm for Distributed Hybrid Flowshop Scheduling Problem with Sequence-Dependent Setup Times. *Int. J. Prod. Res.* **2021**, *59*, 3880–3899. [CrossRef]
18. Huang, J.-P.; Pan, Q.-K.; Miao, Z.-H.; Gao, L. Effective Constructive Heuristics and Discrete Bee Colony Optimization for Distributed Flowshop with Setup Times. *Eng. Appl. Artif. Intell.* **2021**, *97*, 104016. [CrossRef]
19. Karabulut, K.; Öztop, H.; Kizilay, D.; Tasgetiren, M.F.; Kandiller, L. An Evolution Strategy Approach for the Distributed Permutation Flowshop Scheduling Problem with Sequence-Dependent Setup Times. *Comput. Oper. Res.* **2022**, *142*, 105733. [CrossRef]
20. Song, H.-B.; Lin, J. A Genetic Programming Hyper-Heuristic for the Distributed Assembly Permutation Flow-Shop Scheduling Problem with Sequence Dependent Setup Times. *Swarm Evol. Comput.* **2021**, *60*, 100807. [CrossRef]
21. Companys, R.; Ribas, I. Efficient Constructive Procedures for the Distributed Blocking Flow Shop Scheduling Problem. In Proceedings of the 2015 International Conference on Industrial Engineering and Systems Management (IESM), Seville, Spain, 21–23 October 2015; pp. 92–98.
22. Zhang, G.; Liu, B.; Wang, L.; Yu, D.; Xing, K. Distributed Co-Evolutionary Memetic Algorithm for Distributed Hybrid Differentiation Flowshop Scheduling Problem. *IEEE Trans. Evol. Comput.* **2022**, *26*, 1043–1057. [CrossRef]
23. Ying, K.-C.; Lin, S.-W. Minimizing Makespan in Distributed Blocking Flowshops Using Hybrid Iterated Greedy Algorithms. *IEEE Access* **2017**, *5*, 15694–15705. [CrossRef]
24. Zhang, G.; Xing, K.; Cao, F. Discrete Differential Evolution Algorithm for Distributed Blocking Flowshop Scheduling with Makespan Criterion. *Eng. Appl. Artif. Intell.* **2018**, *76*, 96–107. [CrossRef]
25. Shao, Z.; Pi, D.; Shao, W. Hybrid Enhanced Discrete Fruit Fly Optimization Algorithm for Scheduling Blocking Flow-Shop in Distributed Environment. *Expert Syst. Appl.* **2020**, *145*, 113147. [CrossRef]
26. Zhao, F.; Zhao, L.; Wang, L.; Song, H. An Ensemble Discrete Differential Evolution for the Distributed Blocking Flowshop Scheduling with Minimizing Makespan Criterion. *Expert Syst. Appl.* **2020**, *160*, 113678. [CrossRef]
27. Han, X.; Han, Y.; Zhang, B.; Qin, H.; Li, J.; Liu, Y.; Gong, D. An Effective Iterative Greedy Algorithm for Distributed Blocking Flowshop Scheduling Problem with Balanced Energy Costs Criterion. *Appl. Soft Comput.* **2022**, *129*, 109502. [CrossRef]

28. Ruiz, R.; Stützle, T. A Simple and Effective Iterated Greedy Algorithm for the Permutation Flowshop Scheduling Problem. *Eur. J. Oper. Res.* **2007**, *177*, 2033–2049. [CrossRef]
29. Lin, S.-W.; Ying, K.-C.; Huang, C.-Y. Minimising Makespan in Distributed Permutation Flowshops Using a Modified Iterated Greedy Algorithm. *Int. J. Prod. Res.* **2013**, *51*, 5029–5038. [CrossRef]
