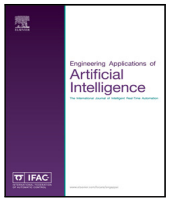
Engineering Applications of Artificial Intelligence 116 (2022) 105418

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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



A self-learning hyper-heuristic for the distributed assembly blocking flow shop scheduling problem with total flowtime criterion

Fuqing Zhao [a,*], Shilu Di [a], Ling Wang [b], Tianpeng Xu [a], Ningning Zhu [a], Jonrinaldi [c]

[a] School of Computer and Communication, Lanzhou University of Technology, Lanzhou 730050, China
[b] Department of Automation, Tsinghua University, Beijing 10084, China
[c] Department of Industrial Engineering, Universitas Andalas, Padang 25163, Indonesia

ARTICLE INFO

Keywords:
Distributed assembly
Blocking flow shop scheduling
Total flowtime
Constructive heuristic
Hyper-heuristic
Problem-specific knowledge

ABSTRACT

The distributed assembly blocking flow shop scheduling problem, which is a significant scenario in modern supply chains and manufacturing systems, has attracted significant attention from researchers and practitioners. To formulate the problem, a mixed-integer linear programming model is introduced to optimize the total flowtime. A constructive heuristic (HHNRa) and a self-learning hyper-heuristic (SLHH) are proposed to address the scheduling problem. HHNRa is designed based on the problem-specific knowledge to obtain initial solutions with high quality. A self-learning high-level strategy based on the historical success rate of low-level heuristics is presented to manipulate the low-level heuristics to operate in the solution space. In addition, a restart scheme with three distinct constructive heuristics is utilized to maintain the diversity of the solution. Based on 900 small-scale benchmark instances and 810 large-scale benchmark instances, comprehensive numerical experiments are conducted to evaluate the performance of the proposed SLHH algorithm. The results of the statistical analysis indicate that the proposed self-learning hyper-heuristic is superior to the compared state-of-the-art algorithms for the problem under consideration. Consequently, the proposed constructive heuristic and the self-learning hyper-heuristic are effective methods for the distributed assembly blocking flow shop scheduling problem.

1. Introduction

With the expansion of the global economy, market competition is intensifying, and more businesses are pursuing profit maximization (Jq et al., 2020). Current manufacturing industry market responsiveness is incompatible with economic growth, technological progress, and customers' individualized needs (Costa et al., 2020). As a result of the current economic trend toward customized production, assembly production is becoming more prevalent. Typically, assembly production consists of two distinct stages: production and assembly. As a subset of scheduling decisions, concurrent manufacturing determines which parts/components/subsets of products or services must be produced in parallel and then assembled in a final stage (Pan et al., 2019b). The assembly scheduling problems have applications in engine assembly (Lee et al., 1993), and personal computer manufacturing (Potts et al., 1995). As an extension of the distributed blocking flow shop scheduling problem (DBFSP) (Ribas et al., 2019, 2017; Zhao et al., 2020b), the distributed assembly blocking flow shop scheduling problem is gaining increasing attention from academics and industry professionals.

The DABFSP combines the distributed blocking flow shop scheduling problems (DBFSP) with assembly problems. Two stages comprise the DABFSP: production and assembly. The production phase consists of multiple identical blocking flow shops. In the assembly stage, there exists a single assembly machine, the jobs processed in the production factory are transferred to the assembly factory for finishing the assembly process. The blocking flow shop scheduling problem (BFSP) in the distributed environment has been proved to be an NP-hard problem (Shao et al., 2020b). Therefore, the DBFSP with the assembly stage is an NP-hard problem as well.

Due to the computational complexity of exact methods, such as mathematical methods and branch and bound, is difficult to solve large-scale problems with them (Zhao et al., 2020a). Although heuristic methods construct scheduling solutions rapidly, they lacks accuracy (He et al., 2021). On the other hand, it is difficult to design an effective heuristic method for complex scheduling problems when the characteristics of the problem are unknown. Metaheuristic algorithms have achieved satisfactory performance on large-scale problems, but the iterative process is time-consuming (Cai et al., 2020; Deng et al., 2020). In addition, numerous algorithms rarely adjust their search behavior based on historical data. Hyper-heuristic has proven to be an

* Corresponding author.
E-mail addresses: Fzhao2000@hotmail.com (F. Zhao), 1373590969@qq.com (S. Di), w.angling@tsinghua.edu.cn (L. Wang), x.utp@lut.edu.cn (T. Xu), 307516638@qq.com (N. Zhu), j.onrinaldi@eng.unand.ac.id (Jonrinaldi).

https://doi.org/10.1016/j.engappai.2022.105418
Received 5 January 2022; Received in revised form 15 May 2022; Accepted 30 August 2022
Available online xxxx
0952-1976/

efficient search technique for obtaining near-optimal solutions to complex optimization problems by controlling other heuristics (Drake et al., 2020). In this study, the selection of the low-level heuristic is guided by historical data. In addition, a heuristic method that takes the knowledge of DABFSP into account is presented. The main contributions are summarized as follows:

- The distributed assembly blocking flow shop scheduling problem is formulated using a mixed-integer linear programming model in which the total flowtime criterion is optimized. The problem-specific knowledge hidden in DABFSP is extracted to obtain initial solutions with high quality.
- A self-learning high-level strategy based on the historical success rate of low-level heuristics is presented to manipulate the low-level heuristics so that they can operate in solution space. The local search in the solution space is performed by seven LLHs based on job insertion and job swap within or between factories.
- A restart strategy with three distinct initial strategies is presented to ensure the diversity of solutions and explore the search space more effectively.

The remainder of the paper is organized as follows. The related literature is reviewed in Section 2. The DABFSP is described and the mixed-integer linear programming (MILP) model of DABFSP is explained in Section 3. The description of the SLHH algorithm is provided in Section 4. The experiment and analysis are implemented in Section 5. Finally, the conclusion and future work are summarized in Section 6.

2. Literature review

Flow shop scheduling problems (FSP), which are prevalent in real-world engineering situations, are popular subject of research (Brum et al., 2022; Morais et al., 2022; Safari et al., 2022). However, the conventional centralized production model does not satisfy enterprise and market demands for distributed manufacturing systems. Progressively, multi-regional and multi-factory cooperative production has come to dominate the international production landscape (Wang and Wang, 2020). Distributed production, as opposed to single-factory production, utilizes the resources of multiple manufacturing enterprises or factories to their maximum capacity. Manufacturing companies use the intelligent distributed production model to maximize production efficiency. In PFSP, Naderi and Ruiz (2010) first expanded the multi-factory environment. In this paper, six distinct MILP models were developed and analyzed. In addition, two factory allocation rules and fourteen heuristics involving different dispatching rules are presented. Based on this ground-breaking work, various solutions to distributed permutation flop shop scheduling problems were presented (DPFSP). Certain effective heuristics and metaheuristics were proposed to address DPFSP with the objective of total flowtime optimization (Pan et al., 2019a). In addition, problem-specific knowledge was explored and accelerations for evaluating neighborhood solutions were considered. To address DPFSP, an iterated greedy (IG) algorithm was presented (Jing et al., 2020). To improve the algorithm's search capability across the entire search space, IG incorporates a destruction operator with a dynamic damage degree. A sequence-dependent setup time (SDST)-aware greedy iterative method for addressing DPFSP is presented (Huang et al., 2020). A restart scheme was also implemented to prevent the IG algorithm from engaging in local optimizations. A distributed hybrid flop shop scheduling problem (DHFSP) comprising the parallel machine scheduling and the property of the distributed flow shop scheduling problem (DFSP) was investigated (Shao et al., 2020). To address the DHFSP, an iterated greedy algorithm with multi-search construction was also presented. An energy-efficient FSP with heterogeneous factories was addressed, taking into account the actual situation of the Chinese automobile industry (Lu et al., 2020). A cooperative coevolution algorithm, which cooperated iterated greedy and estimation of distribution, was proposed to solve multi-objective fuzzy DHFSP (Zheng et al., 2020).

In BFSP, there are no intermediate buffers between adjacent machines, as opposed to the previous production conditions. The blocking constraints are widely in actual production scenarios, such as steel manufacturing (Aqil and Allali, 2021). The presence of blocking increases job wait times, thereby decreasing the job sequence's processing efficiency. In addition, blocking constraints contribute to the waste of energy. For example, the hybrid flow shop scheduling problem with energy-efficient criteria and blocking constraints was investigated (Qin et al., 2022). As a result of the blocking constraint, job completion times are prolonged, and resources are wasted. Therefore, applying distributed flow shop scheduling problems with blocking constraints is substantial. DBFSP was initially proposed in published works (Ribas et al., 2017). In addition, heuristic and metaheuristic methods were presented. Based on this mathematical model, various solutions for DBFSP have been proposed. For instance, a discrete differential evolution (DDE) algorithm was presented to address the DBFSP (Zhang et al., 2018). In the interim, a local search method and an elitist retain strategy are incorporated into the DDE framework to preserve the equilibrium between local exploitation and global exploration. For DBFSP, a discrete fly fruit optimization has been proposed (Shao et al., 2020a). DBFSP accounted for the uncertainty of the manufacturing system and the processing time (Shao et al., 2020c).

Proper production management in a complex production control environment can increase machine utilization and decrease makespan (Chen and Zhao, 2021). In hybrid flow shop scheduling problem (HFSP) production management, particular constraints, including multi-period control and job transport time, were considered. A multi-objective optimization model for the HFSP has been developed, and a faster and more precise genetic algorithm for solving the model is proposed. Improving the manufacturing facility's global competitiveness necessitates effective energy-aware operations management (Cui and Lu, 2021). A mathematical model is proposed that incorporates three interdependent operational aspects, including production, maintenance, and energy, for flow shops operating under the Time-of-Use electricity tariff. The trade-off between energy cost and makespan demonstrates that instances with a later production deadline can generate a greater profit using the model described previously. The production management of an assembly flow shop has substantial practical application in this study. Customers must receive a complete product rather than a collection of scattered accessories. In other words, the manufacturer not only processes jobs but also assembles the processed jobs into the final product. Therefore, consideration of the assembly procedure in a distributed manufacturing system is of great research value.

The distributed assembly permutation flow shop scheduling problem (DAPFSP) was introduced by Hatami et al. (2013), who also developed the DAPFSP MILP. In addition, several positive heuristics and variable neighborhood descent strategies are presented. Diverse strategies to combat DAPFSP have been proposed. A hybrid biogeography-based optimization algorithm (HBBO) was presented for DAPFSP with the makespan criterion (Lin and Zhang, 2016). Path relinking was used to optimize the sequence of product assembly in HBBO. The insertion heuristic was then used to establish the job order within the product. In addition, a local search method that takes the characteristics of the problem into account was developed to improve the quality of the most promising individual. On 910 small instances and 810 large instances, the HBBO algorithm's efficacy was validated. DAPFSP was solved using a matrix-cube-based estimation of the distribution algorithm with the goal of minimizing the maximum completion time (Zhang et al., 2021). In the algorithm, the matrix cube learned valuable information from elites. In addition, a sampling system was implemented to evaluate the distribution of superior solutions. A problem-dependent VNS for the local search was also introduced. To address DAPFSP, a memetic algorithm based on an estimation of distribution algorithm (EDAMA) was proposed (Wang and Wang, 2016). EDAMA incorporated EDA-based exploration and local search-based exploitation into the MA framework. Additionally, a mechanism for selectively enhancing sampling and a

critical path-based local search static were implemented to enhance the search capability. Introduced was a backtracking search hyper-heuristic (BS-HH) algorithm for DAPFSP (Lin et al., 2017). Certain BS-HH operators were intended to be LLHs. A backtracking search was also regarded as a high-level heuristic (HLH) for manipulating these low-level heuristics (LLHs). A genetic programming hyper-heuristic (GPHH) was utilized to address DAPFSP using SDST. In GPHH, genetic programming was regarded as an HLH for producing LLH sequences (Song and Lin, 2021). Three discrete invasive optimizations (DIWO) algorithms were presented to address the DAPFSP total flowtime criterion. Local search strategies for product permutations and job sequences were developed in DIWO (Sang et al., 2019). Shao et al. (2019) presented for the first time the distributed assembly no-idle flow shop scheduling problem (DANIFSP). Based on the MILP model of the DANIFSP, a cooperative algorithm with reinforcement learning was proposed (Zhao et al., 2021) In response to the DANIFSP, an improved ILS and VNS were presented. The job assignment rule took into account production and assembly completion times. In addition, four neighborhood search techniques were incorporated with the accelerated search technique. A water wave optimization algorithm with problem-specific knowledge (KWWO) was proposed to address DABFSP (Zhao et al., 2022). Using a constructive heuristic (KBNEH) that combined a dispatch rule with an insertion-based procedure, an initial solution of high quality was generated. To enhance the performance of KWWO, the destruction–construction operator, four local search methods within the framework of the VNS, and path-relining were implemented in three distinct stages. On 900 small-scale and 810 large-scale instances, the efficiency of the proposed algorithm is finally validated. Widespread use of machine learning and metaheuristics to solve practical mechanical and aerospace engineering problems (Khatir et al., 2021; Nguyen-Le et al., 2020; Wang et al., 2021). It was proposed to detect damage in laminated composite structures using an artificial neural network (ANN) based on a hybrid metaheuristic optimization algorithm (Tran-Ngoc et al., 2021). Plate cracks were identified using an enhanced ANN technique coupled with a Java algorithm (Khatir et al., 2020). A novel machine-learning method based on global search techniques and vectorized data is utilized to detect structural damage (Tran-Ngoc et al., 2020). These studies have inspired the creation of promising machine learning techniques for flow shop scheduling.

Hyper-heuristic has proven to be an effective solution for flow shop scheduling problems (Almeida et al., 2020; Heger and Voss, 2021). Instead of searching the solution space directly, hyper-heuristic uses a collection of low-level heuristics (LLHs) (Dong et al., 2015). As a result, a significant number of researchers are devoted to creating automated algorithms to manage low-level heuristics. The online learning methods incorporate hyper-heuristic to process feedback during the search procedure, and this feedback influences subsequent decisions at the hyper-heuristic level (Drake et al., 2020). In addition, historical data is required for feedback formation and plays a crucial role in hyper-heuristic online learning-based algorithms.

