Expert Systems With Applications 232 (2023) 120837

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Q-learning driven multi-population memetic algorithm for distributed three-stage assembly hybrid flow shop scheduling with flexible preventive maintenance

Yanhe Jia [a], Qi Yan [b], Hongfeng Wang [b,*]

[a] *School of Economics and Management, Beijing Information Science & Technology University, Beijing 100192, China*
[b] *College of Information Science and Engineering, Northeastern University, Shenyang 110819, China*

A R T I C L E I N F O

Keywords:
Distributed hybrid flow shop
Transportation and assembly
Preventive maintenance
Meta-heuristics
Reinforcement learning
Integration

A B S T R A C T

The distributed assembly flow shop scheduling (DAFS) problem has received much attention in the last decade, and a variety of metaheuristic algorithms have been developed to achieve the high-quality solution. However, there are still some limitations. On the one hand, these studies usually ignore the machine deterioration, maintenance, transportation as well as the flexibility of flow shops. On the other hand, metaheuristic algorithms are prone to fall into local optimality and are unstable in solving complex combinatorial optimization problems. Therefore, a multi-population memetic algorithm (MPMA) with Q-learning (MPMA-QL) is developed to address a distributed assembly hybrid flow shop scheduling problem with flexible preventive maintenance (DAHFSP-FPM). Specifically, a mixed integer linear programming (MILP) model targeted at the minimal makespan is first established, followed by an effective flexible maintenance strategy to simplify the model. To efficiently solve the model, MPMA is developed and Q-learning is used to achieve an adaptive individual assignment for each sub-population to improve the performance of MPMA. Finally, two state-of-the-art metaheuristics and their Q-learning-based improvements are selected as rivals of the developed MPMA and MPMA-QL. A series of numerical studies are carried out along with a real-life case of a furniture manufacturing company, to demonstrate that MPMA-QL can provide better solutions on the studied DAHFSP-FPM..

1. Introduction

In today's fast-changing market, distributed manufacturing (DM) is becoming increasingly popular as a new mode to increase production flexibility and tackle the challenges of mass customization (Fu et al., 2021; Lohmer & Lasch, 2021; Srai et al., 2016). Distributed assembly flow-shop scheduling (DAFS) problem, as one of classical and challenging optimization problems under DM, is applicable in many practical manufacturing environments such as pharmaceutical production (Zhao, Xu, et al., 2022), furniture industry (Cai, Lei, Wang, & Wang, 2022). DAFS has also attracted the attention of a wide range of scholars in terms of the review paper of Komaki, Sheikh, and Malakooti (2019) as well as related works of recent three years.

A large portion of the research focused on the two-stage DAFS with distributed flow-shop fabrication and single-machine assembly and presented more and more efficient optimization algorithms. For instance, Zhao, Di, et al. (2022) and Zhao, Xu, et al. (2022) respectively designed a self-learning hyper-heuristic approach and a population-based iterated greedy algorithm to achieve the minimization of the total flow time. Zhang et al. (2022) presented a matrix cube-based estimation of distribution algorithm to tackle an energy-efficient DAFS with the objectives of minimizing the makespan and total carbon emission. Li, Pan, et al. (2022) developed a referenced iterated greedy algorithm to minimize the total tardiness. Song, Yang, Lin, and Ye (2023) proposed an effective hyper heuristic-based memetic algorithm to minimize the maximum completion time.

Some studies have additionally considered assembly processes with multiple assembly machines (Framinan, Perez-Gonzalez, & Fernandez-Viagas, 2019). For instance, Li et al. (2019) investigated a two-stage DAFS with parallel batching and linear deteriorating and developed a knowledge-based hybrid artificial bee colony algorithm. Lei, Su, and Li (2021) proposed a cooperated teaching–learning-based optimization algorithm to deal with a two-stage DAFS targeted at the minimal makespan, where each factory is equipped with an assembly machine.

* Corresponding author.
E-mail addresses: yhejia@bistu.edu.cn (Y. Jia), yanqqz@stumail.neu.edu.cn (Q. Yan), hfwang@mail.neu.edu.cn (H. Wang).

https://doi.org/10.1016/j.eswa.2023.120837
Received 1 March 2023; Received in revised form 11 June 2023; Accepted 11 June 2023
Available online 19 June 2023

Cai et al. (2022) proposed a shuffled frog-leaping algorithm for a three-stage distributed assembly hybrid flow shop scheduling (DAHFS) problem, in which each factory has a hybrid flow shop for fabrication, a transportation machine for collecting and transferring, and an assembly machine for final assembly.

