AIMS Mathematics, 10(5): 11488–11513.
DOI: 10.3934/math.2025523
Received: 27 March 2025
Revised: 30 April 2025
Accepted: 12 May 2025
Published: 20 May 2025

https://www.aimspress.com/journal/Math

Research article

Two-stage iterated greedy algorithm for distributed flexible assembly permutation flowshop scheduling problems with sequence-dependent setup times

Yuan-Zhen Li*, Lei-Lei Meng, and Biao Zhang

School of Computer Science, Liaocheng University, Liaocheng 252000, China

*** Correspondence:** Email: liyuanzhen@lcu.edu.cn.

Abstract: In this study, the distributed flexible assembly permutation flowshop scheduling problem (DFAPFSP) with sequence-dependent setup times and makespan criterion was investigated. The DFAPFSP comprises two distinct phases: a distributed permutation flowshop in the initial stage, followed by an integration phase. The integration stage, which employs multiple parallel assembly machines, achieves significantly higher throughput efficiency compared to monolithic assembly machine architectures in high-volume manufacturing scenarios. A novel mixed-integer linear programming model was established to describe the problem. The DFAPFSP can be divided into two stages: production and assembly. A two-stage iterated greedy (TSIG) algorithm was designed based on the two-stage characteristics of the DFAPFSP. In the first stage, the production plan is optimized, and in the second stage, the assembly plan is optimized. The destruction, reconstruction, and local search algorithms in the two stages and acceptance criterion were redesigned. Numerous computational experiments and performance evaluations were performed by comparing the TSIG algorithm with state-of-the-art algorithms. The results and discussions show that the proposed TSIG algorithm is better than its peers for solving the DFAPFSP.

Keywords: optimization; distributed permutation flowshop scheduling; flexible assembly, sequence-dependent setup times; makespan; two-stage iterated greedy algorithm
Mathematics Subject Classification: 68T40, 68W20

1. Introduction

With the continuous development of the economic globalization and the gradual globalization of the manufacturing industry, manufacturing enterprises are encountering unprecedented competition [1,2]. To improve their competitiveness, the traditional single-factory production mode must be transformed into a multi-factory production mode [3,4]. Enterprises can coordinate and control factories in different places to improve production efficiency and stability, reduce costs, and enhance competitiveness [5]. Scheduling is an important component of the manufacturing system and directly affects the efficiency and competitiveness of enterprises [6,7]. An efficient scheduling method can achieve efficiency improvement, energy savings, consumption reduction, emission reduction, and cost reduction in the enterprise's production and manufacturing process [8,9].

Distributed manufacturing system scheduling has received considerable attention [10–12]. The first study [13] on the distributed permutation flowshop scheduling problem (DPFSP) was conducted in 2010. In 2013, a DPFSP with a single assembly machine was observed; it is called the distributed assembly permutation flowshop scheduling problem (DAPFSP) [14]. However, the production efficiency of a single assembly machine is not high, and this has become a bottleneck in the production system in some cases. In our previous work [15], production efficiency was improved by using an effective scheduling scheme. In this study, another scheme for adding more assembly machines is studied. To solve the aforementioned bottleneck, companies typically use multiple identical assembly machines. In this study, a distributed flexible assembly permutation flowshop scheduling problem (DFAPFSP) is considered, in which multiple assembly machines replace the single machine assembly in the DAPFSP. Compared with the DPFSP and DAPFSP, the DFAPFSP is more complex because it must generate a plan composed of a production sub-schedule for jobs and assembly sub-schedule for products. Our contributions are summarized as follows: (1)we establish a mathematical model for the DFAPFSP with sequence-dependent setup times, (2) a two-stage iterated greedy (TSIG) algorithm for solving DFAPFP is proposed, in which the first stage adjusts the job sequence and the second stage optimizes the product sequence, (3)we redesign the destruction, reconstruction, and local search algorithms based on problem characteristics in two stages, and (4) we verify the performance by using experimental statistics and analysis.

The remainder of this paper is organized as follows. Section 2 presents a review of related studies on the DPFSP, DAPFSP, and DFAPFSP. Section 3 introduces the DFAPFSP with makespan criterion and provides an illustrative example. Section 4 presents the proposed TSIG algorithm with its detailed design. Section 5 describes the experimental calibration of the TSIG and verifies the new mathematical model presented in Section 3. Section 6 details the computational results and comparisons with state-of-the-art algorithms. Finally, Section 7 provides concluding remarks and suggestions for future work.

2. Literature review

Recently, the DPFSP has been receiving more research attention [13]. The DPFSP is a complex scheduling problem that aims to determine which factory each job is assigned to and the order of jobs in each factory. Various metaheuristics have been proposed to solve the DPFSP, including the hybrid immune algorithm [16], tabu algorithm [17], iterated greedy (IG) [18–21], scatter search [22], chemical reaction optimization [23], discrete artificial bee colony [24,25], and discrete fruit fly

optimization [26]. In addition, numerous variants of the DPFSP have been proposed, such as distributed blocking flowshop scheduling [27], distributed no-wait flowshops [28,29], distributed flexible job shop scheduling [30,31], green scheduling for the DPFSP [5], inverse scheduling for the DPFSP [32], the bi-objective DPFSP [33], and the DPFSP with sequence-dependent setup times [26]. Several review papers have systematically summarized the DPFSP [10,11].

Among the many variants of the DPFSP, the DAPFSP is the most important and can be considered as a combination of the DPFSP and assembly scheduling problem. The DAPFSP has also received considerable attention. Hatami et al. published the first paper on the DAPFSP [14]. Wang et al. [34] proposed a memetic algorithm for solving the DAPFSP by combining the estimation of distribution algorithm and local search. Deng et al. [35] proposed a mixed-integer linear programming (MILP) model for the DAPFSP. Subsequently, a competitive memetic algorithm was proposed to solve the DAPFSP. Researchers have proposed many metaheuristics to solve the DAPFSP, including the backtracking search hyperheuristic algorithm [36] that designs ten simple and effective heuristic rules to construct a set of low-level heuristic (LLH) rules, and uses the backtracking search algorithm as a high-level strategy to manipulate the LLH rules to operate on the solution space, the hybrid biogeography-based optimization algorithm [37] that simulates the migration, mutation, and selection processes of species in different habitats to achieve global optimal solution search for complex problems, three-dimensional matrix-cube-based estimation of distribution algorithm [38] that describes the distribution pattern of excellent solutions by establishing a probability model, and generates a new population based on the model sampling to replace the crossover and mutation operations of traditional genetic algorithms, and biased-randomized iterated local search [39] which is a metaheuristic framework that iteratively applies local search, perturbation, and evaluation of the solution against the acceptance criterion.

In these previous studies, the DAPFSP was explored by minimizing the makespan criterion, whereas in following studies, other optimization objectives were investigated. Sang et al. [40] presented three variants of the discrete invasive weed optimization for the DAPFSP with total flowtime criterion. Huang et al. [41] also studied the DAPFSP with total flowtime criterion based on an IG algorithm. Li et al. [15] focused on the DAPFSP with total tardiness criterion. Based on the DAPFSP research, some researchers have added conditions depending on the production reality, such as blocking [42,43], no-idle [44], sequence-dependent setup times [45], no-wait [46], hybrid flowshop [47], fuzzy processing time [48], and three stages (the production, transportation, and assembly stages) [49]. Some researchers have focused on the multi-objective optimization of the DAPFSP. Zhao et al. [50] proposed a brainstorm optimization approach for solving the multi-objective energy-efficient distributed assembly no-wait flowshop scheduling problem. Zhang et al. [51] studied a dual-objective energy-efficient DFAPFSP with minimum makespan and energy consumption. Huang et al. [52] presented a two-phase evolutionary algorithm for the DAPFSP with the total flowtime and total tardiness criteria.

Limited research has been conducted on the DFAPFSP. Zhang et al. [53] studied the DFAPFSP with a sequence-independent setup time which means that the preparation time required to switch between different jobs does not depend on the order of the jobs. They proposed a constructive heuristic (two-phase heuristic search, TPHS) and two hybrid metaheuristics (hybrid variable neighborhood search, HVNS, and hybrid particle swarm optimization, HPSO). Numerical experiments showed that both the TPHS and metaheuristics can obtain a reliable solution that matches the running time. Ying et al. [54] also addressed the DFAPFSP with a sequence-independent

setup time to minimize the completion time. A MILP model of the DFAPFSP was proposed for the first time. Subsequently, constructive heuristics and customized metaheuristics IG_{PD} (iterated greedy with product destruction mechanism) were proposed to solve the DFAPFSP. Extensive computational experiments showed that the proposed IG_{PD} has been the best approach to date.

Zhang et al. [55] proposed a memetic algorithm with meta-Lamarckian learning and simplex search for the DFAPFSP. Statistical results showed that the proposed algorithm was significantly better than the existing algorithms. Yang and Xu [56] studied the DFAPFSP with batch delivery, which is more complicated because of transportation costs. This analysis and summary of previous studies indicate that many studies have been conducted on the DPFSP and DAPFSP, but limited attention has been paid to the DFAPFSP.

The IG algorithm has achieved excellent performance in job shop scheduling [57]. In our recent three studies [15,18], we used the IG algorithm to solve distributed flowshop scheduling problems and compared it with state-of-the-art algorithms. The algorithms compared included the modified IG, cloud-theory-based IG, scatter search, bounded-search IG, evolutionary, discrete artificial bee colony, iterated local search, hybrid discrete invasive weed optimization, biogeography-based optimization, and competitive memetic algorithms. The results showed that the IG algorithm outperformed the others. Therefore, in this study, we use the IG algorithm to solve the DFAPFSP with makespan criterion and sequence-dependent setup times which means preparation time required to switch between different jobs depends on the order of the jobs, which is expected to yield better results.