In recent years, the distributed production scheduling with the assembly process has been studied due to its applicability, as indicated by the literature mentioned above review. Massive heuristics and metaheuristics are proposed as scheduling problem solutions. However, there is only one paper on DABFSP available (Shao et al., 2020b). This article introduced a self-learning hyper-heuristic for DABFSP, with total flowtime as the optimization metric.

Several points reviewed in the preceding sections are of the utmost importance. The motivation of this paper is summarized as follows. (1) In recent years, the distributed assembly permutation flow shop scheduling problem (DAPFSP) and its variants have attracted considerable research interest. However, more realistic constraints must be considered to make the problem applicable to real-world manufacturing systems. The distributed assembly blocking flow shop scheduling problem (DABPSP) is an extension of the distributed assembly flow shop scheduling problem (DAPFDP) that incorporates a blocking constraint. Practical production scenarios where the blocking constraint was prevalent included the heating and rolling of steel (Zhao et al., 2020b), the processing of cider (Merchan and Maravelias, 2016), and the production of chemicals (Riahi et al., 2019, 2017). (2) The DBPFSP with the total flowtime criterion and the DBPFSP with the makespan criterion represent two distinct mathematical problems. Most previous work for the DFSP with the makespan criterion could not guarantee highly competitive results for the DFSP with the total flowtime criterion. In addition, the practical implication is clear: minimizing total flowtime can reduce work in progress and achieve stable resource utilization (Pan and Dong, 2014; Pan and Ruiz, 2013; Sang et al., 2019). Since its inception, hyper-heuristic has been shown to be an effective algorithm for solving various problems. Given the unique framework of hyper-heuristic, the high-level heuristic can be easily adapted to modify the algorithm. This paper proposes a self-learning hyper-heuristic to minimize the total flowtime criterion when solving the DABFSP.

3. Problem statement

In this section, the notations and meanings utilized in this article are listed.

n	The number of jobs.
m	The number of machines in each factory.
F	The number of factories.
S	The number of products.
i	Index of the jobs, where $i \in \{1,2,\ldots,n\}$.
j	Index of the machines, where $j \in \{1,2,\ldots,m\}$.
f	Index of factories, where $f \in \{1,2,\ldots,F\}$.
k	Index of the job position in a given sequence, where $k \in \{1,2,\ldots,n\}$.
h	Index of the products, where $h \in \{1,2,\ldots,S\}$.
J	The set of n jobs to be processed, $J = \{J_1, J_2, \ldots, J_n\}$.
M	The set of m machines, $M = \{M_1, M_2, \ldots, M_m\}$.
Fa	The set of F parallel factories, $Fa = \{Fa_1, Fa_2, \ldots, Fa_F\}$.
P	The set of S products, $P = \{P_1, P_2, \ldots, P_s\}$.
$o_{i,j}$	The operation of the job J_i on the machine M_j.
$p_{i,j}$	The processing time of $o_{i,j}$.
π	Feasible scheduling sequence in processing factory, $\pi = [\pi^1, \pi^2, \ldots, \pi^F]$.
φ	The assembly sequence of products in the assembly factory, $\varphi = [\varphi(1), \varphi(2), \ldots, \varphi(S)]$.
$\varphi(h)$	The hth product to be assembled.
J_P	The sequence of the last processed job of all products, $J_P = [J_P(1), J_P(2), \ldots, J_P(S)]$.
$J_P(h)$	The last processed job belonging to the product P_h, $h \in \{1,2,\ldots,S\}$.
Π	A feasible scheduling solution.
π^f	The processing sequence in the factory f, $\pi^f = [\pi^f(1), \pi^f(2), \ldots, \pi^f(n^f)]$.
n^f	The number of jobs in the factory f.
n_h	The number of jobs belonging to the product P_h.
n_h^f	The number of jobs belonging to the product P_h in the factory f.
L	A fairly large positive integer.
$D_{f,k,j}$	The departure time of the job in position k on machine M_j at factory f.
t_h	The assembly time of the product P_h.
$TF(\Pi)$	The maximum assembly completion time of a feasible scheduling solution Π.
D_i	The departure time of the job J_i on machine M_m.
D_h^P	The departure time of the last processed job of the product P_h.

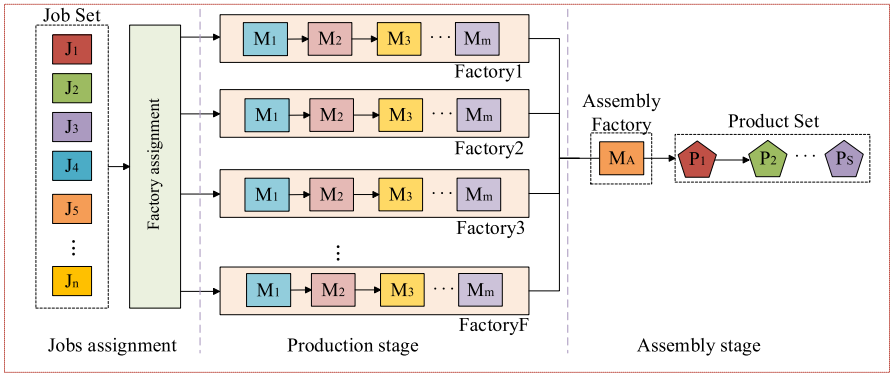


Fig. 1. Schematic diagram of the DABFSP.

DA_h	The assembly completion time of the product P_h in the assembly stage.
$G_{i,h}$	The binary variable that takes value 1 if the job J_i belongs to product P_h, otherwise 0.
$X_{f,k,i}$	The binary variable that takes value 1 if the job J_i occupies position k in factory f, otherwise 0.

There are two stages to the DABFSP: production and assembly. As depicted in Fig. 1, there exists a set of parallel distributed factories in the production stage, with each factory containing an identical blocking flow shop with m machines. The production stage of DABFSP is described as follows. There are n jobs $\{J_1, J_2, \ldots, J_n\}$ need to be allocated to F factories for processing. For each factory, there exists a BFSP, and each job J_i has to be processed regarding the identical order of the machines. The processing time of the job J_i on machine M_j is recorded as $p_{i,j}$. There is no buffer between adjacent machines for each shop that blocks the flow. In other words, if the next machine is unavailable, the task is held on the current machine until the next machine becomes available. Once a job has been assigned to a particular factory, it cannot be transferred to another factory for processing. Additionally, the following constraints must be met: Each job must be completed on a single machine at a predetermined time. Each machine must perform one task at a time. At time zero, all jobs are available for processing. No preemptive scheduling is permitted. Once an operation has begun, it cannot be interrupted until it has been completed. Transportation time between operations and machine setup time are factored into the processing time.

There is only one assembly machine in the assembly stage, and S products must be finished assembling in this process. Each product is composed of distinct jobs. A product is started to assembly only when all the jobs belonging to that product are finished. Simultaneously, it must be ensured that the assembly machine is idle. The subsequent product is not assembled until all previous product-related tasks have been completed. Consequently, the solution to the DABFSP lies in determining the allocation of jobs to factories, the processing sequence of jobs within each factory, and the assembly order of products.

Utilizing mathematical techniques to model practical application issues is a typical modeling technique. Many studies effectively use the mixed-integer linear programming model of DBFSP to reduce the difficulty of solving distributed blocking flow shop scheduling problems (Ribas et al., 2019, 2017; Zhao et al., 2020b). The purpose of optimization in this study is to maximize the total flowtime of all products. The MILP model of DABFSP is shown as follows:

Objective : $\min TF$ (1)

Subjectto :

$$\sum_{f=1}^{F}\sum_{k=1}^{n} X_{i,k,f} = 1, i \in \{1,2,\ldots,n\} \quad (2)$$

$$\sum_{i=1}^{n} X_{i,k,f} \le 1, \quad k \in \{1,2,\ldots,n\}, f \in \{1,2,\ldots,F\} \quad (3)$$

$$D_{f,0,j} \ge 0, \quad j \in \{1,2,\ldots,m\}, f \in \{1,2,\ldots,F\} \quad (4)$$

$$D_{f,k,0} \ge D_{f,k-1,1}, \quad k \in \{1,2,\ldots,n\}, f \in \{1,2,\ldots,F\} \quad (5)$$

$$D_{f,k,j} \ge D_{f,k,j-1} + \sum_{i=1}^{n} X_{i,k,f} \cdot P_{i,j}, \quad k \in \{1,2,\ldots,n\}, j \in \{1,2,\ldots,m\}, f \in \{1,2,\ldots,F\} \quad (6)$$

$$D_{f,k,j} \ge D_{f,k-1,j+1}, \quad k \in \{2,3,\ldots,n\}; j \in \{1,2,\ldots,m-1\}; f \in 1,2,\ldots,F \quad (7)$$

$$D_i \ge D_{f,k,m} - L \cdot \left(1 - X_{i,k,f}\right), \quad k \in \{1,2,\ldots,n\}, i \in \{1,2,\ldots,n\}, f \in \{1,2,\ldots,F\} \quad (8)$$

$$DA_h \ge DA_{h-1} + t_h, h \in \{1,2,\ldots,S\} \quad (9)$$

$$DA_h \ge D_i + t_h - L \cdot \left(1 - G_{i,h}\right), \quad i \in \{1,2,\ldots,n\}, h \in \{1,2,\ldots,S\} \quad (10)$$

$$TF \ge \sum_{h=1}^{S} DA_h, h \in \{1,2,\ldots,S\} \quad (11)$$

$$DA_h > 0, h \in \{1,2,\ldots,S\} \quad (12)$$

$$X_{i,k,f} \in \{0,1\}, \quad i \in \{1,2,\ldots,n\}, k \in \{1,2,\ldots,n\}, f \in \{1,2,\ldots,F\} \quad (13)$$

Constraint (1) indicates that the objective function is to minimize the TF. Constraint (2) ensures that each job is only assigned to a single factory and a single position within that factory. Constraint (3) ensures that each factory position is occupied by only one job. The initial condition is stipulated by constraint (4). In each factory, the starting time of a job on machine one is indicated by constraint (5). The relationship between the departure times of two adjacent operations for the same job in each factory is indicated by constraint (6). The blocking constraint is satisfied when the relationship between the departure times of two adjacent jobs in each factory is represented by constraint (7). Constraint (8) specifies the duration of each task. Constraint (9) specifies the relationship between adjacent assembly products. DA_0 is defined as the starting assembly time of the first assembly product. Constraint (10) ensures that no product is assembled until all of its jobs have been completed. The calculation of objective function TF is represented by constraint (11). The assembly completion time of each product is guaranteed to be non-negative by constraint (12). Constraint (13) defines the decision variables.

The calculation of objective function TF for the DABFSP consists of two stages. In the first stage, the complete time of all jobs is

Table 1
Data for the example of the DABFSP.

Product	Job	Processing time		Assembly time
		M_1	M_2	
P_1	1	19	98	226
	2	23	35	
	4	77	10	
	8	21	89	
P_2	3	56	12	130
	5	51	16	
	6	45	37	
	7	18	48	

computed via Eqs. (14)–(17). Let $\pi = [\pi^1, \pi^2, \ldots, \pi^f, \ldots, \pi^F]$ indicate a feasible scheduling sequence of the production factory, where $\pi^f = \left[\pi^f(1), \pi^f(2), \ldots, \pi^f\left(n^f\right)\right]$ indicates the job sequence of factory f. The n^f represents the number of jobs for factory f. The $p_{i,j}$ represents the processing time of the job J_i on machine M_j. The $D_{f,k,j}$ represents the departure time of a job in position k on machine M_j at factory f. In the second stage, the assembly order is determined based on the total duration of jobs. First, the minimum completion time of the last processed job for all products is determined as the initial time for the first product assembly. Then, the value of objective function TF is calculated via Eqs. (18)–(21)

$$D_{f,1,0} = 0, f \in \{1, 2, \ldots, F\} \tag{14}$$

$$D_{f,1,j} = D_{f,1,j-1} + p_{i,j}, \quad i \in \{1, 2, \ldots, n\}, j \in \{1, 2, \ldots, m\}, f \in \{1, 2, \ldots, F\} \tag{15}$$

$$D_{f,k,0} = D_{f,k-1,1}, \quad k \in \left\{2, 3, \ldots, n^f\right\}, f \in \{1, 2, \ldots, F\} \tag{16}$$

$$D_{f,k,j} = \max\left\{\begin{matrix} D_{f,k,j-1} + p_{i,j}, \\ D_{f,k-1,j+1} \end{matrix}\right\}, \quad i \in \{1, 2, \ldots, n\}, k \in \left\{2, 3, \ldots, n^f\right\}, j \in \{1, 2, \ldots, m-1\}, f \in \{1, 2, \ldots, F\} \tag{17}$$

$$D_{f,k,m} = D_{f,k,m-1} + p_{i,j} \quad i \in \{1, 2, \ldots, n\}, k \in \left\{2, 3, \ldots, n^f\right\}, f \in \{1, 2, \ldots, F\} \tag{18}$$

$$DA_1 = D_1^P + t_1 \tag{19}$$

$$DA_h = \max\left\{DA_{h-1}, D_h^P\right\} + t_h \quad h \in \{2, 3, \ldots, S\} \tag{20}$$

$$TF = \sum_{h=1}^{S} DA_h \tag{21}$$

Table 1 provides an example of the DABFSP with eight jobs, two factories, two products, and two machines. The scheduling Gantt chart is shown in Fig. 2. On the Gantt chart, the tasks for the same product are indicated by the same color. Jobs 6, 5, 1, and 4 are assigned to plant 1. Tasks 7, 3, 2, and 8 are assigned to factory 2. Product 2 assembly begins when Job 5 is processed. After product 2 has been assembled, job 4 has already been completed. Therefore, product 1 assembly begins simultaneously with product 2 assembly. The value of the TF is $242 + 468 = 710$.

4. Description of the SLHH algorithm

4.1. Solution representation

The DABFSP is divided into the stages of processing and assembly. n jobs are assigned to F factories during the processing stage. At the stage of assembly, there are S products to be assembled. To intuitively represent a solution, the scheduling sequence of jobs in the processing factory and the assembly sequence of products in the assembly stage are incorporated into the solution's representation. The sequences of jobs in F factories are recorded as $\pi = \left\{\pi^1, \pi^2, \ldots, \pi^F\right\}$. The assembly sequence of S products is recorded as $\varphi = \{\varphi(1), \varphi(2), \ldots, \varphi(S)\}$, where $\varphi(h)$ represents a specific product. Thus, a complete feasible solution is denoted as $\Pi = (\pi, \varphi)$. An example with eight jobs, two machines, two factories, and two products is given to illustrate the encoding mechanism.

$$\Pi = (\pi, \varphi) = \left(\begin{bmatrix} 6 & 5 & 1 & 4 \\ 7 & 3 & 2 & 8 \end{bmatrix}, \begin{bmatrix} 2 & 1 \end{bmatrix}\right) \tag{22}$$

where jobs 6, 5, 1, and 4 are allocated to factory 1 regarding the sequence $6 \rightarrow 5 \rightarrow 1 \rightarrow 4$, while the remaining jobs are allocated to factory 2 regarding the sequence $7 \rightarrow 3 \rightarrow 2 \rightarrow 8$. The assembly sequence of products is $2 \rightarrow 1$.