Despite these DAFS-related initiatives, there is still more research that has to be refined. Various DAFS variations can be researched further in light of actual scenario demands and past research. On the one hand, DAHFS is rarely studied in existing studies. In reality, hybrid flow shop scheduling (HFS) and distributed hybrid flow shop scheduling (DHFS) problems are very common in real-world applications and have received a lot of attention in academia (Neufeld, Schulz, & Buscher, 2022; Shao, Shao, & Pi, 2020). Therefore, considering the flexibility of flow shops in DAFS is significant and realistic (Cai et al., 2022; Zhao, Zhou, & Liu, 2021). On the other hand, previous research typically ignored the transportation stage that plays an important and essential role between the production and assembly stages; thus, there is a need for a more in-depth study of the three-stage DAFS.

Moreover, machine deterioration and failures are inevitable in real-life assembly production, yet they are often neglected in DAFS-related research. There has been a succession of scholars to integrate appropriate maintenance activities into the assembly scheduling process in other manufacturing scenarios. For example, Zhang and Tang (2021) addressed a two-stage assembly flow shop scheduling problem with flexible preventive maintenance (PM) and parallel assembly machines, in which maintenance levels were defined to evaluate the states of each machine. Wang, Lei, et al. (2022) designed a

$$C_{a} \tag{1}$$

s.t.

$$C_{a} \geq E^{3} + A^{3} \forall \tag{2}$$

$$E^{3} \geq E^{2} + A^{2} \forall \tag{3}$$

$$E^{2} \geq E^{1} + A^{1} \forall \; , \in \Omega \; , \tag{4}$$

$$E^{1} \geq E^{1}_{, -1} + A^{1}_{, -1} \forall , \; \geq 2 \tag{5}$$

$$E^{1} \geq E^{1} + A^{1} + \xi^{1} \, {}^{1}_{PM} - L(2 - \mathscr{X}_{f \;, -1} - \mathscr{X}_{f}) \forall , \; , \; , f, \; , \geq 2 \tag{6}$$

$$E^{1}_{,1} \geq 0 \forall \tag{7}$$

$$a^{1} \geq 0 \forall , \tag{8}$$

$$A^{1} = P^{1} + {}_{1} a^{1} \forall , \tag{9}$$

$$a^{1} \geq a^{1} + A^{1} - L\left(2 - \mathscr{X}_{f \;, -1} - \mathscr{X}_{f} + \xi^{1}\right) \forall , \; , \; , f, \; , \geq 2 \tag{10}$$

$$a^{1} \geq \; - L(1 - \xi^{1}) \forall , \tag{11}$$

$$\xi^{1} \leq 1 - \sum_{f} \sum \mathscr{X}_{f \; 1} \forall , \tag{12}$$

$$E^{2} \geq E^{2} + A^{2} + \xi^{2} \, {}^{2}_{PM} - L(2 - \mathscr{Y}_{f, -1} - \mathscr{Y}_{f}) \forall \; , \; , f, \; \geq 2 \tag{13}$$

$$a^{2} \geq 0 \forall \tag{14}$$

$$A^{2} = P^{2} + {}_{2} a^{2} \forall \tag{15}$$

$$a^{2} \geq a^{2} + A^{2} - L(2 - \mathscr{Y}$$

maintenance is an additional service provided by the supplier when the machine is sold. From the supplier's perspective, the more maintenance is performed, the more additional revenue can be obtained. For this reason, it is assumed that the supplier will accept any maintenance plan presented by the manufacturer. In other words, the purpose of this study is to assist the manufacturer in determining the optimal production and maintenance plans of the DAHFSP-FPM targeted at the minimal makespan. Notations throughout this study are defined in Table 1.

To ensure the optimality of the proposed DAHFSP-FPM, a mixed integer linear programming (MILP) model with position-based maintenance decisions (i.e., PM is possible after each operation) is presented below.