3. Problem description

The mathematical model of the DFAPFSP is described as follows. Two stages are considered: production and assembly; n jobs are processed in the first stage and assembled into t products in the second stage. The first stage involves F production factories. All production factories are identical. Each production factory is a flowshop composed of m machines (M_1, M_2, M_3, …, M_m). The processing time of job j on machine i is the same for all production factories. All jobs are available at initialization.

After all jobs for a product are completed, they can be assembled into a product on the assembly machine (AM) in the second stage. The second stage involves q assembly factories. Each assembly factory has only one assembly machine. Therefore, in the following, the assembly factory and assembly machine are equivalent. The performances of all assembly machines are the same. Assembly can be conducted in any assembly factory. n, m, F, q, and t are known in advance.

Equipment failures, maintenance, and other interruptions are not considered. Operations cannot be transferred to other machines or factories during the processing. A production (assembly) machine can process only one job (product) at a time, and a job (product) can be processed on only one production (assembly) machine at a time. The job (product) operation of each production (assembly) machine cannot be interrupted until it is completed. Each machine has a sequence-dependent setup time to prepare for the processing of each job. Table 1 lists the indices, parameters, and decision variables used.

Table 1. Indices, parameters, and decision variables.

Parameters	
n	The number of jobs to be processed.
m	The number of machines in the production stage.
F	The number of factories.
t	The number of products.
q	The number of machines in the assembly stage.
g_{js}	Binary parameter: 1 if job j is a part of product s, and 0 otherwise.
p_{ji}	The processing time of job j on machine i.
S_{kji}	The setup time between jobs k and j on machine i.
pp_s	The processing time of product s on the assembly machine.
SA_{ls}	The setup time between products l and s on the assembly machine.
D	A large positive constant.
Indexes	
k, j	Denote jobs, $k, j \in 1, \ldots, n$.
f	Denotes a factory in the first stage, $f \in 1, \ldots, F$.
i	Denotes a machine in the first stage, $i \in 0,1, \ldots, m$, where 0 is a dummy machine.
l, s	Denote products, $l, s \in 1, \ldots, t$.
a	Denotes an assembly machine in the second stage, $a \in 1, \ldots, q$.
Variables	
$X_{k,j}$	Binary variable: 1 if job j is processed after job k, and 0 otherwise.
$Y_{j,f}$	Binary variable: 1 if job j is processed in factory f.
C_{ji}	Continuous variable for the completion time of job j on machine i.
$XA_{l,s}$	Binary variable: 1 if product l is processed after product s, and 0 otherwise.
$YA_{l,a}$	Binary variable: 1 if product l is processed in assembly factory a.
CA_s	Completion time of product s on the assembly machine.

The key parameters and variables are emphasized as follows. If job j is a part of product s, then g_{js}=1. The processing time for job j on machine i and the processing time for product s on the assembly machine are represented as p_{ji} and pp_s, respectively. The setup time between jobs k and j on machine i and the setup time between products l and s on the assembly machine are S_{kji} and SA_{ls}, respectively. There are four binary variables, namely $X_{k,j}$, $Y_{j,f}$, $XA_{l,s}$, and $YA_{l,a}$. If job j is processed after job k, then $X_{k,j}$=1; if job j is processed in factory f, then $Y_{j,f}$=1; if product l is processed after product s, then $XA_{l,s}$=1; if product l is processed in assembly factory a, then $YA_{l,a}$=1. Otherwise, these four binary variables are 0. The completion time of job j on machine i and the completion time of product s on the assembly machine are denoted as C_{ji} and CA_s.

3.1. MILP models

The MILP model for the DFAPFSP is extended from the models proposed by Hatami et al. [14] and Naderi et al. [13]. Equation (1) is our optimization function.

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

Subject to

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

$$C_{j,i} \geq C_{j,i-1} + p_{j,i}, \forall j, i \tag{3}$$

$$C_{k,i} \geq C_{j,i} + S_{j,k,i} + p_{k,i} - D \cdot X_{k,j} - D\left(1 - Y_{k,f}\right) - D\left(1 - Y_{j,f}\right), \forall i, k \in \{1,2,\dots,n-1\}, j > k, f \tag{4}$$

$$\begin{gathered} C_{\{(j,i)\}} \geq C_{\{(k,i)\}} + S_{\{(k,j,i)\}} + p_{\{(j,i)\}} - D\left(1 - X_{\{(k,j)\}}\right) - D\left(1 - Y_{\{(k,f)\}}\right) \\ -D\left(1 - Y_{\{(j,f)\}}\right), \forall i, k \in 1,2,\dots,n-1, j > k, f \end{gathered} \tag{5}$$

$$CA_s \geq \left(C_{j,m} \times g_{js}\right) + pp_s, \forall j, s \tag{6}$$

$$\begin{gathered} CA_s \geq CA_l + SA_{ls} + pp_s - D \cdot XA_{s,l} - D\left(1 - YA_{s,a}\right) \\ -D\left(1 - YA_{l,a}\right), \forall s \in \{1,2,\dots,t-1\}, l > s, a \end{gathered} \tag{7}$$

$$\begin{gathered} CA_l \geq CA_s + SA_{sl} + pp_l - D\left(1 - XA_{s,l}\right) - D\left(1 - YA_{s,a}\right) \\ -D\left(1 - YA_{l,a}\right), \forall s \in \{1,2,\dots,t-1\}, l > s, a \end{gathered} \tag{8}$$

$$C_{max} \geq CA_s, \forall s \tag{9}$$

$$C_{j,i} \geq 0, \forall j, i \tag{10}$$

$$X_{k,j} \in \{0,1\}, \forall k \in \{1,2,\dots,n-1\}, j > k \tag{11}$$

$$Y_{j,f} \in \{0,1\}, \forall j, f \tag{12}$$

$$XA_{s,l} \in 0,1, \forall s \in \{1,2,\dots,t-1\}, s > l \tag{13}$$

$$YA_{la} \in 0,1, \forall l, a \tag{14}$$

$$CA_s \geq 0, \forall s \tag{15}$$

Note that machine 0 is a dummy and that $C_{j0} = 0, \forall j$. Constraint set (2) implies that each job is processed in only one factory. Constraint set (3) indicates that a job cannot be processed on two machines simultaneously and the relationship between the completion time of a job on two adjacent machines. Constraint sets (4) and (5) are dichotomous pairs of constraints related to each possible job pair. If job k is processed after job j in factory f, constraint (4) provides the relationship between the completion times of jobs k and j. On the contrary, if job k is processed before job j in factory f, constraint (5) describes the relationship between the completion times of jobs k and j. Similarly, constraint sets (7) and (8) are dichotomous pairs of constraints related to each possible product pair. Constraint (6) implies that each product can be assembled only after all its components are completed in the first stage. Constraint set (9) defines the makespan, and constraints (10)–(15) define the value range of the variables.

3.2. Illustrative example

We consider an example in which n=6, F=3, m=2, q=2, and t=3. Jobs 1 and 6, 2 and 3, and 4

and 5 are assembled into products 1, 2, and 3, respectively. Other parameters, such as the processing and setup times, are listed in Tables 2–5. We consider a scheduling scheme as follows: Jobs 1 and 3, 4 and 6, and 5 and 2 are processed sequentially in factories 1, 2, and 3, respectively. In the assembly stage, product 3 is processed on assembly machine 1, and products 1 and 2 are processed on assembly machine 2. Figure 1 shows a Gantt chart. As shown in Figure 1, the completion time of this scheduling scheme is 163 time units.

Table 2. Processing or assembly time and product assembly plan.

Job	M_1	M_2	Product	AM
J_1	48	27	P_1	28
J_6	38	14		
J_2	36	41	P_2	26
J_3	18	48		
J_4	31	30	P_3	32
J_5	42	36		

Table 3. Setup time between jobs on machine M_1.

	J_1	J_2	J_3	J_4	J_5	J_6
J_0	7	5	6	4	6	8
J_1	-	5	6	4	6	8
J_2	8	-	9	8	4	5
J_3	6	9	-	9	9	6
J_4	8	6	6	-	8	4
J_5	9	8	7	8	-	9
J_6	6	9	5	8	12	-

Table 4. Setup time between jobs on machine M_2.

	J_1	J_2	J_3	J_4	J_5	J_6
J_0	3	5	7	5	7	4
J_1	-	4	7	6	5	6
J_2	2	-	8	8	6	4
J_3	4	8	-	7	6	3
J_4	6	5	6	-	9	6
J_5	8	7	6	9	-	5
J_6	7	6	6	7	5	-

Table 5. Setup time between products.

	P_1	P_2	P_3
P_0	8	6	7
P_1	7	7	8
P_2	8	4	4
P_3	6	9	5

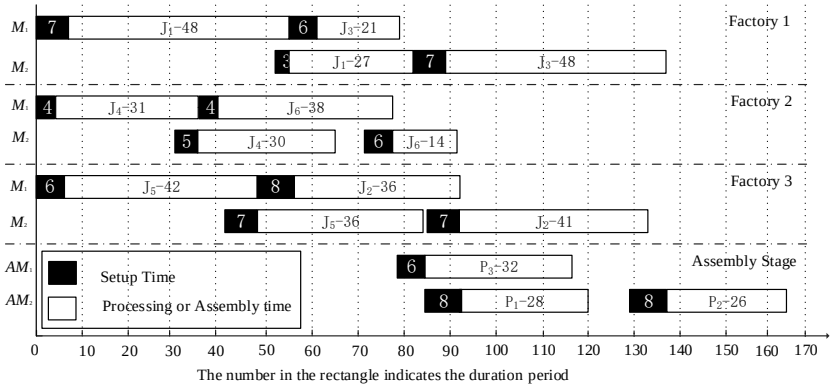

Figure 1. Gantt chart for a solution to the example problem.