4.2. Constructive heuristic

It is necessary to analyze the DAFSP's characteristics to formulate efficient algorithms based on the problem's knowledge (Sang et al., 2019). Consequently, the characteristics of DABFSP are investigated. In DABFSP, the production process of the jobs and the product assembly process determine the final product sequence. In addition, as soon as all jobs for the same product have been processed in the production factories, the assembly phase begins immediately. Inspired by the above characteristics, three kinds of knowledge of DABFSP are extracted, and the quantitative representations of the knowledge are described as follows.

Knowledge 1: In each production factory, all jobs for the same product are assigned together so that the assembly of that product can begin as quickly as possible. The quantitative representation is shown as follows:

$$x_{u,k,f} + x_{v,k+1,f} = 2, if\, J_u, J_v \in P_h \; n_h^f \geq 2 \tag{23}$$

The job J_u belonging to the product P_h occupies the position k in processing factory f, the value of the $x_{u,k,f}$ is 1. The job J_v belonging to the product P_h occupies the position $k+1$ in processing factory f, the value of the $x_{v,k+1,f}$ is 1. In other words, the jobs J_u and J_v of the product P_h are allocated together in the processing factory f. The Gantt chart for the knowledge 1 is shown in Fig. 3. In factory 2, job 2 is not assigned with job 8 for processing, which delays the starting of product 1 assembly.

Knowledge 2: Once all the jobs of the identical product are finished in the processing factories, the assembly stage is started at once. The quantitative representation is shown as follows:

$$J_P = \left\{J_P(1), J_P(2), \ldots, J_P(S)\right\} \tag{24}$$

$$G_{ii,h} = \begin{cases} 1 & if \quad J_P(ii) \in \varphi(h) \\ 0 & otherwise, \end{cases} \quad ii \in \{1, 2, \ldots, S\}, h \in (1, 2, \ldots, S) \tag{25}$$

$$\varphi = \varphi(1), \varphi(2), \ldots, \varphi(S) \tag{26}$$

where $J_P = \left\{J_P(1), J_P(2), \ldots, J_P(S)\right\}$ is the sequence of the last processed job of all products, the $J_P(h)$ is the last processed job belonging to the product P_h. The $\varphi = \varphi(1), \varphi(2), \ldots, \varphi(S)$ is the assembly sequence of products. The $\varphi(h)$ is the product h to be assembled. The Gantt chart for the knowledge is shown in Fig. 4. According to Fig. 4(b), job 2 is the final processing step for product 1, but product 1 is not immediately assembled. The assembly process cannot begin until the final job for product 2 has been completed, resulting in an increase in assembly completion time.

Knowledge 3: The jobs of the identical product should be distributed to multiple factories for parallel processing, accelerating the assembly start time for that product. The quantitative illustration appears as follows:

$$n_h^f \leq \lceil n_h / F \rceil \tag{27}$$

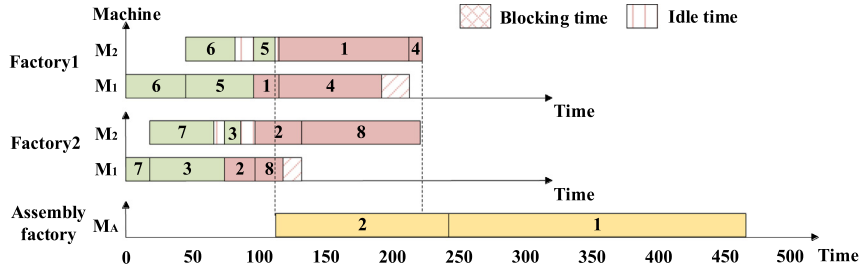


Fig. 2. Gantt chart of the example for the DABFSP.

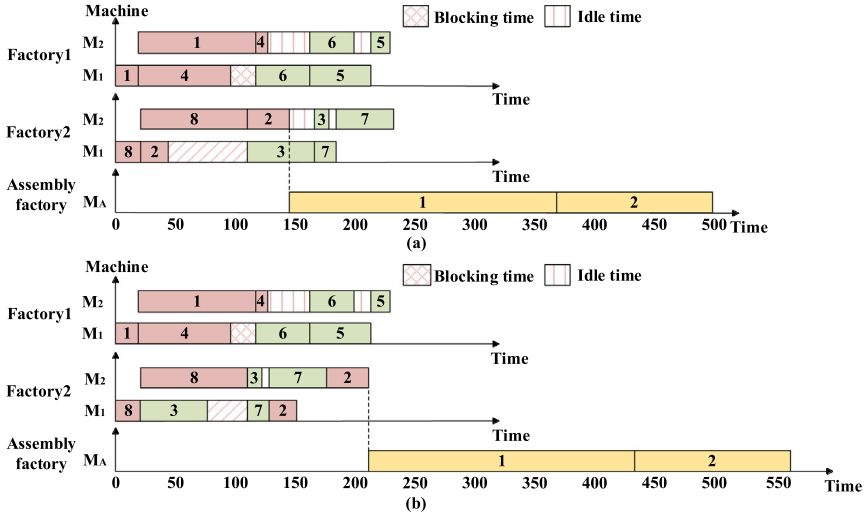


Fig. 3. Gantt chart for the Knowledge 1.

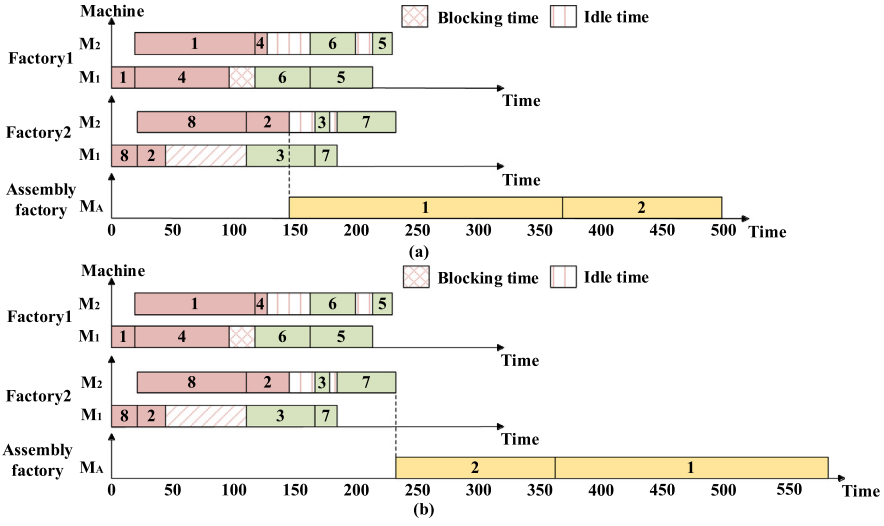


Fig. 4. Gantt chart for the Knowledge 2.

where n_h^f is the number of jobs belonging to the product P_h in processing factory f, n_h is the number of jobs composing product P_h. A straightforward example is provided in Fig. 5. In Fig. 5(b), the jobs associated with two products are assigned separately to two factories for parallel processing, reducing the start assembly time.

Typically, a construction heuristic employs some characteristics of the considered problem to rapidly generate a promising initial solution. To minimize the total flowtime of all products for the DABFSP, not only must the processing time of all jobs be minimized, but the assembly time must also be optimized. Consequently, DABFSP considers three issues: allocating jobs to appropriate factories, scheduling jobs in each processing factory, and scheduling products in the assembly factory. In DAFSP, the assembly stage has a significant impact on the scheduling of the processing stage. To expedite the assembly phase, three types of knowledge are utilized to construct an effective solution. This section constructs an initial job sequence based on prior knowledge. First, based on knowledge 1, the products are arranged in descending order according to the total processing time of the jobs associated with the same product. Consequently, the jobs associated with the same product are sorted in descending order by processing time. The NRa rule

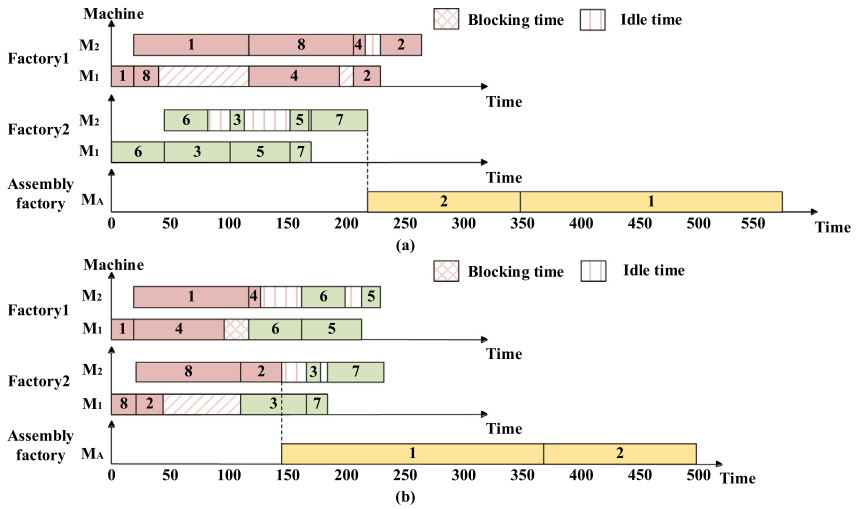


Fig. 5. Gantt chart for the Knowledge 3.

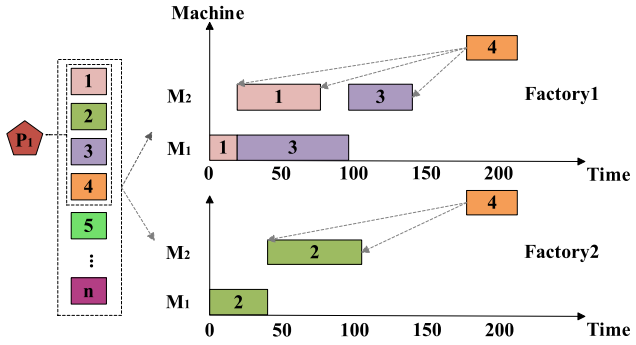


Fig. 6. Gantt chart of assignment principle NRa.

introduced by Shao et al. (2019) assigns jobs to each processing factory. The following job assignment rule is based on knowledge level 3. First, the first F jobs of the initial job sequence are selected and assigned to the first position of each factory in turn. Second, the remaining $n - f$ unscheduled jobs are allocated to factory 1 to f according to NRa rule. The Gantt chart of assignment principle NRa is shown in Fig. 6.

Specifically, the jobs in the unscheduled set are extracted in order. After inserting the job into all possible positions of all factories, the assembly time is computed. The position with the minimum total flowtime is chosen to insert a job. When computing the total flowtime, a special case must be considered. Since jobs are inserted into factories one by one, jobs that comprise a product can be incomplete during the calculation of the assembly stage. Therefore, the processing completion time of the last processed job among the currently completed jobs is considered the assembly start time for this product. The constructive heuristic is denoted by the letters HHNRa. Algorithm1 describes the HHNRa heuristic's procedure in detail.

4.3. SLHH

The SLHH algorithm consists of a high-level heuristic and a collection of LLH. Two essential components of a high-level heuristic are a self-learning heuristic selection mechanism and an acceptance criterion. The high-level heuristic selects an LLH from among the set of LLHs. The selected LLH is then applied to the existing solution to generate a new solution. It is determined whether to accept or reject the new solution based on the acceptance criterion (Kheiri and Özcan, 2016). To ensure the diversity of solutions, a restart scheme with three distinct initial strategies is presented as a conclusion. The specifics of each component of the SLHH algorithm are described in the sections that follow.

4.3.1. Heuristic selection mechanism

According to the basic framework of the hyper-heuristic, a heuristic selection mechanism is utilized to select appreciate LLH. The historical success rate of each LLH is summarized as knowledge to guide the algorithm for self-learning heuristic selection for the heuristic selection mechanism. Initializing the selection rate of each LLH to an infinitely large positive number ensures that each perturbation heuristic can be chosen. The historical success rate must be converted into knowledge that guides the self-learning selection of low-level heuristics. The definition of knowledge is (28) and (29). Finally, the low-level heuristics is selected according to sc_g.

$$sr_g\left(LLH_i\right) = sn_g\left(LLH_i\right)/t_g(LLH_i) \tag{28}$$

$$ss_g\left(LLH_i\right) = sr_g\left(LLH_i\right)/\sum_{i}^{N} sr_g\left(LLH_i\right) \tag{29}$$

where $t_g(LLH_i)$ is the elapsed time of LLH_i that applied to a solution during past g iterations and the sn_g is the success number of LLH_i. The sr_g is the knowledge. The ss_g is the selection rate. Typically, when the quality of the solution is enhanced, the LLH selection rate increases. The heuristic selection mechanism is described in detail in Algorithm 2.

4.3.2. Acceptance criterion

In general, a simple acceptance criterion of the hyper-heuristic is used to accept the enhanced solution. However, by utilizing a single acceptance criterion, the algorithm is susceptible to stagnation or premature convergence. In this study, a probability-based acceptance criterion is used to accept a potentially inferior solution, which is advantageous for assisting the algorithm in escaping local optima and ensuring diversity of search. The specifics are illustrated by Algorithm 3.

4.3.3. Restart scheme

When the perturbation stage and local search stage are executed many times, the effective search space of the current solution is constrained, so the quality of the solution does not improve with increasing iterations. Consequently, a restart scheme is proposed to investigate a more efficient search space. Three distinct operators are utilized to produce an initial solution. (1) Construct a preliminary solution using the heuristic H1NRa. (2) Construct an initial solution using the H2NRa heuristic. (3) Apply the best solution found so far as the initial solution. In Algorithm 4, the restart condition is described.