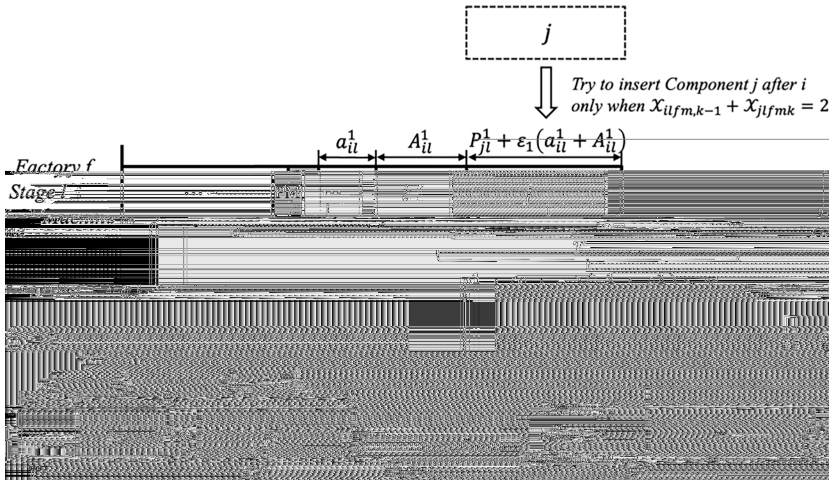

Fig. 2. Illustration of the sufficient condition for PM.

$$\sum \mathscr{Y}_{f} \leq 1 \forall f, \tag{29}$$

$$\sum \mathscr{Y}_{f} \leq \sum \mathscr{Y}_{f,-1} \forall f, \geq 2 \tag{30}$$

$$\sum \mathscr{Z}_{f} \leq 1 \forall f, \tag{31}$$

$$\sum \mathscr{Z}_{f} \leq \sum \mathscr{Z}_{f,-1} \forall f, \geq 2 \tag{32}$$

$$\sum \mathscr{Y}_{f} = \sum \sum_{j} \mathscr{X}_{f\,j} \forall \in \Omega, f \tag{33}$$

$$\sum \mathscr{Z}_{f} = \sum \sum_{j} \mathscr{X}_{f\,j} \forall \in \Omega, f \tag{34}$$

where the optimization objective is determined by (1) and (2), i.e., minimizing the makespan of the three-stage manufacturing process. Constraints (3) and (4) respectively represent the earliest starting time of each product at transportation and assembly stages. Constraint (5) shows that the earliest starting time of each component must be greater than or equal to the completion time of the previous operation of the component (if any). Constraints (6), (13) and (19) specify that the earliest starting time of each component (or product) must be more than or equal to the completion time of the previous component (or product) at the same machine (if any), in which if PM is performed immediately after the previous component (or product), the maintenance time is counted as part of the completion time of the previous component (or product). Constraint (7) initializes the earliest starting time at the production stage. Constraints (8), (14) and (20) ensure the initial machine's age as 0 at all the machines of the three-stage manufacturing process.

Regarding machine deterioration and maintenance, constraints (9), (15) and (21) are used to calculate the actual processing time considering linear deterioration effects at the production, transportation, and assembly stages respectively. Constraints (10), (16) and (22) respectively reflect the update of the machine's age under cumulative deteriorating effects without PM at the production, transportation, and assembly stages. Constraints (11), (17) and (23) demonstrate the perfect effect of PM activities at the above three stages, i.e., the implementation of PM can restore the machine's age to 0.

As for the relationship between decision variables, constraints (12), (18) and (24) specify that the maintenance decision prior to the first operation of any machine must be 0. Constraints (25), (33) and (34) guarantee that all components of one product must be assigned to the same factory. Constraint (26) represents that each operation can only be processed on one machine of one factory. Constraints (27), (29) and (31) ensure that each machine at different stages can process at most one operation at any time. Constraints (28), (30) and (32) show that there is no vacant position before a filled position of the same machine at different stages.