4. TSIG algorithm for the DFAPFSP

The IG algorithm is a fast and effective intelligent optimization algorithm with a simple structure and few parameters. The IG algorithm destroys and reconstructs the current solution through iterations to produce a series of solutions. The new solution is updated by using an acceptance criterion. The IG algorithm has been applied in many fields, particularly for the flowshop scheduling problem, which shows good performance.

4.1. Solution representation and TSIG framework

The representation of the solutions is important. A reasonable representation of the solutions can help the algorithms obtain better results. The solutions of recent studies [53,54], which are similar to this study, are divided into two parts. The first part represents the production factory allocation of jobs in the production stage and processing sequence in each factory, and the second part represents the assembly factory allocation of products and processing sequence of products in the assembly stage. However, the second part can be obtained from the first part using a greedy algorithm [53,54]. Only the first part is decisive. We studied this representation and determined that it leads to local optimization for the DFAPFSP with sequence-dependent setup times. As shown in Figure 2(a), products 1 and 2 are arranged for processing in the assembly factories. Figure 2(b) shows the solution obtained according to the method in [53,54]. That is, a better solution exists when the positions of P_1 and P_2 change, as shown in Figure 2(c). The reason for this result is that the setup time of the concerned problem is sequence-dependent.

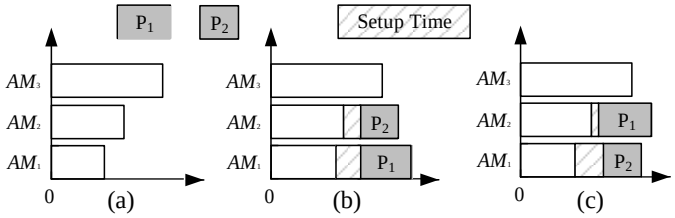

Figure 2. Greedy method leads to local optima.

The solution representation is the same as in the literature [53,54]. The difference is that the second and first halves are independent of one another. A solution can be expressed as

$$sol =< \lambda, \xi >, \text{where } \lambda = (\lambda_1, \lambda_2, \dots, \lambda_f, \dots, \lambda_F)$$

indicates the job allocation in the production stage and

$$\xi = (\xi_1, \xi_2, \dots, \xi_a, \dots, \xi_t)$$

indicates the product allocation in the assembly stage. The job sequence of factory *f* is

$$\lambda_f = \left(\lambda_{f1}, \lambda_{f2}, \dots, \lambda_{fn_f}\right),$$

where n_f is the number of jobs in factory *f*. Similarly, the product sequence of assembly factory *a* is

$$\xi_a = \left(\xi_{a1}, \xi_{a2}, \dots, \xi_{an_a}\right).$$

Therefore, the scheduling scheme of the example in Section 3.2 is

$$\lambda = \{\lambda_1, \lambda_2, \lambda_3\} = \{1,3\}, \{4,6\}, \{5,2\}, \xi = \{\xi_1, \xi_2\} = \{3\}, \{1,2\}.$$

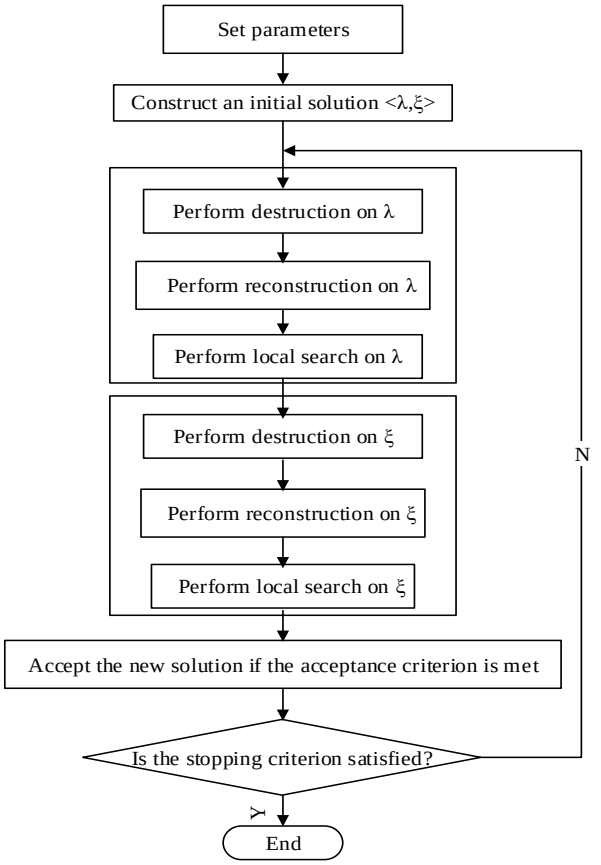

Figure 3. TSIG algorithm framework for the DFAPFSP.

The DFAPFSP is complex and cannot be handled by using a one-stage method. The first stage outputs products with release times. When multiple jobs in the production stage are extracted, they cannot be reinserted because the intermediate solution generated in the first stage cannot be

evaluated. If the job production plan in the first stage is determined, the problem becomes a multi-parallel machine processing problem with release time, which is equivalent to the multi-traveling salesman problem and is NP hard. The approaches used in the original studies [53,54] can obtain better solutions under the condition of sequence-independent setup times. This problem requires further study in the context of sequence-dependent setup times. Based on the problem characteristics, we designed the TSIG algorithm. Figure 3 shows the TSIG algorithm framework.

4.2. Initial solution

A good initial solution significantly impacts the performance of the algorithm. Heuristic H_{11} was proposed in the first study on the DAPFSP [14]. The latest studies [15,40] also show that the H_{11} algorithm has a good performance. Therefore, we designed a new heuristic based on H_{11} and called it Improved H_{11} (IH_{11}), as shown in Algorithm 1. First, a temporary sequence of products is obtained (Line 1). Then, a sequence of jobs for each product is obtained (Lines 2.1 and 2.2). The sequence (π_T) of all jobs is obtained by combining the sequences of each product (Line 3). Each job in π_T is inserted into the optimal position in the product factories in sequence to minimize the makespan (Lines 4–6). Line 7 calculates the release time for each product. The product sequence (Line 8) is obtained in ascending order of the release time. Finally, the products are successively inserted into the assembly factory to obtain the best makespan. In Algorithm 1, the complexity of the code in Lines 1–3 is $O(t^2)$, $O(n^2)$, and $O(1)$, respectively. The complexity of the code in Lines 4–6 is

$$O\left(((n-f)*2f+(n-f-1)*(n-f)/2)*m\right)=O(0.5mn^2+mnf-1.5mf^2).$$

The complexity of the code in Lines 7–9 is $O(t)$, $O(t^2)$, and $O(t^2)$, respectively. Finally, the complexity of Algorithm 1 is

$$O(t^2+n^2+1+0.5mn^2+mnf-1.5mf^2+t+t^2+t^2)=O(mn^2).$$

Algorithm 1. IH_{11}

(1) A temporary product sequence, π, is obtained by using the shortest processing time rule on assembly processing times;
(2) Use the method described in Lines 2.1 and 2.2 to obtain the job sequence (π_h) of each; product h;
(2.1) Sort jobs belonging to product h in ascending order of completion time to obtain a temporary sequence, R_h;
(2.2) Extract jobs in R_h individually in order, attempt to insert them into all possible positions of a production factory, and finally insert them into the best position;
(3) Construct complete job sequence (π_T) by combining all partial job sequences (π_h) following product sequence π;
(4) **for each** job j in π_T **do**
(5) Insert job j into the best position in all production factories to minimize the makespan;
(6) **endfor**
(7) Calculate the release time for each product;
(8) Sort the products in ascending order of the release times;
(9) Insert products sequentially into the optimal location of the distributed assembly factories to minimize the makespan;

4.3. Destruction procedure for the first stage

Algorithm 2 presents the pseudocode for the destruction procedure for the first stage. It randomly selects a product and extracts all jobs belonging to the product.

Algorithm 2. Destruction1(λ)

(1) $J = \emptyset$;
(2) Randomly select a product θ;
(3) Extract all jobs of product θ and append them to J;
(4) **return** J and λ;

4.4. Construction procedure for the first stage

The construction procedure for the first stage returns all jobs in J to λ, as shown in Algorithm 3. The operations are as follows: Randomly select a job in J and place it in the best position in λ. In the second stage, an important parameter is the release time of the products. After all jobs belonging to a product are processed, the product can be processed in the second stage; the shorter the release time of a product, the better. However, the release times of many products are difficult to evaluate. In the TSIG algorithm, a greedy method (Lines 5–8) is used to obtain ξ for the second stage. The final completion time is then obtained and used to measure the fitness of each position.