During the initialization phase, the generated job sequences are assigned to each factory according to the factory allocation rule. The initial solution is constructed using the knowledge extracted from the DABFSP's characteristics. The heuristic selection mechanism applies the

Algorithm 1 Constructive heuristic

Input: F, n
Output: TF, Π
Sort products in a descending order regarding the total processing time of the jobs belonging to the identical product.
Sort jobs within each product in descending order regarding the processing time of each job and an initial job sequence τ is finished.
for i = 1 **to** F **do**
  $\pi^f(1) \leftarrow \tau(i)$;
end
for $i = F + 1$ **to** n **do**
  $JobA \leftarrow \tau(i)$, $valueA \leftarrow$ an infinitely positive number;
  for f = 1 **to** F **do**
    for l = 1 **to** n^f+1 **do**
      $valueB \leftarrow$ Compute TF of Π when inserting $JobA$ to the kth position of factory f;
      if $valueB > valueA$ **then**
        $valueB \leftarrow valueA$, $bestFac \leftarrow f$, $bestPos \leftarrow l$;
      end if
    end for
  end for
  Insert $JobA$ to the $bestPos$-th position of factory $bestFac$;
end for
Return Π

Algorithm 2 Heuristic selection mechanism

Input: f_{current}, Π_{current}
Output: $f_{\text{candidate}}$, $\Pi_{\text{candidate}}$
Initialization probability array, selection rate of each LLH is initialized to an infinite positive number;
$LLH_i \leftarrow$ Select a low-level heuristic according to (29) from $[LLH_1, LLH_2, LLH_3, LLH_4]$;
$[f_{new}, \Pi_{new}] \leftarrow$ apply the LLH_i to the current solution;
$[LLH_5, LLH_6, LLH_7] \leftarrow$ Add local search heuristic to array arrayLc;
while arrayLc is not empty **do**
  $LLH_j \leftarrow$ Randomly select a local search heuristic from arrayLc;
  $[f_{\text{candidate}}, \Pi_{\text{candidate}}] \leftarrow$ Apply LLH_j on Π_{new};
  if $f_{\text{candidate}} < f_{new}$ **do**
    $f_{\text{new}} \leftarrow f_{\text{candidate}}$;
    $\Pi_{\text{new}} \leftarrow \Pi_{\text{candidate}}$;
    arraryLc $\leftarrow [LLH_5, LLH_6, LLH_7]$;
  else
    Remove the LLH_j form the arrayLc;
  end if
end while
$t_g(LLH_i) \leftarrow$ Record the time consumed by the selected LLH_i;
accept $\leftarrow$ Judge whether to accept the Π_{new} according to the Acceptance criterion (**Algorithm 3**)
if accept is true **do**
  $sn_g(LLH_i) \leftarrow sn_g(LLH_i) + 1$;
  $\Pi_{\text{current}} \leftarrow \Pi_{\text{candidate}}$;
  $f_{\text{current}} \leftarrow f_{\text{candidate}}$;
end
Update $ss_g(LLH_i)$ according to (28).

selected low-level heuristic to the current solution and generates a new solution. The local search strategy is then implemented to improve the search capacity. The selection of all low-level heuristics is a process of online learning that makes decisions at each step based on the current state of the solution. The inferior solution is accepted with a certain probability for the acceptance criterion to prevent the algorithm from

Algorithm 3 Acceptance criterion

Input: $f_{\text{candidate}}$, $\Pi_{\text{candidate}}$
Output: f_{current}, Π_{current}
if $f_{\text{candidate}} < f_{\text{current}}$ **then**
  $f_{\text{current}} \leftarrow f_{\text{candidate}}$;
  $\Pi_{\text{current}} \leftarrow \Pi_{\text{candidate}}$;
  $n_{\text{improve}} \leftarrow n_{\text{improve}} + 1$;
  $mu_{\text{improve}} \leftarrow mu_{\text{improve}} + (f_{\text{current}} - f_{\text{candidate}} - mu_{\text{improve}})/n_{\text{improve}}$;
else if rand $< exp((f_{\text{current}} - f_{\text{candidate}})/(0.5 * mu_{\text{improve}}))$
  $f_{\text{current}} \leftarrow f_{\text{candidate}}$;
  $\Pi_{\text{current}} \leftarrow \Pi_{\text{candidate}}$;
end if

Algorithm 4 Restart scheme

Input: f_{current}, $wait$, $wait_{\max}$, $t_{\max}$
Output: f_{current}
if $f_{\text{current}} < f_{\text{best}}$ **then**
  $f_{runbest} \leftarrow f_{current}$;
  $wait_{\max} \leftarrow max\{wait, wait_{\max}\}$;
  $wait \leftarrow 0$;
  if $f_{\text{runbest}} < f_{\text{current}}$ **then**
    $f_{\text{best}} \leftarrow f_{\text{runbest}}$;
    $t_{\text{best}} \leftarrow min\{cputime - t_{\text{start}}, t_{\text{best}}\}$;
  end if
else if
  $wait \leftarrow wait + 1$;
  $Patience \leftarrow wait_{\max} * t_{\max} /(cputime - t_{\text{start}})$;
  if $wait_{\max} > patience$ && $(t_{\max} - (cputime - t_{\text{start}})) \geq t_{\text{best}}$ **then**
    Restart and randomly select an operator to generate the initial solution.
  end if
end if

falling into local optima. Finally, a restart scheme comprising three distinct operators is implemented to explore a more efficient search space. Fig. 7 depicts the flowchart of the SLHH algorithm.

4.4. The low-level heuristics

The low-level heuristics have an impact on the performance of hyper-heuristic. Seven effective low-level heuristics are designed in this study. The job insertion (denoted as LLH_1) is utilized to perturb a solution. Procedure 1 involves selecting a job at random from a random factory. Afterward, the selected job is tentatively inserted into all possible positions of all factories. The position with the best TF is selected. The concrete details are described in Procedure 1. The time complexity of each low-level heuristic is given due to the large solution space. According to the computational method of objective function from Section 3, the time complexity of an insertion operation is evaluated as $O(m \times n)$. Therefore, the time complexity of performing a full insertion process is $O\left(m \times n^2 \times (n + F - 1)\right)$. The insertion neighborhood is one of the most effective methods for addressing scheduling problems in flop shops. An insert-neighborhood-based speed-up method (Wang et al., 2010) is proposed to decrease the time consumption. Based on the speed-up method, the time complexity of an insertion operation is decreased as $O(m)$. Furthermore, the time complexity of performing a full insertion process is decreased as $O(m \times (n + F - 1))$ for job insertion.

The Destruction–Construction (denoted as LLH_2) is utilized to perturb a solution. First, several jobs are randomly chosen and removed from a random factory. Afterward, those selected jobs are re-inserted into all possible positions of all factories. The concrete details are shown in Procedure 2. The d jobs are sequentially inserted into all factory positions. The time complexity of each step is analyzed. The time complexity of the first construction is $O((n-d+F)\times m)$. Afterward, the time complexity of the second construction is $O((n-d+1+F)\times m)$, and so on. The time complexity of the last construction is $O((n-1+F)\times m)$. Therefore, the time complexity of performing a full insertion process is $O(d \times m \times (n + F - d/2))$.

An insertion for each factory (denoted as LLH_3) is proposed. Each job is removed from the factory and reinserted into all possible positions from the original factory. The position with the best TF is selected. The procedure is repeated until the quality of the solution no longer improves. The concrete details are illustrated in Procedure 3. The time complexity of each insertion is decreased from $O(m \times n_f)$ to $O(m)$ according to the speed-up method. The time complexity of insertion for each factory is $O(\left(n_f\right)^2 \times m)$. The time complexity of insertion for all factories is $O(\sum_{f=1}^{F} \left(n_f\right)^2 \times m)$.

A swap for each factory (denoted as LLH_4) is proposed. Each job is removed from the factory and swapped with all possible positions. The position with the best TF is selected. Repeat the procedure until the quality of the solution no longer improves. The details are shown in Procedure 4. The speed-up method is inappropriate for the swaption. Therefore, the time complexity of each swap operation is $O(\left(n_f\right)^2 \times m)$.

Procedure 1 Job insertion

Input: current solution Π
Output: Π
Select a job randomly from a random factory;
Remove the selected job;
$TF_{\text{MIN}} \leftarrow$ MAX;
for $f = 1$ **to** F **do**
  for $i = 1$ to n^f **do**
    $TF_{\text{proposed}} \leftarrow$ Calculate the TF for Π' after inserting the selected job into the ith position of factory f;
    if $TF_{\text{proposed}} < TF_{\text{MIN}}$ **then**
      $TF_{\text{MIN}} \leftarrow TF_{\text{proposed}}$, $bestPos \leftarrow i$, $bestFac \leftarrow f$;
    end if
  end for
end for
Insert the selected job into $bestPos$ position of factory $bestFac$;

Procedure 2 Destruction-Construction

Input: current solution Π, $d = 8$
Output: Π
Select d jobs randomly and put them into the sequence σ;
Remove d jobs from Π ;
for $i = 1$ **to** d **do**
  $job_{\text{insert}} \leftarrow \sigma(i)$, $TF_{\text{MIN}} \leftarrow$ an infinitely positive number;
  for $f = 1$ **to** F **do**
    for $j = 1$ **to** n^f **do**
      $TF_{\text{proposed}} \leftarrow$ Calculate the TF for Π' after inserting the job_{insert} into the kth position of factory f;
      if $TF_{\text{proposed}} < TF_{\text{MIN}}$ **then**
        $TF_{\text{MIN}} \leftarrow TF_{\text{proposed}}$, $bestPos \leftarrow j$, $bestFac \leftarrow f$;
      end if
    end for
  end for
  The job_{insert} is inserted into position $bestPos$ of factory $bestFac$;
end for

Procedure 3 Insertion for each factory

Input: current solution Π
Output: Π
flag ← true, $\Pi_0 \leftarrow \Pi$;
while tag **do**
  tag ← false;
  for $i = 1$ **to** n^f **do**
    $\Pi_1 \leftarrow \pi_0^f(i)$ is removed from π_0^f;
    $\Pi_2 \leftarrow$ The schedule with the best TF by inserting $\pi_0^f(i)$ to all positions of factory f of Π_1;
    if $TF(\Pi_2) < TF(\Pi_0)$ **then**
      $\Pi_0 \leftarrow \Pi_2$, flag ← true;
      break;
    end if
  end for
end while
$\Pi \leftarrow \Pi_0$;

The time complexity of the swap for each factory is $O(((n_f)^3 - (n_f)^2) \times m)$. The time complexity of swap for all factories is $O(\sum_{f=1}^{F}((n_f)^3 - (n_f)^2) \times m)$.

An insertion for the critical factory (denoted as LLH_5) is proposed. The critical factory is not the one with the largest make-span, but rather the one with the largest total flowtime. Each job from the critical factory is removed and reinserted into all possible positions at other factories. The position with the best TF is selected. The procedure is repeated until an improved solution is found. The dependent details are described in Procedure 5. In addition, Fig. 8 depicts the Gantt chart of insertion for the critical factory. According to Fig. 8(a), the job sequence for the critical factory consists of job 1, 2, 4, 6, and 5. The

Procedure 4 Swap for each factory

Input: current solution Π

Output: Π

tag ← true, $\Pi_0 \leftarrow \Pi$;

while tag **do**

  tag ← false;

  for $i = 1$ **to** n^f **do**

    $\Pi_1 \leftarrow$ The schedule with the best TF by swapping $\pi_0^f(i)$ with all jobs of factory f of Π_0;

    if $TF(\Pi_1) < TF(\Pi_0)$ **then**

      $\Pi_0 \leftarrow \Pi_1$, flag ← true;

      break;

    end if

  end for

end while

$\Pi \leftarrow \Pi_0$;

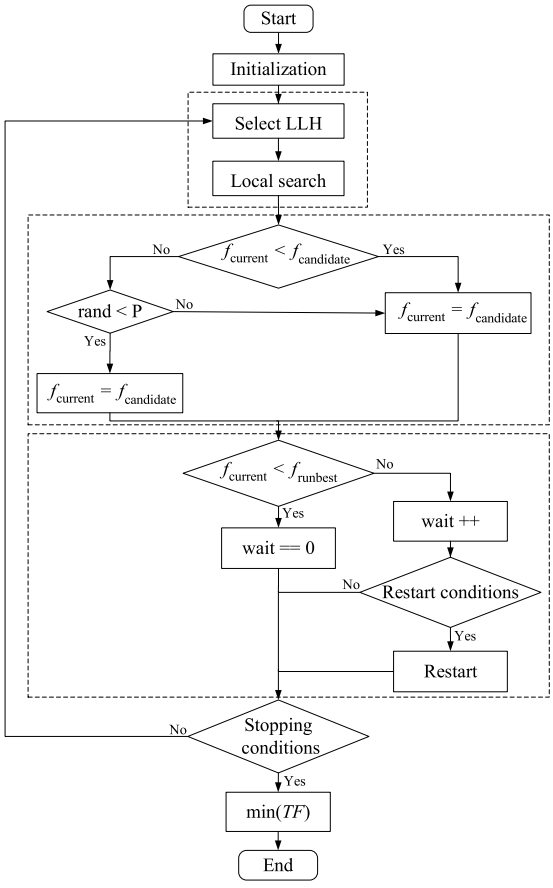


Fig. 7. The flowchart of the SLHH algorithm.

jobs from the critical factory are inserted into factory 2 to calculate the total flowtime. Suppose that job 2 is inserted into factory 2. There are four positions for factory 2 to insert the job 2. As shown in Fig. 8(b), in factory 2, the job 2 is inserted between job 8 and job 3. The starting assembly time of product 1 is advanced since job 2 is the last processed job instead of job 4 in Fig. 8(a). The time complexity of the insertion for the critical factory is $O(n^{fc} \times \sum_{f=1}^{F-1}(n^f+1) \times m)$, where n^{fc} represents the number of jobs in the critical factory.

A swap for the critical factory (denoted as LLH_6) is introduced. Each job from the critical factory is removed and swapped with all jobs of other factories. The position with the best TF is selected. The procedure is repeated when an improved solution is found. The relevant information is described in Procedure 6. Moreover, the Gantt chart of swap for the critical factory is depicted in Fig. 9. According to Fig. 9(a), the job sequence for the critical factory consists of job 1, 2, 6, 5, and 7. The jobs belonging to the critical factory are swapped with jobs in factory 2 for computing the total flowtime in turn. Suppose job 2 is selected to be swapped with jobs in factory 2. The position of job 2, which is swapped with job 4, decreases the starting assembly time of product 1 as shown in Fig. 9(b). The time complexity of the swap for the critical factory is $O(n^{fc} \times \sum_{f=1}^{F-1} \left(n^f\right)^2 \times m)$.