The MILP model has been validated by the CPLEX solver under small-scale cases. Due to the NP-hard nature of DAHFSP-FPM, a medium-scale case, e.g., six products, each of which consists of two to five components, and two factories, each of which has two stages and two to five machines per stage in the flow-shop production process, can hardly find an optimal solution in two hours. Although the production-maintenance joint scheduling plan derived in this way is theoretically optimal, finding the optimal solution in such a huge solution space is almost impossible using any optimization approach. As a result, we reduce the position-based maintenance decision to an efficient maintenance strategy, that is, the cumulative running time of the machine cannot exceed a predetermined value T. In this way, maintenance activities can be determined given a production sequence, avoiding a large number of maintenance decisions while ensuring maintenance periodicity. Hence, constraints (12), (18) and (24) need to be adjusted to the following constraints respectively. Fig. 2 illustrates the sufficient condition for maintenance execution with constraint (35) as an example.

$$\xi^1 = \begin{cases} 1, \text{if } P^1 + (1+\varepsilon_1)(a^1 + A^1) > T \\ \text{and } \mathscr{X}_{f,-1} + \mathscr{X}_{f\,j} = 2 \quad \forall, , f, , j \geq 2 \\ 0, \text{otherwise} \end{cases} \tag{35}$$

$$\xi^2 = \begin{cases} 1, \text{if } P^2 + (1+\varepsilon_2)\left(a^2 + A^2\right) > T \\ \text{and } \mathscr{Y}_{f,-1} + \mathscr{Y}_{f} = 2 \quad \forall, f, \geq 2 \\ 0, \text{otherwise} \end{cases} \tag{36}$$

$$\xi^3 = \begin{cases} 1, \text{if } P^3 + (1+\varepsilon_3)\left(a^3 + A^3\right) > T \\ \text{and } \mathscr{Z}_{f,-1} + \mathscr{Z}_{f} = 2 \quad \forall, f, \geq 2 \\ 0, \text{otherwise} \end{cases} \tag{37}$$

However, the simplified model considering the above constraints is still NP-hard, and the optimal solution can hardly be obtained in practice. To efficiently solve the simplified model, an MPMA and its Q-learning-based improvement are developed in the next section to find near-optimal solutions.

3. MPMA-QL for DAHFSP-FPM

The basic idea behind memetic algorithms (MAs) is combining evolutionary operators such as crossover and mutation with local search to achieve better performance than either approach alone. Different designs of evolutionary search and local search strategies correspond to different MAs. To enhance the search capability during the solving process of DAHFSP-FPM, an improved MA called MPMA-QL is specially designed in this study, where the multi-population strategy is applied to MA and Q-learning is introduced to adaptively adjust the individual quantity among multiple subpopulations. In general, the first three subsections introduce the main components of MPMA, followed by the Q-learning process, and the overall framework of MPMA-QL is given in the last subsection.

3.1. Encoding and decoding

In this study, a three-string encoding strategy including factory string (FS), product string (PS), and component string (CS) is introduced to represent the solution. FS is used to specify the factory to which each product is assigned. PS indicates the processing sequence for all products during the three-stage manufacturing process. Moreover, CS is used to represent the processing sequence for all components of each product.

Algorithm 2: Pseudo code of $internalCrossover(\cdot)$

Input: $s \in [1,2,\cdots,7]$, Π, C

Output: Π, C

1. F

Regarding the generation of the three-string encoding, PS and CS are completely randomly generated, while some FSs are generated using the following *Heuristic* to ensure the quality of the initial population and others are randomly generated to maintain population diversity. The pseudo code of the population initialization is given in **Algorithm 1**, where n denotes the population size.

Heuristic: The total time for each product to be manufactured in three consecutive stages without considering deterioration is calculated and sorted by the longest processing time first (LPT) rule, and then the sorted products are distributed to each factory in turn based on the randomly generated factory order.

An illustration of the three-string encoding with the DAHFSP-FPM in Fig. 1 as an example is presented in Fig. 3. It is clear that products 1, 4 and 6 are assigned to factory 1 in the order of 6–1-4, and the permutation of corresponding components is 21-22-2-3-1-16-13-15-14, while the other three products are assigned to factory 2 in the order of 2-3-5, and the permutation of corresponding components is 4-6-5-7-9-8-10-12-11-17-19-18-20. The three-step decoding process is defined in detail as follows.

The first step is the decoding of the production phase. The product manufacturing sequence and the factory assigned to each product are first determined based on PS and FS, and the machine with the earliest available time is assigned to each component of each product in turn according to the order of component codes of each product at each stage of hybrid flow-shop production under the corresponding factory. Moreover, the earliest start time of each component must satisfy constraints (5) and (6) in the MILP model. The second step is the decoding of the transportation phase. The earliest start time for each product is determined in order of the product code in turn. This depends on the maximum completion time for all components of that product, and the transportation completion time of the previous product, as shown in constraints (4) and (13). Similarly, constraints (3) and (19) are strictly satisfied in the decoding of the assembly phase.