Algorithm 3. Construction1(λ, ξ, J)

(1) **While** sizeof(J) > 0 **do**
(2) Randomly extract job *j* in J;
(3) C_{min}= inf;
(4) **for** $f = 1$ to F **do**
(5) **for** $p = 1$ to $n_f + 1$ **do**
(6) Insert job *j* at position *p* in factory *f*;
(7) Calculate the release time for each product;
(8) Sort the products in ascending order of the release times;
(9) Insert products sequentially into the optimal location of the distributed assembly factories to minimize the makespan;
(10) C is the temporary makespan;
(11) **if** $C < C_{min}$ **do** C_{min}=C; $bestpos = p$; $bestf = f$; **endif**
(12) **endfor**
(13) **endfor**
(14) $Insert\ j\ at\ position\ bestpos\ in\ factory\ bestf$;
(15) **endwhile**
(16) **return** $< \lambda, \xi >$;

4.5. Local search for the first stage

The basic operation of the local search for the first stage (LocalSearch1 in Algorithm 4) is as follows: Randomly select a job, extract the job from the current position, insert the extracted job into a new random position, and calculate the makespan using the same greedy method (Lines 5–7 in Algorithm 4) as described in Subsection 4.4. If a better solution is obtained, then the original solution is replaced. The basic operation is performed $iter_L S$ times, and $C(\lambda, \xi)$ is the makespan of a solution $< \lambda, \xi >$.

Algorithm 4. LocalSearch1$(\lambda, \xi, iter_L S)$

(1) **for** $iter = 1$ **to** $iter_l S$ **do**
(2) $< \lambda', \xi' \geq < \lambda, \xi >$;
(3) Randomly select job j and remove it from λ';
(4) Randomly select a position at which to insert job j and a new λ' obtained;
(5) Calculate the release time for each product;
(6) Sort the products in ascending order of the release times;
(7) Insert products sequentially into the optimal position of the distributed assembly factories to minimize the makespan and a new ξ' obtained;
(8) **if** $\mathrm{C}(\lambda', \xi') < \mathrm{C}(\lambda, \xi)$ **do** $< \lambda, \xi \geq < \lambda', \xi' >$ **endif**
(9) **endif**
(10) **return** $< \lambda, \xi >$;

4.6. Destruction procedure for the second stage

Note that the problem considered in the second stage is the parallel machine problem with release time. The release time is determined by the production scheduling of the first stage. The destruction procedure for the second stage (Destruction2 in Algorithm 5) involves randomly selecting d products and extracting them from ξ.

Algorithm 5. Destruction2(ξ, d)

(1) $\Theta = \emptyset$;
(2) **While** sizeof$(\Theta) < d$ **do**
(3) Randomly select a product θ from ξ;
(4) Extract product θ and append it to Θ;
(5) **endwhile**
(6) **return** Θ and ξ;

4.7. Construction procedure for the second stage

In the second stage, the construction procedure inserts the extracted product at the optimal position of the assembly sequence. Algorithm 6 presents the pseudocode.

Algorithm 6. Construction2(ξ, Θ)

(1) **While** sizeof$(\Theta) > 0$ **do**
(2) Randomly extract product θ from Θ;
(3) Test product θ in all possible positions of ξ;
(4) Insert θ at position p leading to the lowest completion time;
(5) **endwhile**
(6) **return** ξ;

4.8. Local search for the second stage

In the second stage, a local search attempts to extract a product and insert it into other positions to improve the performance of the solution. To balance efficiency and performance, when half of all products have been tested and have not been improved, the local search stops. Algorithm 7 presents the pseudocode of the local search for the second stage.

Algorithm 7. LocalSearch2(sol)

(1) $Cnt = 0$;
(2) $a = rand\%q, j = rand\%n_a$;
(3) **While** $Cnt<t/2$ **do**
(4) Extract product $\theta = \xi_{aj}$ from ξ;
(5) Insert product θ into all possible positions of ξ;
(6) sol'= solution with the lowest makespan after inserting product θ into ξ;
(7) **if** $C(sol') < C(sol)$ **then**
(8) $sol = sol', Cnt = 0$;
(9) **else**
(10) $Cnt = Cnt + 1$;
(11) **end**
(12) $j = j + 1$;
(13) **if** $j == n_a$**then** $a = (a + 1)\%q + 1, j = 1$; **endif**
(14) **endwhile**
(15) **return** sol

4.9. Acceptance criterion

After the destruction, construction, and local search procedures in the two stages, a new solution is obtained. This new solution may worsen or improve. Generally, an acceptance criterion allows the worse solution to be accepted with a certain probability, which can help the IG algorithm exit the local optima to ensure a continuous evolution of the IG algorithm. Researchers have designed various acceptance criteria [58]. However, in our recent study [15], we concluded that rejecting a worse solution can achieve a better final result in some cases. We designed a new acceptance criterion, as shown in Algorithm 8, based on the acceptance criteria in [54] and [15]. If the new solution is better than the current solution, it replaces the current solution. If it is also better than the current optimal solution (sol^*), it replaces the current optimal solution. Parameter β is used in the new acceptance criterion. If β equals zero, then the worse solution is not accepted. If β is larger than 0, the worse solution is accepted with a probability of e^{-RPD}, where $RPD = \big(C(sol') - C(sol)\big)/C(sol) \times 100$.

Algorithm 7. AcceptanceCriterion(sol, sol', β)

(1) **If** $C(sol') < C(sol)$ **then**
(2) $sol = sol'$;
(3) **If** $C(sol') < C(sol^*)$ **then** $sol^* = sol'$; **endif**
(4) **else if** $\beta > 0$ **then**
(5) **if**$rand(0,1) < exp(-RPD)$ **then** $sol = sol'$; **endif**
(6) **endif**

4.10. Procedure of the proposed TSIG

Algorithm 9 presents the complete implementation of the TSIG algorithm. First, heuristic IH_{11} is used to obtain an initial solution, which is temporarily considered the optimal solution. The TSIG then enters the loop operation and exits the loop when the termination conditions are satisfied. In the loop, the following operations are performed on the temporary solution (sol'): Destruction1,

Construction1, LocalSearch1, Destruction2, Construction2, LocalSearch2, and AcceptanceCriterion. Finally, the results are returned.

Algorithm 8. TSIG$(d, iter_L S, iter_L 2, \beta)$

(1) $sol = IH_{11}$;
(2) $sol^* = sol$; //best solution so far
(3) **While** stopping condition not met **do**
(4) $sol' = sol$;
(5) $[J, \lambda'] = Destruction1(\lambda')$;
(6) $sol' = Construction1(\lambda', \xi', J)$;
(7) $sol' = LocalSearch1(\lambda', \xi', iter_L S)$;
(8) $[\Theta, \xi'] = Destruction2(\xi', d)$;
(9) $sol' = Construction2(\xi', \Theta)$;
(10) $sol' = \text{LocalSearch2}(sol')$;
(11) $AcceptanceCriterion(sol, sol', \beta)$
(12) **endwhile**
(13) **return** sol^*

5. Experimental calibration and mathematical model verification

The parameters of the proposed TSIG algorithm were adjusted to enable the TSIG algorithm to show its optimal performance. The TSIG algorithm has four parameters: *d*, *iter_ LS*, *iter_S2*, and β. For these four parameters, different possible values were determined based on the relevant literature. In the literature [53,54], the optimal settings of parameters may differ for different problem scales. For small-scale instances, the possible values of *d* are 1, 3, and 5; that of *iter_LS*, 0, 10, and 30; *iter_S2*, 1, 3, and 5; and β, 0, and 1. These factors lead to a total of 3×3×3×2=54 different configurations for the TSIG algorithm (small-scale). For large-scale instances, the only difference is the possible values of *iter_S2*, which are 0, 1, and 3. Similarly, the TSIG algorithm has 54 different configurations for large-scale instances.

The method in [53,54] was used to generate test instances. The calibration experiments involved two types of test instances: large- and small-scale. Table 6 lists their parameters. The processing times for each job and product are a random number between 1 and 99. The setup time for all machines is uniformly generated from [1,20]. Small-scale data involve 3×2×2×2×3=72 combinations, and large-scale data involve 2×2×2×2×2=32 combinations. Each combination produces an instance. Thus, 104 numerical instances are obtained.

Table 6. Instance parameters.

	n	*m*	*F*	*t*	*q*	p_{ji}	pp_s	S_{kji}	SA_{ls}
small-scale	{20,24,30}	{2,3}	{2,3}	{6,8}	{2,3,4}	[1,99]	[1,99]	[1,20]	[1,20]
large-scale	{100, 200 };	{6,8}	{5,10}	{30,40}	{6,8}	[1,99]	[1,99]	[1,20]	[1,20]

All algorithms were coded using C++, and the IDE was MS VS 2017. We ran configurations on an Intel (R) Core (TM) i7-4790 CPU@3.60 GHz with 16.00 GB of RAM on a Windows 10 operating system and stopped them when the termination criterion of the maximum elapsed CPU time of 6×*n*×*m* ms was met, where n and m are the number of jobs and machines, respectively. The

algorithm under each configuration was run independently five times. Thus, 72×54×5=19,440 small- and 32×54×5=8640 large-scale numerical results were generated.

We measured the relative percentage increase (RPI) from the results as follows:

$$RPI = \frac{100\times(C-C^*)}{C^*} \tag{16}$$

where C is the makespan obtained in each configuration and instance, and C^* is the best makespan obtained for that instance among the 54×5=270 results. Considering all instances, we obtained 270×104= 28,080 RPI results for the calibration of the TSIG algorithm.

The mean plots with 95% Tukey honest significant difference (HSD) confidence intervals (shown in Figures 4 and 5) were used to analyze the best values of the parameters. From the results of the experimental analysis shown in Figure 4, we can easily obtain the following parameter settings for the small-scale instances: d=3, *iter_LS*=10, *iter_S2*=3, and β=0. Figure 5 shows the best parameter configuration for large-scale datasets: d=3, *iter_LS*=10, *iter_S2*=1, and β=0.