The VNS, an efficient and simple local search method, has been applied to combinatorial optimization problems. VNS searches the region surrounding the current solution systematically by altering neighborhood structures (Pan et al., 2019b). Variable search decent (VND) is a simplified version of variable search decent (VNS) (Pan et al., 2019b). DFSP has been successfully addressed using VND. VND begins with an initial solution and performs a search by modifying neighboring structures until the solution's state does not change. The VND will return to the initial neighborhood if a better solution is found. The search concludes when the final neighborhood has been applied, and no improved solution has been discovered. Presented is a VND based on three neighborhood structures. The relevant information is shown in Procedure 7. The time complexity of the proposed VND is difficult to evaluate since the number of while loops is uncertain. The execution number of times under the three cases is represented by three variables, which are k_1, k_2, and k_3 respectively. The time complexity of the proposed VND is $O\left(k_1 \times \sum_{f=1}^{F} \left(n^f\right)^2 \times m + k_2 \times \sum_{f=1}^{F} \left(\left(n^f\right)^3 - \left(n^f\right)^2\right) \times m + k_3 \times n^{fc} \times \sum_{f=1}^{F-1} \left(n^f+1\right) \times m\right)$. The minimum time complexity is $O(\sum_{f=1}^{F} \left(n^f\right)^3 \times m + n^{fc} \times \sum_{f=1}^{F-1} \left(n^f+1\right) \times m)$.

5. Experiment and analysis

To evaluate the effectiveness of the proposed constructive heuristic and hyper-heuristic algorithm, the computational experiments are conducted based on the benchmark set presented by Hatami et al. (2013). The bench set contains 910 small-scale instances and 810 large-scale

Procedure 5 Insertion for critical factory

Input: current solution Π

Output: Π

tag ← true, $\Pi_0 \leftarrow \Pi$;

while tag **do**

  tag ← false, and find the job sequence of critical factory π_0^{fc};

  for $i = 1$ **to** n^{fc} **do**

    Π_1 ← Remove $\pi_0^{fc}(i)$ from π_0^{fc};

    Π_2 ← The permutation with best TF by inserting $\pi_0^{fc}(i)$ into all positions of other factories of Π_0;

    if $TF(\Pi_2) < TF(\Pi_0)$ **then**

      $\Pi_2 \leftarrow \Pi_0$, flag ← true;

      break;

    end if

  end for

end while

$\Pi \leftarrow \Pi_0$;

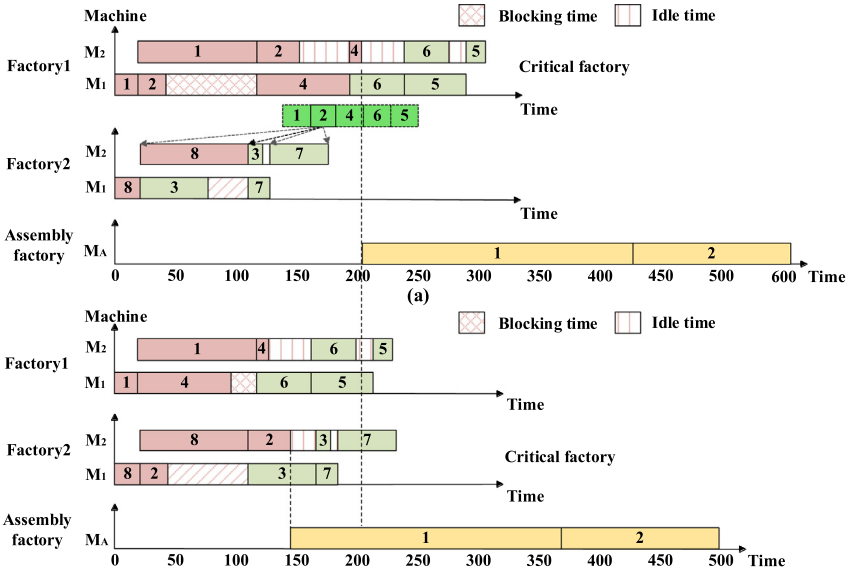


Fig. 8. Gantt chart of insertion for critical factory.

instances. There are the following combinations for the small-scale instances, where n = {8,12,16,20,24}, m = {2,3,4,5}, F = {2,3,4}, and S = {30,40,50}. For the large-scale instances, where n = {100,200,300}, m = {5,10,20}, F = {4,6,8}, and S = {30,40,50}. Small-scale instances contain five text instances per combination, while large-scale instances contain ten text instances per combination. To evaluate the quality of the solutions, the average relative percent deviation (ARPD) is utilized. The ARPD is defined in the following manner:

$$\mathrm{ARPD} = \left(\sum_{i=1}^{R} (TF_i - TF_{\mathrm{best}})/TF_{\mathrm{best}} \times 100\% \right) / R \tag{30}$$

where the TF_i represents the solution obtained by a certain algorithm for the ith run of a given instance, and TF_{best} is the best solution found so far. R is the number of run times for an algorithm on a single case. Each algorithm is executed five times per instance, and the solutions obtained by each algorithm are recorded to calculate ARPD values. In addition, the Bonferroni–Dunn method is used to compute the critical difference (CD). CD is determined as follows:

$$CD = q_a \sqrt{k_a(k_a+1)/(6 \times N)} \tag{31}$$

where N is the number of instances, the k_a is the number of all algorithms. The q_a is related with k_a at 95 or 90 percent confidence interval. Moreover, all algorithms are coded using the programing language MATLAB. The experiment strictly adheres to the parameters outlined in the original literature. The simulation experiments are conducted on a server with Intel (R) Xeon (R) gold 5218 CPU @ 2.30 GHz 2.29 GHz processors and 64 GB of RAM running Windows Server 2019.

5.1. Evaluation of constructive heuristic

To validate the effectiveness of the proposed constructive heuristic, the H1NRa, H2NRa, H3NRa, and HLPT are compared to the HHNRa (Shao et al., 2019, 2020b). Since the five constructive heuristics are used to address various DAFSP, they are re-implemented using the same evaluation function. The testbed employs the benchmark instances

Procedure 6 Swap for critical factory

Input: current solution Π

Output: Π

tag ← true, $\Pi_0 \leftarrow \Pi$;

while tag **do**

  tag ← false, and find the job sequence of critical factory π_0^{fc};

  for $i = 1$ **to** n^{fc} **do**

    $\Pi_1 \leftarrow$ The schedule with best TF by swapping $\pi_0^{fc}(i)$ with all jobs of other factories of Π_0;

    if $TF(\Pi_1) < TF(\Pi_0)$ **then**

      $\Pi_0 \leftarrow \Pi_1$, flag ← true;

      break;

    end if

  end for

end while

$\Pi \leftarrow \Pi_0$;

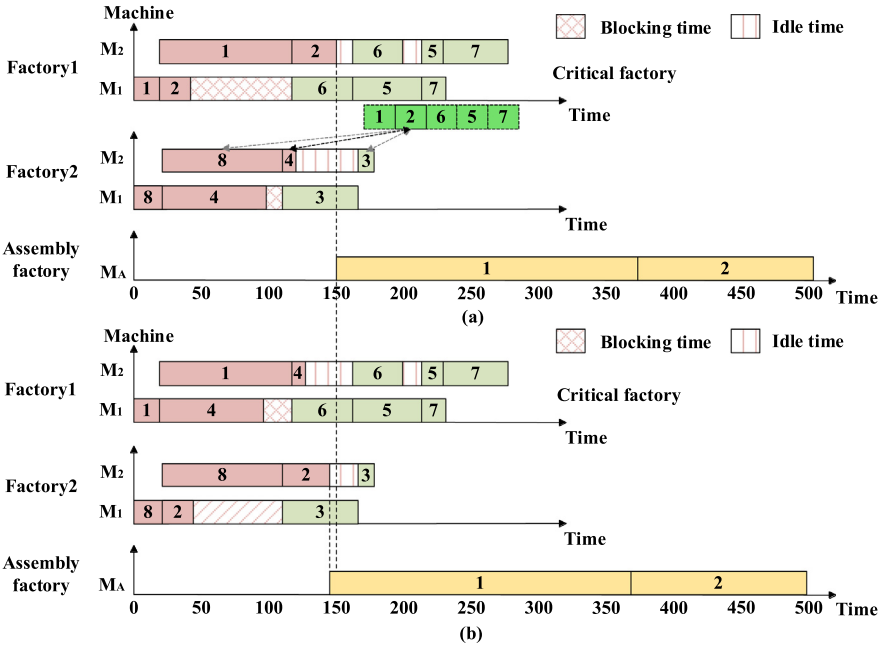


Fig. 9. Gantt chart of swap for critical factory.

proposed by Hatami et al. (2013). To illustrate the performance of the proposed constructive heuristic, the statistical results of ARPD values are grouped by jobs, factories, products, and machines in Tables 2 and 3. The minimum values in the experimental results are bold.

As seen in Table 2, the HLPT and HHNRa are two competitive constructive heuristic methods for small-scale instances, as shown in Table 2. In addition, HHNRa's performance is slightly superior to that of HLPT. In large-scale instances, the performance of HLPT is inferior to that of other algorithms compared in Table 3. The experimental results indicate that the heuristic based on problem-specific knowledge performs exceptionally well on both small-scale and large-small instances. In addition, for a clear comparison of experimental results, Figs. 10 and 11 illustrate the ARPD values for the various constructive heuristics. The vertical axis represents the ARPD values of the comparative constructive heuristics, whereas the horizontal axis depicts the various instances. For various instances, Figs. 10 and 11 demonstrate that the proposed constructive heuristic HHNRa is effective in both small-scale and large-scale instances.

5.2. Comparison with other algorithms

On 810 large-scale benchmark instances, the performance of the proposed SLHH is compared to that of other algorithms. Three effective strategies used to address other DAFSP are extended to DABFSP. The compared algorithms are iterated local search method and variable search decent method from literature (Shao et al., 2019), the iterated local search method from literature (Shao et al., 2020b), and the water wave optimization algorithm with problem-specific knowledge (Zhao et al., 2022). The abbreviations for the three methods are ILS1, VNS1, ILS2, and KWWO, respectively. In addition, SLHH* algorithm, removes the restart scheme from the SLHH algorithm, is introduced for comparison with the other algorithms. To closely achieve published performance, all specifics provided in the original literature

Procedure 7 The VND

Input: current solution Π_0

Output: Π_0

$l_{\max} \leftarrow 3, l \leftarrow 1;$

while $l \leq l_{\max}$ **do**

  switch l **do**

    case 1: $\Pi_1 \leftarrow$ Insertion for critical factory;

    case 2: $\Pi_1 \leftarrow$ Swap for critical factory;

    case 3: $\Pi_1 \leftarrow$ Swap for each factory;

  end switch

  if $TF(\Pi_1) < TF(\Pi_0)$ **then**

    $l \leftarrow 1, \Pi_0 \leftarrow \Pi_1;$

  else

    $l \leftarrow l + 1;$

  end if

end while

Table 2
ARPD values of the constructive heuristics on the small-scale instances.

Category	Number	H1NRa	H2NRa	H3NRa	HLPT	HHNRa
n	8	2.659	3.095	7.750	1.808	**1.017**
	12	3.966	4.446	7.718	1.639	**1.047**
	16	3.989	5.099	8.523	**1.068**	1.097
	20	3.933	4.564	10.355	1.444	**1.369**
	24	4.213	5.278	9.355	**0.994**	1.020
F	2	4.204	4.439	8.301	**0.324**	0.975
	3	3.668	4.556	8.987	**0.944**	1.117
	4	3.384	4.494	9.053	2.903	**1.237**
S	2	2.737	3.142	6.637	2.017	**1.141**
	3	3.450	4.253	9.099	1.196	**1.086**
	4	5.069	6.094	10.604	**0.959**	1.103
m	2	3.430	4.268	8.460	1.475	**1.034**
	3	3.393	4.014	8.244	1.682	**1.138**
	4	3.991	4.673	9.331	**1.128**	1.196
	5	4.195	5.031	9.086	1.276	**1.072**
Mean		3.752	4.496	8.767	1.390	**1.110**

Table 3
ARPD values of the constructive heuristics on the large-scale instances.

Category	Number	H1NRa	H2NRa	H3NRa	HLPT	HHNRa
n	100	**0.727**	4.138	10.314	29.694	1.503
	200	2.045	6.139	15.072	33.140	**0.467**
	500	2.456	4.312	10.271	30.636	**0.207**
F	4	2.289	6.257	15.439	27.343	**0.780**
	6	1.526	4.407	11.107	31.629	**0.775**
	8	1.412	3.926	9.201	34.498	**0.622**
S	30	1.925	5.048	12.018	28.226	**0.811**
	40	1.726	4.623	11.797	32.266	**0.836**
	50	1.578	4.918	11.842	32.978	**0.531**
m	5	1.066	3.453	7.454	34.964	**0.795**
	10	1.562	4.640	12.022	31.150	**0.753**
	20	2.599	6.496	16.181	27.056	**0.630**
Mean		1.743	4.863	11.894	31.137	**0.726**

are adhered to precisely. All compared algorithms' objective function are evaluated based on their total flowtime. To investigate algorithm performance, the CPU time of all compared algorithms is set as follows: $T_{max} = n \times m \times f \times 5$. The ARPD values of all algorithms grouped by jobs, factories, products, and machines are exhibited in Table 4. Moreover, to facilitate a clear comparison of experimental results, Fig. 12 depicts the experimental results for compared algorithms on large-scale instances. The vertical axis represents the ARPD values of all algorithms being compared, while the horizontal axis represents the various instances. Fig. 12 demonstrates that the proposed SLHH algorithm is effective on large-scale instances for various instances.

Table 4
ARPD values of all compared algorithms on the large-scale instances.