Unlike previous studies such as Cai et al. (2022), the decoding process of components (or products) requires calculating the actual processing time and updating the machine's age based on the linear deterioration effect, as well as determining in real time whether the accumulated machine operation time exceeds a set threshold. If the threshold is exceeded (see Fig. 2), PM is performed to reset machine's age to 0 and the component (or product) is processed immediately afterwards; otherwise, the component (or product) can be processed directly.

3.2. Population division and exploration search

The idea of multi-population collaborative optimization is introduced to enhance the performance of exploration search in solving complex DAHFSP-FPM. The exploration search consists of crossover and mutation operations. Regarding crossover operations, we design seven crossover strategies based on the characteristics of three-level coding and these crossover strategies have their own advantages in different scenarios. Compared with a single crossover approach, the solutions generated by multiple crossover approaches correspond to different solution structures, which can avoid falling into the local optimum prematurely. As a consequence, the whole population with n individuals is divided into seven subpopulations with respective crossover strategies, in which the number of individuals in each subpopulation is relatively even and two crossover processes are performed using each crossover strategy. The details are presented as follows.

The first crossover strategy C_1 is dedicated to FS, as shown in Fig. 4. The first step is the crossover within a subpopulation, as shown in **Algorithm 2**. The best and worst individuals in the current subpopulation s are first determined, and one individual Π from the rest of the subpopulation is randomly selected as the optimized object. The codes with the same position as the worst individual are removed and the blanks are filled in order with reference to the coding order of the best individual, which is essentially a position-based crossover (PBX). Such an approach can guide individuals away from the poor solution and explore better neighborhood structures based on the current optimal individual. If the new solution after the above crossover is worse than Π, the PBX operation in Cai et al. (2022) is performed for Π and a random individual from the current subpopulation s.

The second step is the crossover between subpopulations, as presented in **Algorithm 3**. The subpopulation s^* with the global best solution Π^{b^*} is first determined and the worst solution Π^{w^*} of subpopulation s^* is also found. Then, Π^{b^*} and Π^{w^*} are used to guide the update of Π using the crossover strategy in subpopulation s^*. If the new solution after the above crossover is worse than Π, the PBX operation is executed for Π and a random individual from a random subpopulation s^{Δ}. Such an approach allows for interaction between subpopulations, which can effectively improve the structure of solutions.

Algorithm 3: Pseudo code of $externalCrossover(\cdot)$

Input: $s \in [1,2,\cdots,7]$, Π, C

Output: Π, C

1. Find the *subpopulation* [redacted]

The other six crossover strategies are similar to C_1 except that crossover operations are performed for different parts of the three-level code. C_2 is specifically designed for PS. C_3 is a separate operation for CS. C_4-C_7 perform multi-level crossover operations for the combinations of FS and PS, FS and CS, PS and CS, and FS and PS and CS, respectively.

After two rounds of crossover processes, two mutation mechanisms including NS_1 and NS_2 proposed by Cai et al. (2022) are randomly assigned to each individual, as shown in the following **Algorithm 4**.

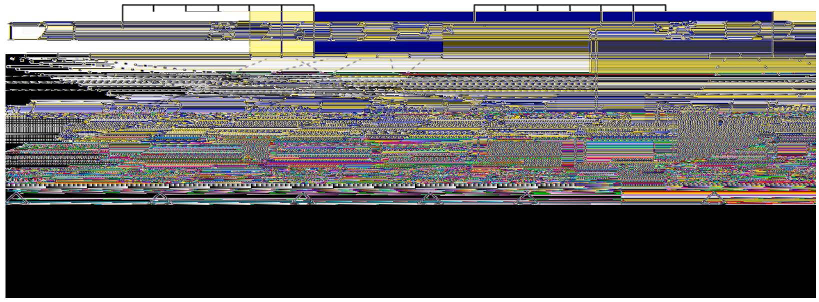

Fig. 3. Illustration of three-string representation.

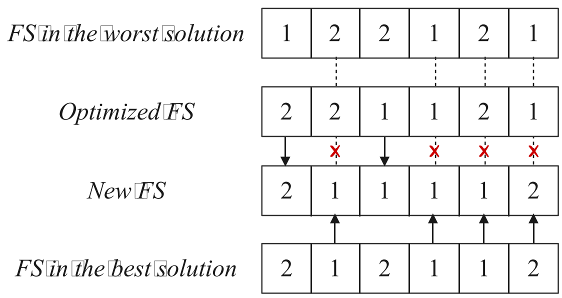

Fig. 4. Crossover illustration with FS as an example.