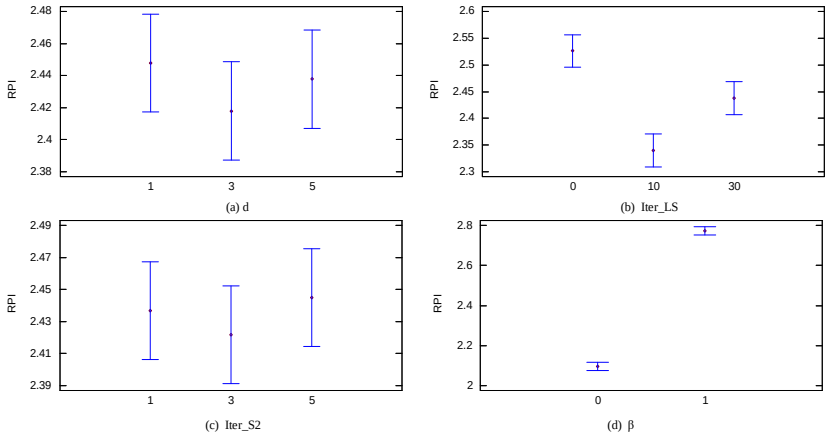

Figure 4. Mean plots with 95% Tukey honest significant difference (HSD) confidence intervals for all factors for the TSIG calibration (small-scale instances).

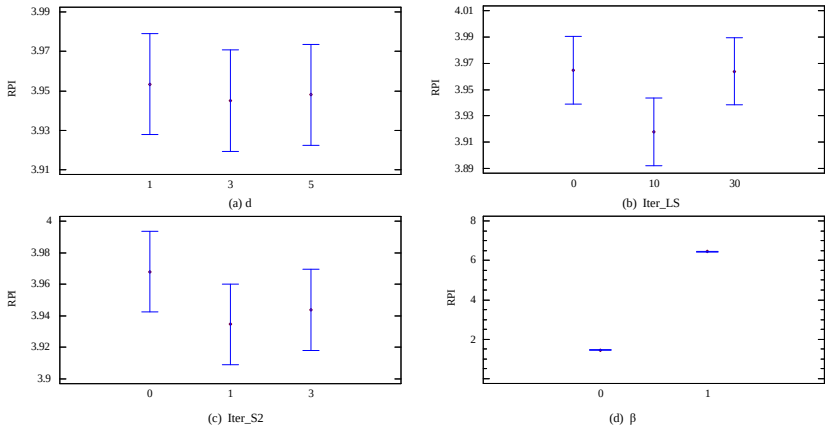

Figure 5. Mean plots with 95% Tukey HSD confidence intervals for all factors for TSIG calibration (large-scale instances).

The mathematical model in Section 3.1 is implemented using CPLEX to solve small-scale instances. For the most complex instance in which n, m, F, t, and q are 30, 3, 3, 8, and 4, respectively, the solver does not end running for 2 hours. Then, we use CPLEX to solve 72 small-scale instances and get results in 600 s. A solution for each instance is obtained at the set termination time. For the briefest purposes, detailed results are not listed, which can be obtained through our email. The mathematical model in Section 3.1 is verified to be correct, but time-consuming.

6. Computational evaluation

The TSIG algorithm was compared with state-of-the-art algorithms, namely, HVSN [53], HPSO [53], and IG_{PD} [54], which are similar to our research topic. Table 7 lists the parameters of the compared algorithms, which are sourced from their original papers. The programming and running environments of all algorithms were the same as those mentioned in Section 5. To show the performance of all algorithms panoramically, we set three algorithm termination criteria:

$$t=v\times m\times n \text{ ms},$$

where v was tested at three values: 20, 40, and 60. In addition, the experimental dataset was the same as that described in Section 5. The only difference was that each combination produced five numerical instances. Thus, we obtained

$$104\times 5=520$$

instances (including 360 small- and 160 large-scale instances). Each algorithm was run independently 10 times for each instance.

Table 7. Parameter setting for HPSO [53], HVNS [53], and IG_{PD} [54].

HPSO	$M_{iter}=500, C_1=C_2=2, P_{size}=50, \gamma=0.5, \varpi=1.2, \tau=0.5;$
HVNS	$M_{iter}=500;$
IG_{PD}	*No parameters*

Tables 8–13 list the average RPI (ARPI) values grouped by factory, production machine, job, assembly machine, and product. Tables 8 and 9 present the ARPI values of all competitive algorithms when the termination time was 20mn ms. The TSIG algorithm is the best for small-scale instances, followed by IGPD, HPSO, and HVNS. For large-scale instances, for m=6, the performance of TSIG ranks second but is similar to that of IG_{PD}. In other large-scale instances, TSIG is optimal. Therefore, in large-scale instances, TSIG is the best, followed by IG_{PD}, HPSO, and finally HVNS.

A multifactor analysis of variance (ANOVA) was conducted to determine whether the differences observed in Tables 8 and 9 were significant. The types of algorithms, number of jobs, number of production machines, number of factories, number of assembly machines, and number of products are considered factors.

Figure 6 shows the mean plots of the types of algorithms, interactions of the algorithms and number of factories, interactions of the algorithms and number of jobs, interactions of the algorithms and number of production machines, interactions of the algorithms and number of assembly machines, and interactions of the algorithms and number of products. We employed a 95% confidence level and the Tukey HSD confidence interval.

Figure 6 shows that the total ARPI values generated by the four algorithms are statistically different. From best to worst, the algorithms are as follows: TSIG, IG_{PD}, HPSO, and HVNS. When the scale of the DFAPFSP is small (f=2, 3; n=20, 24, 30; m=2, 3; q=2, 3, 4; t=6, 8), each algorithm can obtain a better solution. However, the TSIG algorithm is the best. When the scale of the DFAPFSP is large (f=5, 10; n=100, 200; m=6, 8; q=6, 8; t=30, 40), the performance differences of the four algorithms improve. However, the proposed TSIG algorithm yields the best results.

Table 8. ARPI for 20mn ms (minimum ARPI values are in bold; small-scale instances).

Type	HPSO	HVNS	IG_{PD}	TSIG
f=2	2.215	2.916	1.983	**1.281**
f=3	3.559	4.086	3.155	**1.834**
n=20	3.237	3.810	2.722	**2.060**
n=24	3.802	4.851	3.381	**1.810**
n=30	1.623	1.840	1.604	**0.804**
m=2	2.730	3.161	2.433	**1.483**
m=3	3.045	3.841	2.706	**1.632**
t=6	2.949	3.553	2.616	**1.514**
t=8	2.826	3.448	2.523	**1.602**
q=2	2.879	3.492	2.504	**1.534**
q=3	2.918	3.514	2.575	**1.522**
q=4	2.866	3.497	2.628	**1.618**
Mean	2.887	3.501	2.569	**1.558**

Table 9. ARPI for 20mn ms (minimum ARPI values are in bold; large-scale instances).

Type	HPSO	HVNS	IG_{PD}	TSIG
f=5	6.164	9.444	1.973	**1.858**
f=10	8.862	12.482	2.605	**2.330**
n=100	8.384	11.902	2.145	**2.093**
n=200	6.641	10.023	2.434	**2.095**
m=6	7.637	10.924	2.184	2.204
m=8	7.389	11.001	2.394	**1.984**
t=30	7.412	10.778	2.204	**1.987**
t=40	7.613	11.147	2.374	**2.201**
q=6	7.453	10.938	2.239	**2.029**
q=8	7.573	10.988	2.339	**2.160**
Mean	7.513	10.963	2.289	**2.094**

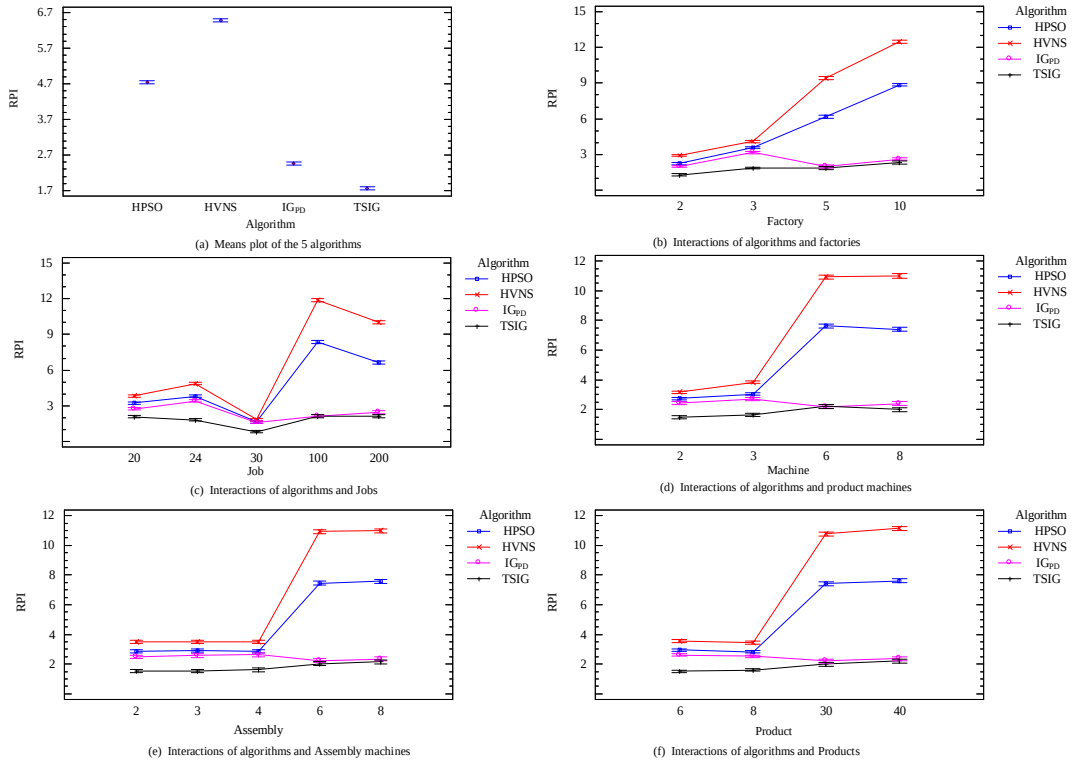

Figure 6. Mean plots, interactions, and 95.0% Tukey HSD intervals at $t = 20mn$ ms.