Category	Number	ILS1	VNS1	ILS2	KWWO	SLHH*	SLHH
n	100	1.419	1.405	7.107	2.069	0.819	**0.632**
	200	1.488	1.745	6.160	3.066	1.160	**1.109**
	500	2.062	2.045	4.903	2.943	0.381	**0.366**
F	4	2.213	2.318	7.139	3.373	1.151	**0.972**
	6	1.444	1.518	5.915	2.563	0.672	**0.628**
	8	1.314	1.360	5.119	2.143	0.537	**0.507**
S	30	1.878	2.039	5.644	2.912	0.763	**0.706**
	40	1.716	1.822	6.291	2.796	0.760	**0.667**
	50	1.377	1.335	6.237	2.732	0.838	**0.735**
m	5	1.148	1.206	6.253	2.184	0.719	**0.632**
	10	1.588	1.622	6.098	2.684	0.804	**0.772**
	20	2.237	2.368	5.822	3.211	0.838	**0.705**
Mean		1.657	1.732	6.057	2.723	0.789	**0.703**

The 95% confidence box plots are used to analyze the performance of the proposed SLHH algorithm further. Fig. 13 depicts the box plots of the ARPD values for large-scale benchmark instances, which are grouped by jobs, factories, products, and machines and evaluated using various algorithms. Box plot, a graphical representation of the discrete data distribution, is frequently employed in data analysis. The middle line of the box plot represents the sample data's mean level. The upper and lower limits of the box plot separately represent the data's upper quartile (Q3) and lower quartile (Q1). The difference between the third and first quartiles is the quartile difference (IQR). The degree of fluctuation of the data is reflected by the IQR (Zhao et al., 2022). Specifically, the smaller the IQR, the less variance in the data and the greater the algorithm's stability. Based on Fig. 13, the data fluctuation of the SLHH algorithm is less than that of the other compared algorithms, indicating that the SLHH algorithm is more stable than ILS1, VNS1, ILS2 and KWWO.

This study utilizes the Wilcoxon-test to compare SLHH to other algorithms under consideration. Based on the sign test of paired observation data, this method is more effective than the standard plus

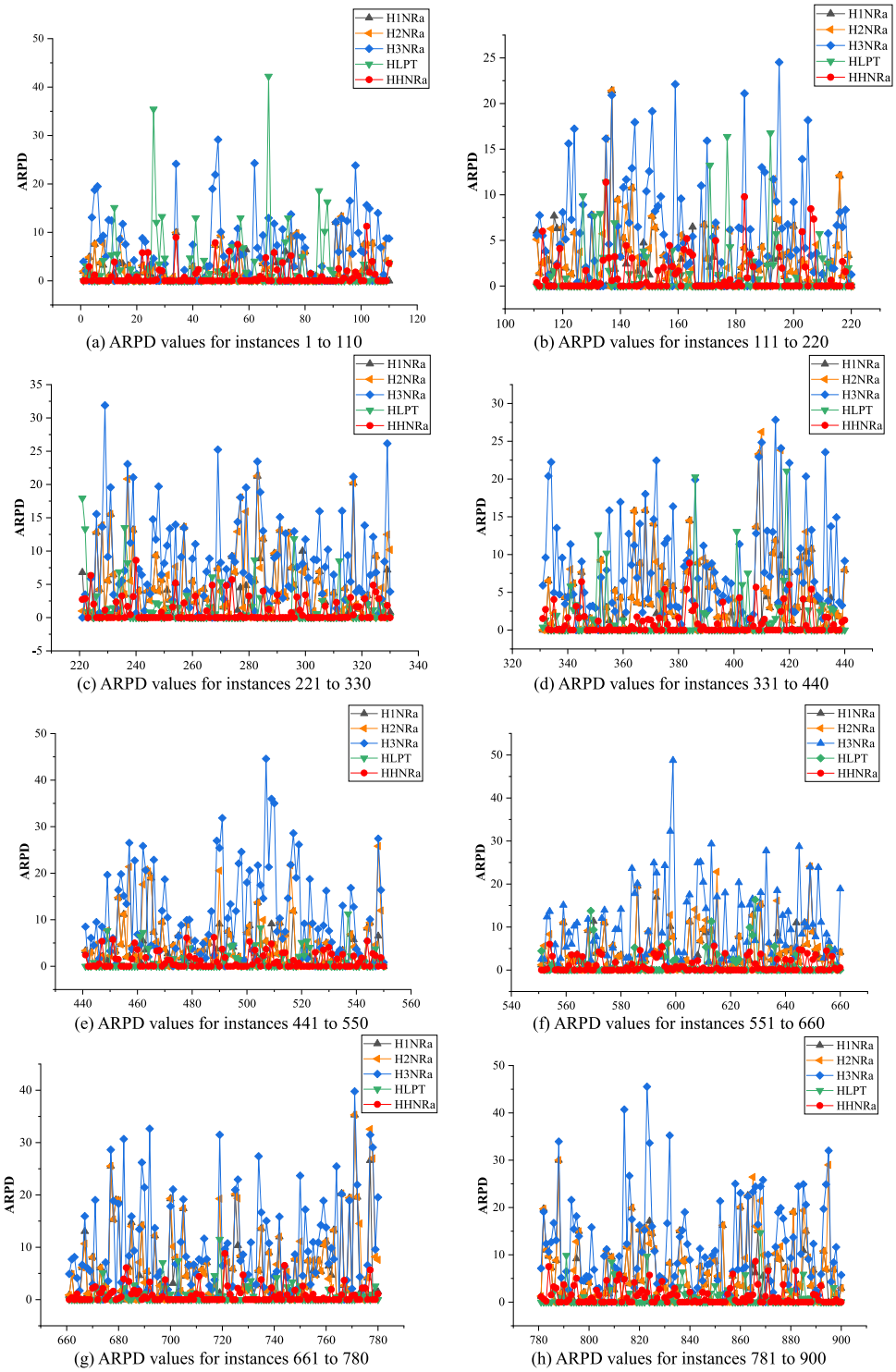


Fig. 10. The visualization of the ARPD values for different constructive heuristics on the small-scale instances.

or minus sign alone. Table 5 shows the results of the Wilcoxon sign test. The R^+ represents that the results are outstanding compared with the other compared algorithms. R^- is the opposite. "Yes" indicates that there exists a significant difference between the SLHH algorithm and the other compared algorithms. "Bonding value" indicates how many objective function values are the same on a large-scale instance in the pairwise comparison algorithm. If the p-value is less than α in the Wilcoxon test, it indicates that there are significant differences between the pairwise algorithms. According to Table 5, all p-values for large-scale instances are less than α. According to the Wilcoxon-test, the SLHH is, therefore, more promising than other algorithms.

In the meantime, the Friedman test is used to analyze significant differences between all algorithms being compared. Table 6 demonstrates the mean ranks of all compared algorithms as determined by the Friedman test under various conditions. In the experiment, the N is set to 270 and k_a is set to 6 when $\alpha = 0.10$, $q_a = 2.327$, when $\alpha = 0.05$, $q_a =$

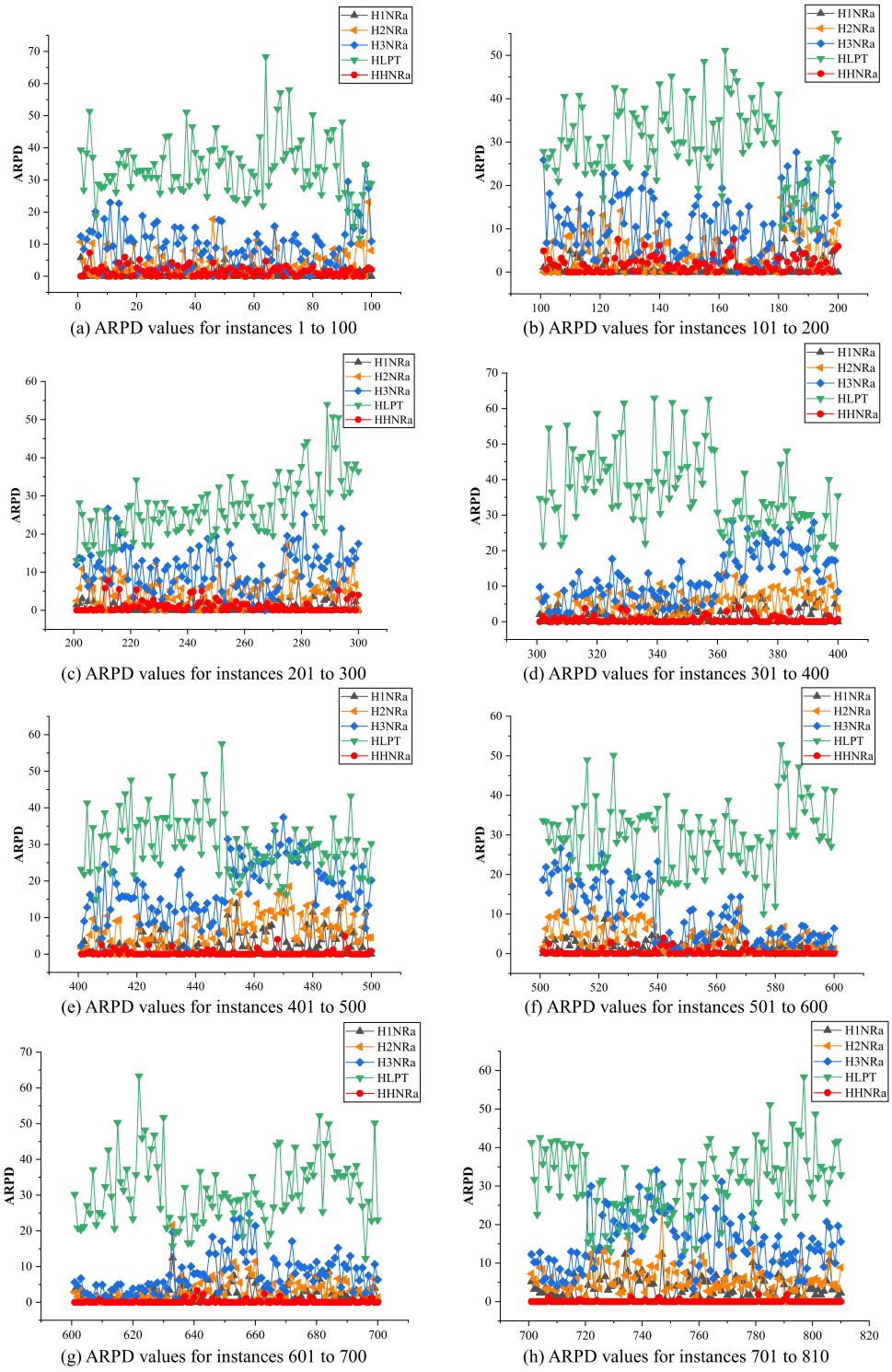


Fig. 11. The visualization of the ARPD values for different constructive heuristics on the large-scale instances.

Table 5
Results achieved by the Wilcoxon-test.

Comparison	R^+	R^-	Bonding value	p-value	$\alpha = 0.05$	$\alpha = 0.1$
SLHH vs. ILS1	534	276	0	2.30E−33	Yes	Yes
SLHH vs. VNS1	554	256	0	5.79E−38	Yes	Yes
SLHH vs. ILS2	805	5	0	5.05E−133	Yes	Yes
SLHH vs. KWWO	769	41	0	2.59E−127	Yes	Yes
SLHH vs. SLHH*	501	255	54	1.60E−22	Yes	Yes

2.576 (Zhao et al., 2020b) the critical difference (CD) is computed according to (31). When $\alpha = 0.10$, CD = 0.375, when $\alpha = 0.05$, CD = 0.415. From Table 6, the mean ranks of the SLHH algorithm under different conditions are the smallest in the five compared algorithms. Several sets of results for the Bonferroni–Dunn method under various conditions are presented in Fig. 14, and the SLHH is the control algorithm. It is evident from Fig. 14 that the SLHH algorithm differs significantly from other comparable algorithms. Table 7 shows the mean ranks of all compared algorithms as determined by the Friedman test on all large-scale instances. Where N is 810 and the other parameters are the same