3.3. Knowledge-based exploitation search

Exploration search alone easily falls into local optima, so it is crucial to design knowledge-based exploitation search strategies to efficiently adjust the neighborhood structure of the solution. To improve the computational efficiency of MPMA, this study conducts three knowledge-based exploitation searches including LS_1, LS_2, LS_3 for the best individual of each subpopulation, as shown in **Algorithm 5**.

LS_1: Select one product from the factory with longer completion time (which is treated as the critical factory) and exchange it with one product from other factories. The above procedure is repeated five times. If C_{max} cannot be improved, the best individual from the five experiments is tried to replace the worst individual in the subpopulation.

LS_2: A product is randomly selected from PS and inserted sequentially into all possible positions to evaluate fitness values. There are P possible neighborhood structures, and thus the fitness is evaluated P times. By comparing the fitness values, the optimal insertion position of the product is found to ensure a better neighborhood structure.

LS_3: The component codes of each product are adjusted in a similar way to $LS2$. Specifically, one component is selected randomly from each product in turn and is inserted into the optimal position of the corresponding component code, and thus the total number of fitness assessments depends on the total number of components.

Algorithm 4: Pseudo code of $mutation(\cdot)$

Input: $s \in [1,2,\cdots,7]$, Π, C

Output: Π^*, C^*

if $rand() < 0.5$ **then**

Π' [redacted]

Algorithm 5: Pseudo code of $localSearch(\cdot)$

Input: $s \in \{1,2,\cdots,7\}$, Π^*, C^*

Output: Π^*, C^*

1. $\Pi' \leftarrow LS_I(\Pi^*)$
2. Decode the makespan C' of Π'
3. **if** $C' < C^*$ **then**
4. $\Pi^* \leftarrow \Pi'$; $C^* \leftarrow C'$
5. **else**
6. Find the worst solution Π^w of *subpopulation* s
7. Find the fitness C^w of Π^w
8. **if** C

Algorithm 6: Pseudo code of *Q-learning Update*$(\cdot)$

Input: ω_{min}, ω_{max}, C°, C^*,

3.4. Q-learning process

In the developed MPMA, there is a lack of adaptive adjustment of the number of individuals of each subpopulation. To further achieve effective information exchange between subpopulations and enhance the solving performance of MPMA, Q-learning is employed to dynamically adjust individual numbers of seven subpopulations instead of random adjustment. The procedure of the Q-learning update is given in **Algorithm 6**, in which $_{min}$, $_{max}$, C°, C^*, σ, a, Q, σ' and a' are defined in **Algorithm 7**. In addition, the definitions of state, action and reward in the Q-learning process are presented below.

State: System state is evaluated by the difference between the maximum value $_{max}$ and minimum value $_{min}$ of the number of individuals in each subpopulation. It can be found that the number of states is not fixed. If a new state σ' is generated during the Q-learning process that did not appear before, the state is added to the Q-table Q.

Action: Action set $\mathbb{A}$ is composed of three actions, i.e., increase the number of individuals of the subpopulation that generates more new solutions; decrease the number of individuals of the subpopulation that nu® generates more new * * א ǂ ή