Tables 10–13 show the ARPI values of all competitive algorithms at 40*mn* and 60*mn* ms, respectively. Figures 7 and 8 show the mean plots, interactions, and 95.0% Tukey HSD intervals at t=40*mn* ms and those at t=60*mn* ms, respectively. When the scale of the DFAPFSP is small, the difference between all algorithms is large. When the scale of the DFAPFSP becomes larger, the problem becomes more difficult, as well as obtaining the optimal solution. The TSIG algorithm is the best except for n=100 or m=6; and when n=100 or m=6, it is second-best and performs almost as well as the best. The ranking of the competitive algorithms is the same as that of 20*mn* ms, which can be concluded from these tables and figures.

Table 10. ARPI for 40*mn* ms (minimum ARPI values are in bold; small-scale instances).

Type	HPSO	HVNS	IG_{PD}	TSIG
f=2	1.841	2.672	1.706	**1.153**
f=3	3.009	3.727	2.708	**1.639**
n=20	2.734	3.455	2.300	**1.883**
n=24	3.249	4.498	2.971	**1.624**
n=30	1.292	1.646	1.350	**0.681**
m=2	2.272	2.878	2.073	**1.318**
t=6	2.502	3.247	2.279	**1.345**

Continued on next page

Type	HPSO	HVNS	IG_{PD}	TSIG
t=8	2.348	3.152	2.135	**1.447**
q=2	2.395	3.192	2.158	**1.348**
q=3	2.457	3.209	2.201	**1.361**
q=4	2.422	3.199	2.262	**1.479**
Mean	2.425	3.200	2.207	**1.396**

Table 11. ARPI for 40mn ms (minimum ARPI values are in bold; large-scale instances).

Type	HPSO	HVNS	IG_{PD}	TSIG
f=5	5.528	9.248	1.456	**1.406**
f=10	8.183	12.234	1.618	**1.412**
n=100	7.703	11.662	**1.496**	1.533
n=200	6.008	9.820	1.578	**1.286**
m=6	6.994	10.709	**1.446**	1.478
m=8	6.717	10.773	1.628	**1.344**
t=30	6.724	10.566	1.477	**1.350**
t=40	6.987	10.916	1.596	**1.469**
q=6	6.802	10.729	1.500	**1.375**
q=8	6.909	10.754	1.573	**1.443**
Mean	6.855	10.741	1.537	**1.409**

Table 12. ARPI for 60mn ms (minimum ARPI values are in bold; small-scale instances).

Type	HPSO	HVNS	IG_{PD}	TSIG
f=2	1.609	2.556	1.548	**1.077**
f=3	2.732	3.521	2.463	**1.533**
n=20	2.447	3.258	2.055	**1.780**
n=24	2.931	4.322	2.751	**1.512**
n=30	1.133	1.534	1.211	**0.623**
m=2	2.018	2.725	1.880	**1.222**
m=3	2.323	3.352	2.131	**1.388**
t=6	2.260	3.088	2.093	**1.259**
t=8	2.081	2.989	1.918	**1.351**
q=2	2.128	3.036	1.946	**1.252**
q=3	2.209	3.044	1.996	**1.275**
q=4	2.174	3.035	2.074	**1.388**
Mean	2.170	3.038	2.005	**1.305**

Table 13. ARPI for 60*mn* ms (minimum ARPI values are in bold; large-scale instances).

Type	HPSO	HVNS	IG_{PD}	TSIG
f=5	5.159	9.138	**1.192**	**1.192**
f=10	7.838	12.092	1.164	**0.965**
n=100	7.332	11.505	**1.236**	1.275
n=200	5.666	9.725	1.119	**0.882**
m=6	6.640	10.575	**1.095**	1.131
m=8	6.358	10.654	1.260	**1.026**
t=30	6.377	10.420	1.126	**1.028**
t=40	6.621	10.809	1.230	**1.129**
q=6	6.458	10.604	1.134	**1.053**
q=8	6.539	10.625	1.222	**1.103**
Mean	6.499	10.615	1.178	**1.078**

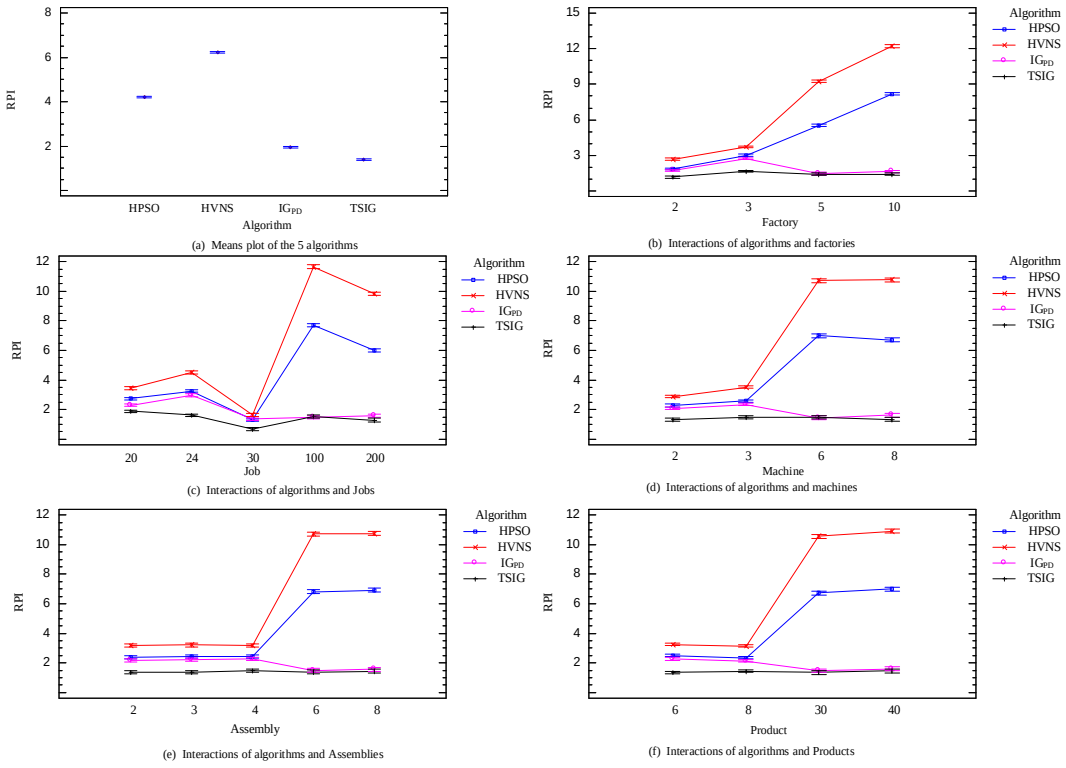

Figure 7. Means plots, interactions, and 95.0% Tukey HSD intervals at *t* = 40*mn* ms.

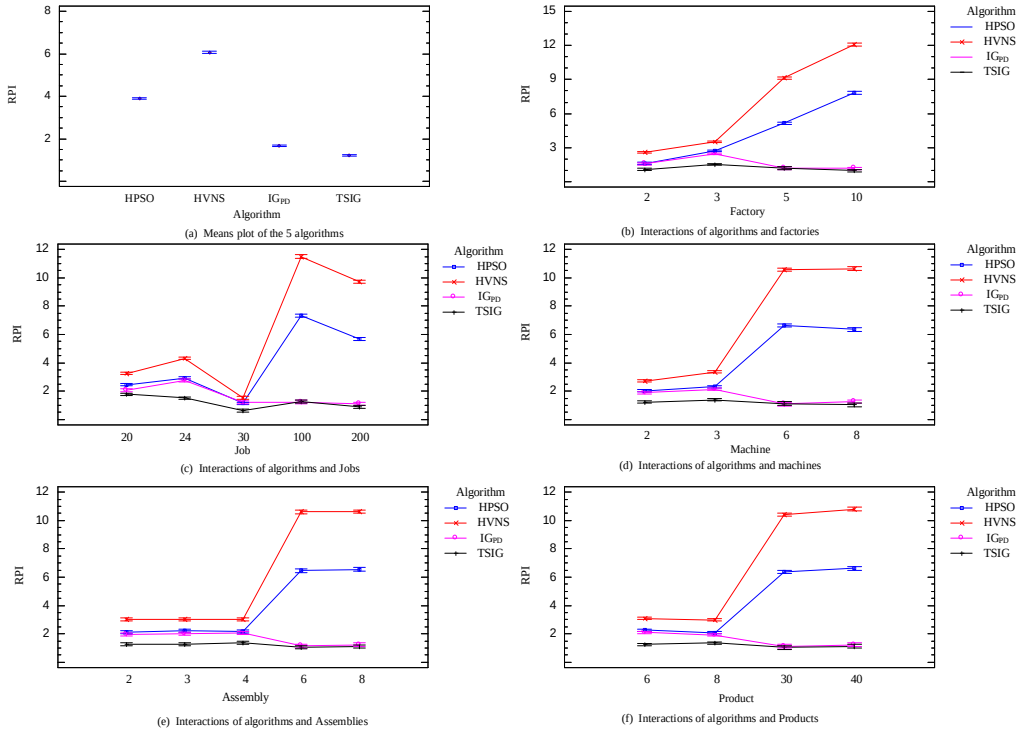

Figure 8. Means plots, interactions, and 95.0% Tukey HSD intervals at $t = 60mn$ ms.