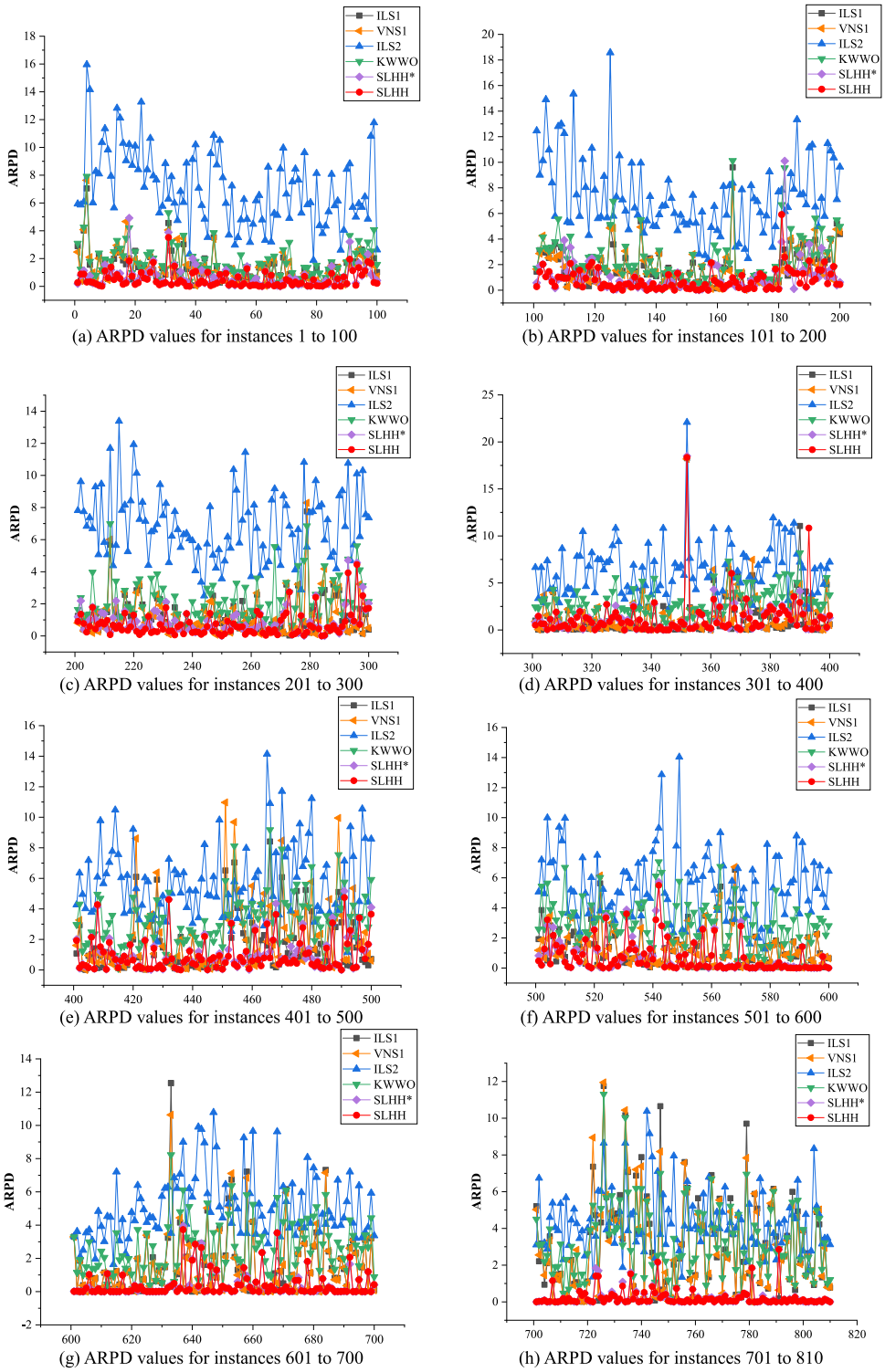


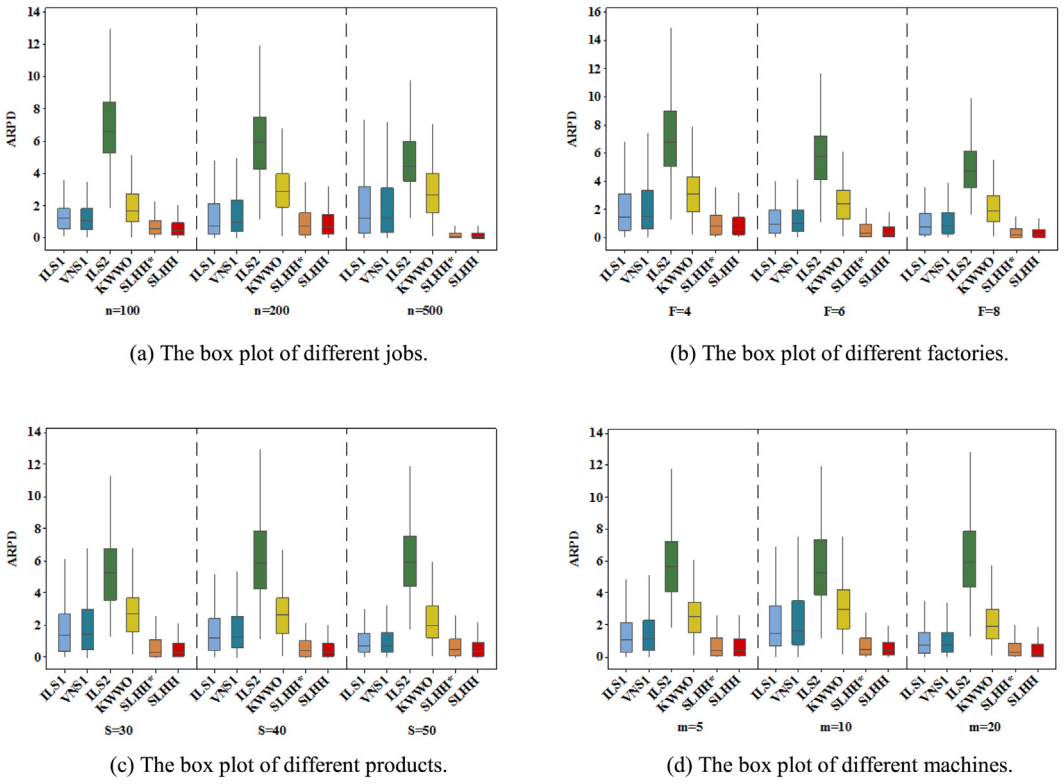
Fig. 12. The visualization of the ARPD values for the compared algorithms on the large-scale instances.

as in the previous section. Lastly, the Bonferroni–Dunn method results are depicted in Fig. 15. The SLHH algorithm is significantly superior to the other compared algorithms as demonstrated by the preceding analysis.

In this section, the effectiveness analysis of the restart scheme is described. As stated previously, the restart strategy is eliminated from the SLHH algorithm. Table 8 shows the ARPD values of two algorithms grouped by jobs, factories, products, and machines. According to Table 8, the ARPD values of the SLHH algorithm are superior to those of the SLHH*. Moreover, Fig. 16 depicts the ARPD comparison diagram for the two algorithms under various conditions. Based on the results, the restart scheme for the proposed algorithm is effective.

6. Conclusion and future work

This paper investigated the distributed assembly blocking flow shop scheduling problem with the total flowtime minimization criterion. Effective self-learning hyper-heuristic is designed to address the problem



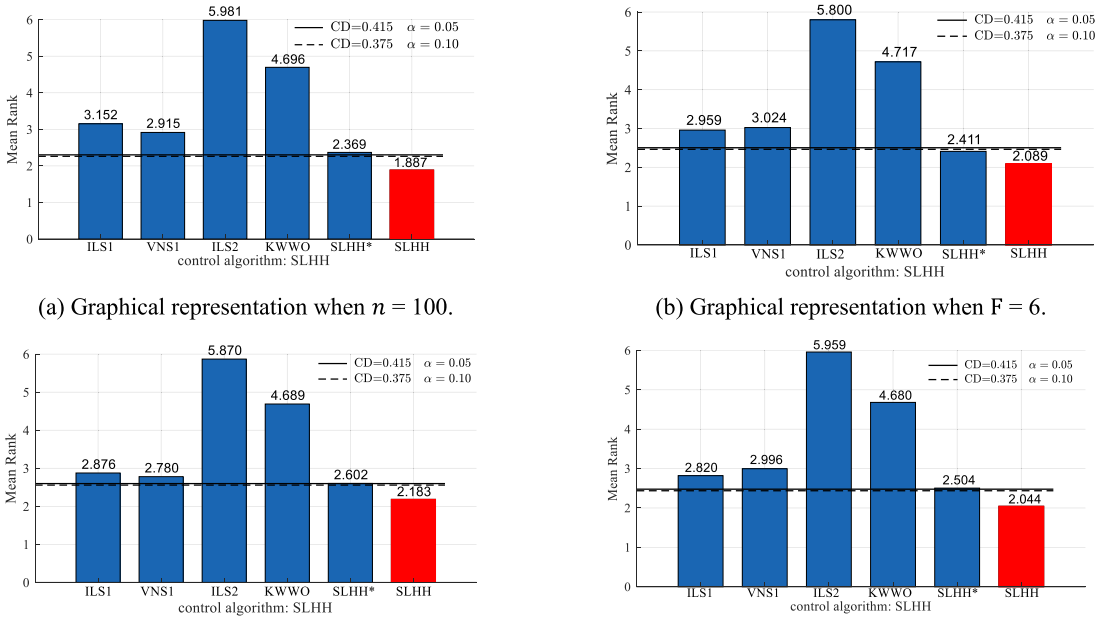
(a) The box plot of different jobs.

(b) The box plot of different factories.

(c) The box plot of different products.

(d) The box plot of different machines.

Fig. 13. Box plots of the compared algorithms on large-scale benchmark instances.



(a) Graphical representation when n = 100.

(b) Graphical representation when F = 6.

(c) Graphical representation when S = 50.

(d) Graphical representation when m = 5.

Fig. 14. Graphical representation of the Bonferroni–Dunn's test under different conditions.

under consideration. In terms of solution quality, numerical experimental results demonstrate that the proposed algorithm outperforms the well-established algorithms. Therefore, it is confirmed that the self-learning hyper-heuristic is an effective method for increasing factory efficiency by optimizing the goals set by the decision-maker. Due to the existence of problem-specific low-level heuristics, it should be noted that the algorithm cannot be directly applied to the other scheduling problem. However, the idea based on problem-specific knowledge and self-learning has a certain guiding significance for solving other complex problems in the modern manufacturing system.

Future research will consider other production scheduling problem in actual production scenarios, including distributed production

Table 6
Friedman test results on the large-scale benchmark instances.

Category	Number	ILS1	VNS1	ILS2	KWWO	SLHH*	SLHH
	100	3.152	2.915	5.981	4.696	2.369	1.887
n	200	2.344	2.996	5.778	4.700	2.757	2.424
	500	3.404	3.363	5.533	4.733	2.093	1.874
	4	2.922	3.170	5.681	4.670	2.515	2.041
F	6	2.959	3.024	5.800	4.717	2.411	2.089
	8	3.019	3.080	5.811	4.743	2.293	2.056
	30	3.063	3.322	5.685	4.689	2.267	1.974
S	40	2.961	3.172	5.737	4.752	2.350	2.028
	50	2.876	2.780	5.870	4.689	2.602	2.813
	5	2.820	2.996	5.956	4.680	2.504	2.044
m	10	2.870	2.909	5.841	4.375	2.424	2.220
	20	3.209	3.369	5.496	4.715	2.291	1.920

Table 7
Friedman test results of Six algorithms.

Algorithm	Mean rank
ILS1	2.967
VNS1	3.091
ILS2	5.764
KWWO	4.710
SLHH*	2.406
SLHH	2.062
Crit.Diff. $\alpha = 0.05$	0.415
Crit.Diff. $\alpha = 0.10$	0.375

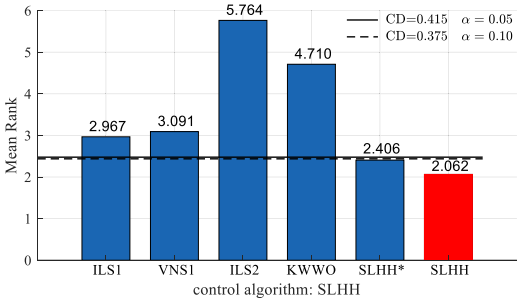


Fig. 15. Graphical representation of the Bonferroni–Dunn's test.

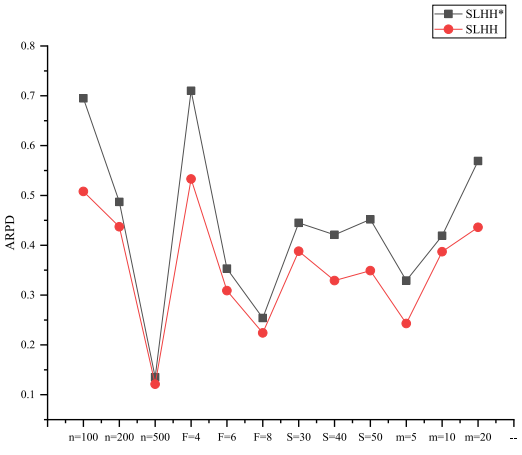


Fig. 16. The comparison diagram of ARPD of the two algorithms under different conditions.

scheduling in a heterogeneous environment, distributed production scheduling in an uncertain environment, and distributed flexible hybrid flow shop scheduling. In addition, other realistic constraints and scopes, such as green energy conservation, sequence-dependent setup time, and transportation time, are key factors to consider.

Table 8
ARPD of the two algorithms under different conditions.

Category	Number	SLHH*	SLHH
	100	0.695	**0.508**
n	200	0.487	**0.437**
	500	0.135	**0.121**
	4	0.710	**0.533**
F	6	0.353	**0.309**
	8	0.254	**0.224**
	30	0.445	**0.388**
S	40	0.421	**0.329**
	50	0.452	**0.349**
	5	0.329	**0.243**
m	10	0.419	**0.387**
	20	0.569	**0.436**
Mean		0.439	**0.355**

CRediT authorship contribution statement

Fuqing Zhao: Funding acquisition, Investigation, Supervision. **Shilu Di:** Investigation, Software, Writing – original draft, Experiments of the algorithms. **Ling Wang:** Methodology, Resources. **Tianpeng Xu:** Project administration, Writing – review & editing. **Ningning Zhu:** Conceptualization, Formal analysis. **Jonrinaldi:** Visualization.

Declaration of competing interest

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

Acknowledgments

This work was financially supported by the National Natural Science Foundation of China under grant 62063021. It was also supported by the Key talent project of Gansu Province (ZZ2021G50700016), the Key Research Programs of Science and Technology Commission Foundation of Gansu Province (21YF5WA086), Lanzhou Science Bureau project (2018-rc-98), and Project of Gansu Natural Science Foundation (21JR7RA204), respectively.

References

Almeida, C.P., Gonçalves, R.A., Venske, S., Lüders, R., Delgado, M., 2020. Hyper-heuristics using multi-armed bandit models for multi-objective optimization. Appl. Soft Comput. 95, 106520. http://dx.doi.org/10.1016/J.ASOC.2020.106520.

Aqil, S., Allali, K., 2021. Two efficient nature inspired meta-heuristics solving blocking hybrid flow shop manufacturing problem. Eng. Appl. Artif. Intell. 100, 104196. http://dx.doi.org/10.1016/J.ENGAPPAI.2021.104196.

Brum, A., Ruiz, R., Ritt, M., 2022. Automatic generation of iterated greedy algorithms for the non-permutation flow shop scheduling problem with total completion time minimization. Comput. Ind. Eng. 163, 107843. http://dx.doi.org/10.1016/J.CIE.2021.107843.

Cai, J., Zhou, R., Lei, D., 2020. Dynamic shuffled frog-leaping algorithm for distributed hybrid flow shop scheduling with multiprocessor tasks. Eng. Appl. Artif. Intell. 90, 103540. http://dx.doi.org/10.1016/J.ENGAPPAI.2020.103540.

Chen, D., Zhao, X.R., 2021. Production management of hybrid flow shop based on genetic algorithm. Int. J. Simul. Model. 20, 571–582. http://dx.doi.org/10.2507/IJSIMM20-3-CO12.

Costa, A., Cappadonna, F.V., Fichera, S., 2020. Minimizing makespan in a flow shop sequence dependent group scheduling problem with blocking constraint. Eng. Appl. Artif. Intell. 89, 103413. http://dx.doi.org/10.1016/J.ENGAPPAI.2019.103413.

Cui, W., Lu, B., 2021. Energy-aware operations management for flow shops under TOU electricity tariff. Comput. Ind. Eng. 151, 106942. http://dx.doi.org/10.1016/J.CIE.2020.106942.

Deng, G., Su, Q., Zhang, Z., Liu, H., Zhang, S., Jiang, T., 2020. A population-based iterated greedy algorithm for no-wait job shop scheduling with total flow time criterion. Eng. Appl. Artif. Intell. 88, 103369. http://dx.doi.org/10.1016/J.ENGAPPAI.2019.103369.