30. Pan, Q.-K.; Ruiz, R. An Effective Iterated Greedy Algorithm for the Mixed No-Idle Permutation Flowshop Scheduling Problem. *Omega* **2014**, *44*, 41–50. [CrossRef]
31. Ying, K.-C.; Lin, S.-W.; Cheng, C.-Y.; He, C.-D. Iterated Reference Greedy Algorithm for Solving Distributed No-Idle Permutation Flowshop Scheduling Problems. *Comput. Ind. Eng.* **2017**, *110*, 413–423. [CrossRef]
32. Huang, Y.-Y.; Pan, Q.-K.; Huang, J.-P.; Suganthan, P.; Gao, L. An Improved Iterated Greedy Algorithm for the Distributed Assembly Permutation Flowshop Scheduling Problem. *Comput. Ind. Eng.* **2021**, *152*, 107021. [CrossRef]
33. Mao, J.; Pan, Q.; Miao, Z.; Gao, L. An Effective Multi-Start Iterated Greedy Algorithm to Minimize Makespan for the Distributed Permutation Flowshop Scheduling Problem with Preventive Maintenance. *Expert Syst. Appl.* **2021**, *169*, 114495. [CrossRef]
34. Ribas, I.; Companys, R.; Tort-Martorell, X. An Iterated Greedy Algorithm for Solving the Total Tardiness Parallel Blocking Flow Shop Scheduling Problem. *Expert Syst. Appl.* **2019**, *121*, 347–361. [CrossRef]
35. Qin, H.; Han, Y.; Chen, Q.; Li, J.; Sang, H. A Double Level Mutation Iterated Greedy Algorithm for Blocking Hybrid Flow Shop Scheduling. *Control Decis.* **2022**, *37*, 2323–2332. [CrossRef]
36. Chen, S.; Pan, Q.-K.; Gao, L. Production Scheduling for Blocking Flowshop in Distributed Environment Using Effective Heuristics and Iterated Greedy Algorithm. *Robot. Comput. Integr. Manuf.* **2021**, *71*, 102155. [CrossRef]
37. Öztop, H.; Fatih Tasgetiren, M.; Eliiyi, D.T.; Pan, Q.-K. Metaheuristic Algorithms for the Hybrid Flowshop Scheduling Problem. *Comput. Oper. Res.* **2019**, *111*, 177–196. [CrossRef]
38. Taillard, E. Some Efficient Heuristic Methods for the Flow Shop Sequencing Problem. *Eur. J. Oper. Res.* **1990**, *47*, 65–74. [CrossRef]
39. Missaoui, A.; Ruiz, R. A Parameter-Less Iterated Greedy Method for the Hybrid Flowshop Scheduling Problem with Setup Times and Due Date Windows. *Eur. J. Oper. Res.* **2022**, *303*, 99–113. [CrossRef]
40. Karabulut, K.; Kizilay, D.; Tasgetiren, M.F.; Gao, L.; Kandiller, L. An Evolution Strategy Approach for the Distributed Blocking Flowshop Scheduling Problem. *Comput. Ind. Eng.* **2022**, *163*, 107832. [CrossRef]
41. Meng, L.; Zhang, C.; Ren, Y.; Zhang, B.; Lv, C. Mixed-Integer Linear Programming and Constraint Programming Formulations for Solving Distributed Flexible Job Shop Scheduling Problem. *Comput. Ind. Eng.* **2020**, *142*, 106347. [CrossRef]
42. Meng, L.; Gao, K.; Ren, Y.; Zhang, B.; Sang, H.; Chaoyong, Z. Novel MILP and CP Models for Distributed Hybrid Flowshop Scheduling Problem with Sequence-Dependent Setup Times. *Swarm Evol. Comput.* **2022**, *71*, 101058. [CrossRef]
43. Zhao, F.; Di, S.; Wang, L. A Hyperheuristic With Q-Learning for the Multiobjective Energy-Efficient Distributed Blocking Flow Shop Scheduling Problem. *IEEE Trans. Cybern.* **2022**, 1–14. [CrossRef]