The experimental results show that TSIG performs better than HVNS, HPSO, and IG_{PD}. This should be due to the following reasons. On the one hand, as mentioned earlier, the IG algorithm has shown excellent performance in solving combinatorial optimization problems, especially in job shop scheduling problems. On the other hand, the IG algorithm has been improved based on the characteristics of the problem to be solved. For example, two parts are used in the representation of the solution, representing the scheduling scheme for the production stage and the assembly stage, respectively; the destruction, reconstruction, and local search algorithms are redesigned based on the characteristics of the production stage and the assembly stage.

7. Conclusions

This study addressed the DFAPFSP, which involves multiple assembly machines. An efficient scheduling algorithm is crucial for the DFAPFSP. However, few studies have been conducted on this topic. First, we established a new mathematical model for the DFAPFSP with sequence-dependent setup times and makespan criterion. The TSIG algorithm was proposed to solve the DFAPFSP based on problem-specific knowledge. All IG algorithm components were redesigned. Several experiments were conducted to verify that the TSIG algorithm is superior to existing algorithms.

The DFAPFSP is a new research topic, with only a few studies being conducted. In fact, the DFAPFSP may face more complex situations in actual production, such as dynamic order changes and uncertainties of machine failures. This study does not account for these dynamic uncertainties, constituting a primary limitation of the current work. Significant research efforts remain necessary to address these gaps. Future investigations could explore alternative optimization objectives, such as

minimizing total flow time or reducing total tardiness. The energy-efficient DFAPFSP should also be investigated for optimizing multiple objectives simultaneously. Finally, the DFAPFSP can be integrated with the transportation process for research purposes.

Author contributions

Yuanzhen Li: Software, Investigation, Validation, Conceptualization, Methodology, Writing-original draft preparation; Leilei Meng: Writing-review & editing, Validation, Formal analysis; Biao Zhang: Data Curation, Investigation, Visualization. All authors have read and approved the final version of the manuscript for publication.

Use of Generative-AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

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

Conflict of interest

The authors declare that there is no conflict of interest.

References

1. K. Karabulut, H. Öztop, D. Kizilay, M. F. Tasgetiren, L. Kandiller, An evolution strategy approach for the distributed permutation flowshop scheduling problem with sequence-dependent setup times, *Comput. Oper. Res.*, **142** (2022), 105733. https://doi.org/10.1016/j.cor.2022.105733
2. L. Meng, K. Gao, Y. Ren, B. Zhang, H. Sang, C. Zhang, Novel MILP and CP models for distributed hybrid flowshop scheduling problem with sequence-dependent setup times, *Swarm Evol. Comput.*, **71** (2022), 101058. https://doi.org/10.1016/j.swevo.2022.101058
3. S. Schulz, M. Schönheit, J. S. Neufeld, Multi-objective carbon-efficient scheduling in distributed permutation flow shops under consideration of transportation efforts, *J. Clean. Prod.*, **365** (2022), 132551. https://doi.org/10.1016/j.jclepro.2022.132551
4. W. Li, X. Chen, J. Li, H. Sang, Y. Han, S. Du, An improved iterated greedy algorithm for distributed robotic flowshop scheduling with order constraints, *Comput. Ind. Eng.*, **164** (2022), 107907. https://doi.org/10.1016/j.cie.2021.107907
5. Y. Li, Q. Pan, K. Gao, M. F. Tasgetiren, B. Zhang, J. Li, A green scheduling algorithm for the distributed flowshop problem, *Appl. Soft Comput.*, **109** (2021), 107526. https://doi.org/10.1016/j.asoc.2021.107526
6. M. Li, G. Wang, A review of green shop scheduling problem, *Inf. Sci.*, **589** (2022), 478–496. https://doi.org/10.1016/j.ins.2021.12.122

7. Z. Wang, Q. Pan, L. Gao, Y. Wang, An effective two-stage iterated greedy algorithm to minimize total tardiness for the distributed flowshop group scheduling problem, *Swarm Evol. Comput.*, **74** (2022), 101143. https://doi.org/10.1016/j.swevo.2022.101143
8. B. Zhang, Q. Pan, L. Meng, C. Lu, J. Mou, J. Li, An automatic multi-objective evolutionary algorithm for the hybrid flowshop scheduling problem with consistent sublots, *Knowledge-Based Syst.*, **238** (2022), 107819. https://doi.org/10.1016/j.knosys.2021.107819
9. W. Niu, J. Li, A two-stage cooperative evolutionary algorithm for energy-efficient distributed group blocking flow shop with setup carryover in precast systems, *Knowledge-Based Syst.*, **257** (2022), 109890. https://doi.org/10.1016/j.knosys.2022.109890
10. Y. Fu, Y. Hou, Z. Wang, X. Wu, K. Gao, L. Wang, Distributed scheduling problems in intelligent manufacturing systems, *Tsinghua Sci. Technol.*, **26** (2021), 625–645. https://doi.org/10.26599/TST.2021.9010009
11. N. Bagheri Rad, J. Behnamian, Recent trends in distributed production network scheduling problem, *Artif. Intell. Rev.*, **55** (2021), 2945–2995. https://doi.org/10.1007/s10462-021-10081-5
12. C. Lu, Y. Huang, L. Meng, L. Gao, B. Zhang, J. Zhou, A Pareto-based collaborative multi-objective optimization algorithm for energy-efficient scheduling of distributed permutation flow-shop with limited buffers, *Robot. Comput.-Integr. Manuf.*, **74** (2022), 102277. https://doi.org/10.1016/j.rcim.2021.102277
13. B. Naderi, R. Ruiz, The distributed permutation flowshop scheduling problem, *Comput. Oper. Res.*, **37** (2010), 754–768. https://doi.org/10.1016/j.cor.2009.06.019
14. S. Hatami, R. Ruiz, C. Andres-Romano, The distributed assembly permutation flowshop scheduling problem, *Int. J. Prod. Res.*, **51** (2013), 5292–5308. https://doi.org/10.1080/00207543.2013.807955
15. Y. Li, Q. Pan, R. Ruiz, H. Sang, A referenced iterated greedy algorithm for the distributed assembly mixed no-idle permutation flowshop scheduling problem with the total tardiness criterion, *Knowledge-Based Syst.*, **239** (2022), 108036. https://doi.org/10.1016/j.knosys.2021.108036
16. Y. Xu, L. Wang, S. Wang, M. Liu, An effective hybrid immune algorithm for solving the distributed permutation flow-shop scheduling problem, *Eng. Optim.*, **46** (2014), 1269–1283. https://doi.org/10.1080/0305215X.2013.827673
17. J. Gao, R. Chen, W. Deng, An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem, *Int. J. Prod. Res.*, **51** (2013), 641–651. https://doi.org/10.1080/00207543.2011.644819
18. Y. Li, Q. Pan, X. He, H. Sang, K. Gao, X. Jing, The distributed flowshop scheduling problem with delivery dates and cumulative payoffs, *Comput. Ind. Eng.*, **165** (2022), 107961. https://doi.org/10.1016/j.cie.2022.107961
19. X. Han, Y. Han, B. Zhang, H. Qin, J. Li, Y. Liu, et al., An effective iterative greedy algorithm for distributed blocking flowshop scheduling problem with balanced energy costs criterion, *Appl. Soft Comput.*, **129** (2022), 109502. https://doi.org/10.1016/j.asoc.2022.109502
20. C. Lu, Q. Liu, B. Zhang, L. Yin, A Pareto-based hybrid iterated greedy algorithm for energy-efficient scheduling of distributed hybrid flowshop, *Expert Syst. Appl.*, **204** (2022), 117555. https://doi.org/10.1016/j.eswa.2022.117555