Dong, B., Jiao, L., Wu, J., 2015. A two-phase knowledge based hyper-heuristic scheduling algorithm in cellular system. Knowl.-Based Syst. 88, 244–252. http://dx.doi.org/10.1016/J.KNOSYS.2015.07.028.

Drake, J.H., Kheiri, A., Özcan, E., Burke, E.K., 2020. Recent advances in selection hyper-heuristics. Eur. J. Oper. Res. 285, 405–428. http://dx.doi.org/10.1016/J.EJOR.2019.07.073.

Hatami, S., Ruiz, R., Andrés-Romano, C., 2013. The distributed assembly permutation flowshop scheduling problem. Int. J. Prod. Res. 51, 5292–5308. http://dx.doi.org/10.1080/00207543.2013.807955.

He, X., Pan, Q., Gao, L., Wang, L., Suganthan, P.N., 2021. A greedy cooperative co-evolution ary algorithm with problem-specific knowledge for multi-objective flowshop group scheduling problems. IEEE Trans. Evol. Comput. 1. http://dx.doi.org/10.1109/TEVC.2021.3115795.

Heger, J., Voss, T., 2021. Dynamically adjusting the k-values of the ATCS rule in a flexible flow shop scenario with reinforcement learning. http://dx.doi.org/10.1080/00207543.2021.1943762.

Huang, J.P., Pan, Q.K., Gao, L., 2020. An effective iterated greedy method for the distributed permutation flowshop scheduling problem with sequence-dependent setup times. Swarm Evol. Comput. 59, 100742. http://dx.doi.org/10.1016/J.SWEVO.2020.100742.

Jing, X.L., Pan, Q.K., Gao, L., Wang, Y.L., 2020. An effective iterated greedy algorithm for the distributed permutation flowshop scheduling with due windows. Appl. Soft Comput. 96, 106629. http://dx.doi.org/10.1016/J.ASOC.2020.106629.

Jq, L., Mx, S., L, W., Py, D., Yy, H, Hy., S., Qk, P., 2020. Hybrid artificial bee colony algorithm for a parallel batching distributed flow-shop problem with deteriorating jobs. IEEE Trans. Cybern. 50, 2425–2439. http://dx.doi.org/10.1109/TCYB.2019.2943606.

Khatir, S., Boutchicha, D., Le Thanh, C., Tran-Ngoc, H., Nguyen, T.N., Abdel-Wahab, M., 2020. Improved ANN technique combined with jaya algorithm for crack identification in plates using XIGA and experimental analysis. Theor. Appl. Fract. Mech. 107, 102554. http://dx.doi.org/10.1016/J.TAFMEC.2020.102554.

Khatir, S., Tiachacht, S., Le Thanh, C.., Ghandourah, E., Mirjalili, S., Abdel Wahab, M., 2021. An improved artificial neural network using arithmetic optimization algorithm for damage assessment in FGM composite plates. Compos. Struct. 273, 114287. http://dx.doi.org/10.1016/J.COMPSTRUCT.2021.114287.

Kheiri, A., Özcan, E., 2016. An iterated multi-stage selection hyper-heuristic. Eur. J. Oper. Res. 250, 77–90. http://dx.doi.org/10.1016/J.EJOR.2015.09.003.

Lee, C.Y., Cheng, T.C.E., Lin, B.M.T., 1993. Minimizing the Makespan in the 3-Machine Assembly-Type Flowshop Scheduling Problem. 39, 616–625. http://dx.doi.org/10.1287/MNSC.39.5.616.

Lin, J., Wang, Z.J., Li, X., 2017. A backtracking search hyper-heuristic for the distributed assembly flow-shop scheduling problem. Swarm Evol. Comput. 36, 124–135. http://dx.doi.org/10.1016/J.SWEVO.2017.04.007.

Lin, J., Zhang, S., 2016. An effective hybrid biogeography-based optimization algorithm for the distributed assembly permutation flow-shop scheduling problem. Comput. Ind. Eng. 97, 128–136. http://dx.doi.org/10.1016/J.CIE.2016.05.005.

Lu, C., Gao, L., Yi, J., Li, X., 2020. Energy-efficient scheduling of distributed flow shop with heterogeneous factories: A real-world case from automobile industry in China. IEEE Trans. Ind. Inform. http://dx.doi.org/10.1109/TII.2020.3043734.

Merchan, A.F., Maravelias, C.T., 2016. Preprocessing and tightening methods for time-indexed MIP chemical production scheduling models. Comput. Chem. Eng. 84, 516–535. http://dx.doi.org/10.1016/J.COMPCHEMENG.2015.10.003.

Morais, M. de F., Ribeiro, M.H.D.M., da Silva, R.G., Mariani, V.C., Coelho, L. dos S., 2022. Discrete differential evolution metaheuristics for permutation flow shop scheduling problems. Comput. Ind. Eng. 166, 107956. http://dx.doi.org/10.1016/J.CIE.2022.107956.

Naderi, B., Ruiz, R., 2010. The distributed permutation flowshop scheduling problem. Comput. Oper. Res. 37, 754–768. http://dx.doi.org/10.1016/J.COR.2009.06.019.

Nguyen-Le, D.H., Tao, Q.B., Nguyen, V.H., Abdel-Wahab, M., Nguyen-Xuan, H., 2020. A data-driven approach based on long short-term memory and hidden Markov model for crack propagation prediction. Eng. Fract. Mech. 235, 107085. http://dx.doi.org/10.1016/J.ENGFRACMECH.2020.107085.

Pan, Q.K., Dong, Y., 2014. An improved migrating birds optimisation for a hybrid flowshop scheduling with total flowtime minimisation. Inf. Sci. (Ny) 277, 643–655. http://dx.doi.org/10.1016/J.INS.2014.02.152.

Pan, Q.K., Gao, L., Wang, L., Liang, J., Li, X.Y., 2019a. Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem. Expert Syst. Appl. 124, 309–324. http://dx.doi.org/10.1016/J.ESWA.2019.01.062.

Pan, Q.K., Gao, L., Xin-Yu, L., Framinan, J.M., 2019b. Effective constructive heuristics and meta-heuristics for the distributed assembly permutation flowshop scheduling problem. Appl. Soft Comput. 81, 105492. http://dx.doi.org/10.1016/J.ASOC.2019.105492.

Pan, Q.K., Ruiz, R., 2013. A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. Comput. Oper. Res. 40, 117–128. http://dx.doi.org/10.1016/J.COR.2012.05.018.

Potts, C.N., Sevast'janov, S.V., Strusevich, V.A., Van Wassenhove, L.N., Zwaneveld, C.M., 1995. The two-stage assembly scheduling problem: complexity and approximation 43, 346–355. http://dx.doi.org/10.1287/OPRE.43.2.346.

Qin, H.X., Han, Y.Y., Zhang, B., Meng, L.L., Liu, Y.P., Pan, Q.K., Gong, D.W., 2022. An improved iterated greedy algorithm for the energy-efficient blocking hybrid flow shop scheduling problem. Swarm Evol. Comput. 69, 100992. http://dx.doi.org/10.1016/J.SWEVO.2021.100992.

Riahi, V., Khorramizadeh, M., Hakim Newton, M.A., Sattar, A., 2017. Scatter search for mixed blocking flowshop scheduling. Expert Syst. Appl. 79, 20–32. http://dx.doi.org/10.1016/J.ESWA.2017.02.027.

Riahi, V., Newton, M.A.H., Su, K., Sattar, A., 2019. Constraint guided accelerated search for mixed blocking permutation flowshop scheduling. Comput. Oper. Res. 102, 102–120. http://dx.doi.org/10.1016/J.COR.2018.10.003.

Ribas, I., Companys, R., Tort-Martorell, X., 2017. Efficient heuristics for the parallel blocking flow shop scheduling problem. Expert Syst. Appl. 74, 41–54. http://dx.doi.org/10.1016/J.ESWA.2017.01.006.

Ribas, I., Companys, R., Tort-Martorell, X., 2019. An iterated greedy algorithm for solving the total tardiness parallel blocking flow shop scheduling problem. Expert Syst. Appl. 121, 347–361. http://dx.doi.org/10.1016/J.ESWA.2018.12.039.

Safari, G., Hafezalkotob, A., Malekpour, H., Khalilzadeh, M., 2022. Competitive scheduling in a hybrid flow shop problem using multi-leader–multi-follower game - A case study from Iran. Expert Syst. Appl. 195, 116584. http://dx.doi.org/10.1016/J.ESWA.2022.116584.

Sang, H.Y., Pan, Q.K., Li, J.Q., Wang, P., Han, Y.Y., Gao, K.Z., Duan, P., 2019. Effective invasive weed optimization algorithms for distributed assembly permutation flowshop problem with total flowtime criterion. Swarm Evol. Comput. 44, 64–73. http://dx.doi.org/10.1016/J.SWEVO.2018.12.001.

Shao, W., Pi, D., Shao, Z., 2019. Local search methods for a distributed assembly no-idle flow shop scheduling problem. IEEE Syst. J. 13, 1945–1956. http://dx.doi.org/10.1109/JSYST.2018.2825337.

Shao, Z., Pi, D., Shao, W., 2020a. Hybrid enhanced discrete fruit fly optimization algorithm for scheduling blocking flow-shop in distributed environment. Expert Syst. Appl. 145, 113147. http://dx.doi.org/10.1016/J.ESWA.2019.113147.

Shao, W., Shao, Z., Pi, D., 2020. Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem. Knowl.-Based Syst. 194, 105527. http://dx.doi.org/10.1016/J.KNOSYS.2020.105527.

Shao, Z., Shao, W., Pi, D., 2020b. Effective constructive heuristic and metaheuristic for the distributed assembly blocking flow-shop scheduling problem. Appl. Intell. 5012 (50), 4647–4669. http://dx.doi.org/10.1007/S10489-020-01809-X, 2020.

Shao, Z., Shao, W., Pi, D., 2020c. Effective heuristics and metaheuristics for the distributed fuzzy blocking flow-shop scheduling problem. Swarm Evol. Comput. 59, 100747. http://dx.doi.org/10.1016/J.SWEVO.2020.100747.

Song, H.B., Lin, J., 2021. A genetic programming hyper-heuristic for the distributed assembly permutation flow-shop scheduling problem with sequence dependent setup times. Swarm Evol. Comput. 60, 100807. http://dx.doi.org/10.1016/J.SWEVO.2020.100807.

Tran-Ngoc, H., Khatir, S., Ho-Khac, H., De Roeck, G., Bui-Tien, T., Abdel Wahab, M., 2021. Efficient artificial neural networks based on a hybrid metaheuristic optimization algorithm for damage detection in laminated composite structures. Compos. Struct. 262, 113339. http://dx.doi.org/10.1016/J.COMPSTRUCT.2020.113339.

Tran-Ngoc, H., Khatir, S., Le-Xuan, T., De Roeck, G., Bui-Tien, T., Abdel Wahab, M., 2020. A novel machine-learning based on the global search techniques using vectorized data for damage detection in structures. Int. J. Eng. Sci. 157, 103376. http://dx.doi.org/10.1016/J.IJENGSCI.2020.103376.

Wang, L., Pan, Q.K., Suganthan, P.N., Wang, W.H., Wang, Y.M., 2010. A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems. Comput. Oper. Res. 37, 509–520. http://dx.doi.org/10.1016/J.COR.2008.12.004.

Wang, S.Y., Wang, L., 2016. An estimation of distribution algorithm-based memetic algorithm for the distributed assembly permutation flow-shop scheduling problem. IEEE Trans. Syst. Man, Cybern. Syst. 46, 139–149. http://dx.doi.org/10.1109/TSMC.2015.2416127.

Wang, J.J., Wang, L., 2020. A knowledge-based cooperative algorithm for energy-efficient scheduling of distributed flow-shop. IEEE Trans. Syst. Man, Cybern. Syst. 50, 1805–1819. http://dx.doi.org/10.1109/TSMC.2017.2788879.

Wang, S., Wang, H., Zhou, Y., Liu, J., Dai, P., Du, X., Abdel Wahab, M., 2021. Automatic laser profile recognition and fast tracking for structured light measurement using deep learning and template matching. Measurement 169, 108362. http://dx.doi.org/10.1016/J.MEASUREMENT.2020.108362.

Zhang, Z.Q., Qian, B., Hu, R., Jin, H.P., Wang, L., 2021. A matrix-cube-based estimation of distribution algorithm for the distributed assembly permutation flow-shop scheduling problem. Swarm Evol. Comput. 60, 100785. http://dx.doi.org/10.1016/J.SWEVO.2020.100785.

Zhang, G., Xing, K., Cao, F., 2018. Discrete differential evolution algorithm for distributed blocking flowshop scheduling with makespan criterion. Eng. Appl. Artif. Intell. 76, 96–107. http://dx.doi.org/10.1016/J.ENGAPPAI.2018.09.005.

Zhao, F., He, X., Wang, L., 2020a. A two-stage cooperative evolutionary algorithm with problem-specific knowledge for energy-efficient scheduling of no-wait flow-shop problem. IEEE Trans. Cybern. 1–13. http://dx.doi.org/10.1109/TCYB.2020.3025662.

Zhao, F., Shao, D., Wang, L., Xu, T., Zhu, N., Jonrinaldi, ., 2022. An effective water wave optimization algorithm with problem-specific knowledge for the distributed assembly blocking flow-shop scheduling problem. Knowl.-Based Syst. 243, 108471. http://dx.doi.org/10.1016/J.KNOSYS.2022.108471.

Zhao, F., Zhang, L., Cao, J., Tang, J., 2021. A cooperative water wave optimization algorithm with reinforcement learning for the distributed assembly no-idle flowshop scheduling problem. Comput. Ind. Eng. 153, 107082. http://dx.doi.org/10.1016/J.CIE.2020.107082.

Zhao, F., Zhao, L., Wang, L., Song, H., 2020b. An ensemble discrete differential evolution for the distributed blocking flowshop scheduling with minimizing makespan criterion. Expert Syst. Appl. 160, 113678. http://dx.doi.org/10.1016/J.ESWA.2020.113678.

Zheng, J., Wang, L., Wang, J. jing, 2020. A cooperative coevolution algorithm for multi-objective fuzzy distributed hybrid flow shop. Knowl.-Based Syst. 194, 105536. http://dx.doi.org/10.1016/J.KNOSYS.2020.105536.