Al

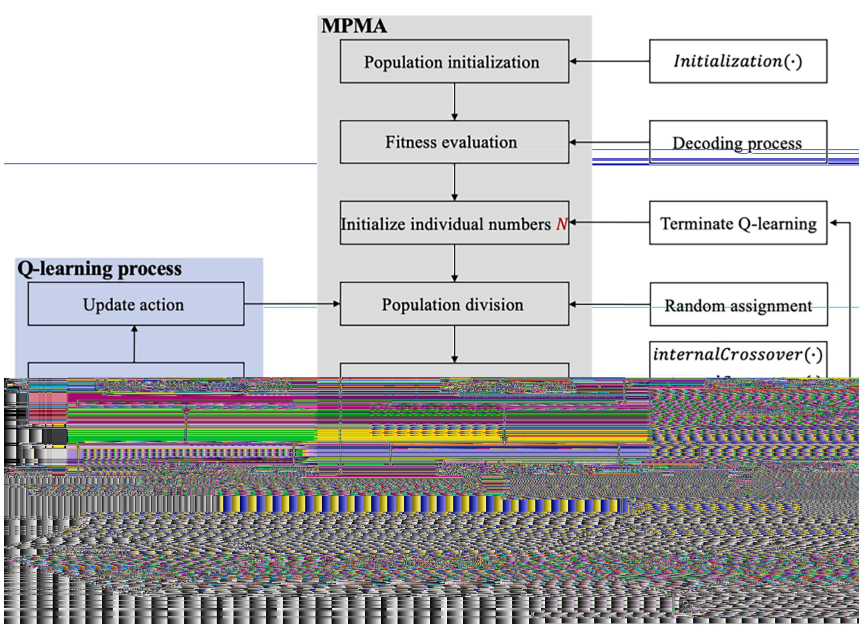

Fig. 5. Flow chart of MPMA-QL.

Table 2
Orthogonal experiment settings of MPMA-QL.

Trial number	Factor level					RV
	n	T		γ	ε	
1	40	100	0.1	0.7	0.1	1156.55
2	40	120	0.2	0.8	0.2	1159.45
3	40	150	0.3	0.9	0.3	1165.75
4	40	180	0.4	1	0.4	1170.46
5	60	100	0.2	0.9	0.4	1155.32
6	60	120	0.1	1	0.3	1160.09
7	60	150	0.4	0.7	0.2	1159.29
8	60	180	0.3	0.8	0.1	1170.84
9	80	100	0.3	1	0.2	1151.59
10	80	120	0.4	0.9	0.1	1159.04
11	80	150	0.1	0.8	0.4	1159.28
12	80	180	0.2	0.7	0.3	1175.28
13	100	100	0.4	0.8	0.3	1149.14
14	100	120	0.3	0.7	0.4	1149.37
15	100	150	0.2	1	0.1	1157.40
16	100	180	0.1	0.9	0.2	1163.39

Table 3
Response and rank of parameters for MPMA-QL.

Level	n	T		γ	ε
1	1163.05	1153.15	1159.83	1160.12	1160.96
2	1161.39	1156.99	1161.86	1159.68	1158.43
3	1161.30	1160.43	1159.39	1160.88	1162.57
4	1154.83	1169.99	1159.48	1159.88	1158.61
Delta	8.22	16.84	2.47	1.20	4.14
Rank	2	1	4	5	3

The difference between MPMA and MPMA-QL is mainly the Q-learning process as presented in **Algorithm 6**. The complexity of the Q-learning process is $O(3)$, since the only operation required is to obtain the maximum Q-value or a random one from $\mathbb{A}$ of size 3. As a result, the complexity overhead of MPMA-QL is only $O(3) = O(1)$ extra computations per generation when compared to MPMA. In fact, MPMA-QL may even achieve better results with even less computation time than MPMA, as the Q-learning process can assist the *meta*-heuristic algorithm to converge quickly. Experimental evidence for this fact is provided in Section 4.5.

4. Computational experiments

In this section, a series of computational experiments were conducted to evaluate the performance of the developed MPMA and MPMA-QL, in which two state-of-the-art *meta*-heuristics and their Q-learning-based improvements were selected as rivals. All algorithms were implemented in Python 3.8 and run on an Apple M1 CPU (3.20 GHz/ 8.00 GB RAM).

4.1. Test instance settings

To examine the algorithm performance for solving the proposed DAHFSP-FPM, 30 instances (depicted as $P \times F \times S$) were randomly generated based on the combination of $P \in \{10, 15, 20, 25, 30\}$, $F = \{2, 4\}$, $S \in \{2, 4, 6\}$, in which P^1_{il}, P^2_g and P^3_g were randomly taken integer values from the interval $[1, 100]$, each product consists of 2 to 5 components, and each stage of the hybrid flow shop consists of 2 to 5 parallel machines. Besides, it is assumed that deterioration rates and maintenance durations were known in advance: ε_1, ε_2 and ε_3 were set to 0.1, 0.05 and 0.15 respectively, and t^1_{PM}, t^2_{PM}, t^3_{PM} were all 10.