21. H. Qin, Y. Han, Y. Liu, J. Li, Q. Pan, X. Han, A collaborative iterative greedy algorithm for the scheduling of distributed heterogeneous hybrid flow shop with blocking constraints, *Expert Syst. Appl.*, **201** (2022), 117256. https://doi.org/10.1016/j.eswa.2022.117256
22. B. Naderi, R. Ruiz, A scatter search algorithm for the distributed permutation flowshop scheduling problem, *Eur. J. Oper. Res.*, **239** (2014), 323–334. https://doi.org/10.1016/j.ejor.2014.05.024
23. H. Bargaoui, O. Belkahla Driss, K. Ghédira, A novel chemical reaction optimization for the distributed permutation flowshop scheduling problem with makespan criterion, *Comput. Ind. Eng.*, **111** (2017), 239–250. https://doi.org/10.1016/j.cie.2017.07.020
24. H. Li, X. Li, L. Gao, A discrete artificial bee colony algorithm for the distributed heterogeneous no-wait flowshop scheduling problem, *Appl. Soft Comput.*, **100** (2021), 106946. https://doi.org/10.1016/j.asoc.2020.106946
25. Y. Yu, F. Zhang, G. Yang, Y. Wang, J. Huang, Y. Han, A discrete artificial bee colony method based on variable neighborhood structures for the distributed permutation flowshop problem with sequence-dependent setup times, *Swarm Evol. Comput.*, **75** (2022), 101179. https://doi.org/10.1016/j.swevo.2022.101179
26. H. Guo, H. Sang, B. Zhang, L. Meng, L. Liu, An effective metaheuristic with a differential flight strategy for the distributed permutation flowshop scheduling problem with sequence-dependent setup times, *Knowledge-Based Syst.*, **242** (2022), 108328. https://doi.org/10.1016/j.knosys.2022.108328
27. G. Zhang, K. Xing, Differential evolution metaheuristics for distributed limited-buffer flowshop scheduling with makespan criterion, *Comput. Oper. Res.*, **108** (2019), 33–43. https://doi.org/10.1016/j.cor.2019.04.002
28. S. Lin, K. Ying, Minimizing makespan for solving the distributed no-wait flowshop scheduling problem, *Comput. Ind. Eng.*, **99** (2016), 202–209. https://doi.org/10.1016/j.cie.2016.07.027
29. C. Cheng, K. Ying, H. Chen, H. Lu, Minimising makespan in distributed mixed no-idle flowshops, *Int. J. Prod. Res.*, **57** (2019), 48–60. https://doi.org/10.1080/00207543.2018.1457812
30. L. Meng, C. Zhang, Y. Ren, B. Zhang, C. Lv, Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem, *Comput. Ind. Eng.*, **142** (2020), 106347. https://doi.org/10.1016/j.cie.2020.106347
31. X. Tao, Q. Pan, L. Gao, An efficient self-adaptive artificial bee colony algorithm for the distributed resource-constrained hybrid flowshop problem, *Comput. Ind. Eng.*, **169** (2022), 108200. https://doi.org/10.1016/j.cie.2022.108200
32. J. Mou, P. Duan, L. Gao, X. Liu, J. Li, An effective hybrid collaborative algorithm for energy-efficient distributed permutation flow-shop inverse scheduling, *Future Gener. Comput. Syst.*, **128** (2022), 521–537. https://doi.org/10.1016/j.future.2021.10.003
33. Y. Pan, K. Gao, Z. Li, N. Wu, Solving biobjective distributed flow-shop scheduling problems with lot-streaming using an improved Jaya algorithm, *IEEE T. Cybern.*, **53** (2022), 3818–3828. https://doi.org/10.1109/TCYB.2022.3164165
34. S. Wang, L. Wang, An estimation of distribution algorithm-based memetic algorithm for the distributed assembly permutation flow-shop scheduling problem, *IEEE Trans. Syst. Man Cybernet.: Syst.*, **46** (2016), 139–149. https://doi.org/10.1109/TSMC.2015.2416127
35. J. Deng, L. Wang, S. Wang, X. Zheng, A competitive memetic algorithm for the distributed two-stage assembly flow-shop scheduling problem, *Int. J. Prod. Res.*, **54** (2016), 3561–3577. https://doi.org/10.1080/00207543.2015.1084063

36. J. Lin, Z. Wang, X. Li, A backtracking search hyper-heuristic for the distributed assembly flow-shop scheduling problem, *Swarm Evol. Comput.*, **36** (2017), 124–135. https://doi.org/10.1016/j.swevo.2017.04.007
37. J. Lin, S. Zhang, An effective hybrid biogeography-based optimization algorithm for the distributed assembly permutation flow-shop scheduling problem, *Comput. Ind. Eng.*, **97** (2016), 128–136. https://doi.org/10.1016/j.cie.2016.05.005
38. Z. Q. Zhang, B. Qian, R. Hu, H. P. Jin, L. Wang, A matrix-cube-based estimation of distribution algorithm for the distributed assembly permutation flow-shop scheduling problem, *Swarm Evol. Comput.*, **60** (2021). https://doi.org/10.1016/j.swevo.2020.100785
39. D. Ferone, S. Hatami, E. M. Gonzalez-Neira, A. A. Juan, P. Festa, A biased-randomized iterated local search for the distributed assembly permutation flow-shop problem, *Int. Trans. Oper. Res.*, **27** (2020), 1368–1391. https://doi.org/10.1111/itor.12719
40. H. Sang, Q. Pan, J. Li, P. Wang, Y. Han, K. Gao, et al., Effective invasive weed optimization algorithms for distributed assembly permutation flowshop problem with total flowtime criterion, *Swarm Evol. Comput.*, **44** (2019), 64–73. https://doi.org/10.1016/j.swevo.2018.12.001
41. Y. Huang, Q. Pan, J. Huang, P. N. Suganthan, L. Gao, An improved iterated greedy algorithm for the distributed assembly permutation flowshop scheduling problem, *Comput. Ind. Eng.*, **152** (2021), 107021. https://doi.org/10.1016/j.cie.2020.107021
42. Y. H. Yang, X. Li, A knowledge-driven constructive heuristic algorithm for the distributed assembly blocking flow shop scheduling problem, *Expert Syst. Appl.*, **202** (2022), 117269. https://doi.org/10.1016/j.eswa.2022.117269
43. Z. S. Shao, W. S. Shao, D. C. Pi, Effective constructive heuristic and metaheuristic for the distributed assembly blocking flow-shop scheduling problem, *Appl. Intell.*, **50** (2020), 4647–4669. https://doi.org/10.1007/s10489-020-01809-x
44. F. Zhao, L. Zhang, J. Cao, J. Tang, A cooperative water wave optimization algorithm with reinforcement learning for the distributed assembly no-idle flowshop scheduling problem, *Comput. Ind. Eng.*, **153** (2021), 107082. https://doi.org/10.1016/j.cie.2020.107082
45. H. B. Song, J. Lin, A genetic programming hyper-heuristic for the distributed assembly permutation flow-shop scheduling problem with sequence dependent setup times, *Swarm Evol. Comput.*, **60** (2021), 100807. https://doi.org/10.1016/j.swevo.2020.100807
46. F. Q. Zhao, J. L. Zhao, L. Wang, J. X. Tang, An optimal block knowledge driven backtracking search algorithm for distributed assembly No-wait flow shop scheduling problem, *Appl. Soft Comput.*, **112** (2021), 107750. https://doi.org/10.1016/j.asoc.2021.107750
47. J. C. Cai, D. M. Lei, J. Wang, L. Wang, A novel shuffled frog-leaping algorithm with reinforcement learning for distributed assembly hybrid flow shop scheduling, *Int. J. Prod. Res.*, (2022), 1233–1251. https://doi.org/10.1080/00207543.2022.2031331
48. M. Li, B. Su, D. M. Lei, A novel imperialist competitive algorithm for fuzzy distributed assembly flow shop scheduling, *J. Intell. Fuzzy Syst.*, **40** (2021), 4545–4561. https://doi.org/10.3233/JIFS-201391
49. J. G. Zheng, Y. L. Wang, A hybrid bat algorithm for solving the three-stage distributed assembly permutation flowshop scheduling problem, *Appl. Sci.*, **11** (2021), 10102. https://doi.org/10.3390/app112110102

50. F. Q. Zhao, X. T. Hu, L. Wang, T. P. Xu, N. N. Zhu, N. N. Zhu, A reinforcement learning-driven brain storm optimisation algorithm for multi-objective energy-efficient distributed assembly no-wait flow shop scheduling problem, *Int. J. Prod. Res.*, **61** (2022), 2854–2872. https://doi.org/10.1080/00207543.2022.2070786
51. Z. Q. Zhang, R. Hu, B. Qian, H. P. Jin, L. Wang, J. B. Yang, A matrix cube-based estimation of distribution algorithm for the energy-efficient distributed assembly permutation flow-shop scheduling problem, *Expert Syst. Appl.*, **194** (2022), 116484. https://doi.org/10.1016/j.eswa.2021.116484
52. Y. Huang, Q. Pan, L. Gao, Z. Miao, C. Peng, A two-phase evolutionary algorithm for multi-objective distributed assembly permutation flowshop scheduling problem, *Swarm Evol. Comput.*, **74** (2022), 101128. https://doi.org/10.1016/j.swevo.2022.101128
53. G. H. Zhang, K. Y. Xing, F. Cao, Scheduling distributed flowshops with flexible assembly and set-up time to minimise makespan, *Int. J. Prod. Res.*, **56** (2018), 3226–3244. https://doi.org/10.1080/00207543.2017.1401241
54. K. Ying, P. Pourhejazy, C. Cheng, R. Syu, Supply chain-oriented permutation flowshop scheduling considering flexible assembly and setup times, *Int. J. Prod. Res.*, **61** (2020), 258–281. https://doi.org/10.1080/00207543.2020.1842938
55. G. H. Zhang, K. Y. Xing, G. Y. Zhang, Z. X. He, Memetic algorithm with meta-Lamarckian learning and simplex search for distributed flexible assembly permutation flowshop scheduling problem, *IEEE Access*, **8** (2020), 96115–96128. https://doi.org/10.1109/ACCESS.2020.2996305
56. S. Yang, Z. Xu, The distributed assembly permutation flowshop scheduling problem with flexible assembly and batch delivery, *Int. J. Prod. Res.*, **59** (2021), 4053–4071. https://doi.org/10.1080/00207543.2020.1757174
57. Z. Zhao, M. Zhou, S. Liu, Iterated greedy algorithms for flow-shop scheduling problems: A tutorial, *IEEE Trans. Autom. Sci. Eng.*, **19** (2022), 1941–1959. https://doi.org/10.1109/TASE.2021.3062994
58. S. Hatami, R. Ruiz, C. Andrés-Romano, Heuristics and metaheuristics for the distributed assembly permutation flowshop scheduling problem with sequence dependent setup times, *Int. J. Prod. Econ.*, **169** (2015), 76–88. https://doi.org/10.1016/j.ijpe.2015.07.027

AIMS Press