4.2. Performance metric

The relative percentage deviation (RPD) metric (Mao, Pan, Miao, & Gao, 2021) was introduced to measure the performance of MPMA-QL and five other competitive algorithms, which is defined as follows:

$$RPD = \frac{C_{a} \quad - C_{be}}{C_{be}} \tag{38}$$

where C_{alg} denotes the makespan obtained by a certain optimization algorithm on an instance, and C_{best} represents the optimal makespan among the results obtained by all the competing algorithms on that instance. Each algorithm under each test instance was carried out 10 times independently to achieve consistent and reliable results, reducing the variance caused by the randomness. Finally, the average RPD (*aRPD*), the best RPD (*bRPD*), and the standard deviation of RPD (*sRPD*) were calculated respectively to evaluate the solution quality of the algorithm.

4.3. Key parameter settings of MPMA-QL

There are five key parameters of MPMA-QL, i.e., population size n, upper limit of cumulative running time T and Q-learning-related three parameters , γ and ε. We selected four levels for each parameter to analyze the impact of different parameter configurations on the performance of MPMA-QL, i.e., $n = \{40, 60, 80, 100\}$, $T = \{100, 120, 150, 180\}$, $= \{0.1, 0.2, 0.3, 0.4\}$, $\gamma = \{0.7, 0.8, 0.9, 1\}$, $\varepsilon = \{0.1, 0.2, 0.3, 0.4\}$. There are a total of 4^5 parameter combinations. We picked an orthogonal array with 16 parameter combinations based on Taguchi's approach to lessen the complexity of the parameter analysis, where instance $20 \times 2 \times 6$ was chosen as the test instance. To assess the sensitivity of the above key parameters, MPMA-QL with each parameter combination was run 10 times, and the mean value of the makespan over ten independent runs was determined as the response variable (RV), as shown in Table 2. Besides, Table 3 shows the significant rank of parameter combinations, and then Fig. 6 intuitively shows the factor level trend of parameters.

From Table 3, it is obvious that T is the most significant parameter, which reflects that a proper maintenance cycle can greatly improve deteriorating effects. n plays the second most important role, which means that a proper population size can improve the solution performance of metaheuristics. Regarding Q-learning-related parameters, ε, and γ play the third, fourth and fifth roles respectively. Based on the RV results in Fig. 6, a promising parameter combination is suggested below: $n = 100$, $T = 100$, $= 0.3$, $\gamma = 0.8$, $\varepsilon = 0.2$, which will be used in the subsequent experiments.

Fig. 6. Factor level trend of MPMA-QL for each key parameter.

Table 4

Comparative results of six algorithms on *aRPD*, *bRPD*,.*sRPD*

Instance	SFLA			QSFLA			ABC			QABC			MPMA			MPMA-QL		
	aRPD	*bRPD*	*sRPD*	*aRPD*	*bRPD*	*sRPD*	*aRPD*	*bRPD*	*sRPD*	*aRPD*	*bRPD*	*sRPD*	*aRPD*	*bRPD*	*sRPD*	*aRPD*	*bRPD*	*sRPD*
10 × 2 × 2	0.0122	0.0021	0.0055	0.0112	**0.0000**	0.0058	0.0085	**0.0000**	0.0053	0.0099	0.0021	0.0048	0.0037	**0.0000**	0.0045	**0.0019**	**0.0000**	**0.0037**
10 × 2 × 4	0.0254	0.0132	0.0066	0.0137	0.0026	0.0066	0.0218	0.0090	0.0070	0.0191	0.0086	0.0075	0.0035	**0.0000**	0.0038	**0.0013**	**0.0000**	**0.0027**
10 × 2 × 6	0.0083	**0.0000**	0.0030	0.0052	**0.0000**	0.0031	0.0073	0.0016	**0.0023**	0.0054	**0.0000**	0.0034	0.0039	**0.0000**	0.0037	**0.0026**	**0.0000**	0.0036
10 × 4 × 2	0.0156	0.0013	0.0140	0.0160	**0.0000**	0.0104	0.0020	**0.0000**	0.0045	0.0056	**0.0000**	0.0059	**0.0009**	**0.0000**	**0.0017**	**0.0009**	**0.0000**	**0.0017**
10 × 4 × 4	0.0079	0.0036	0.0036	0.0077	**0.0000**	0.0058	0.0022	**0.0000**	0.0033	0.0007	**0.0000**	0.0015	**0.0000**	**0.0000**	**0.0000**	0.0004	**0.0000**	0.0011
10 × 4 × 6	0.0031	**0.0000**	0.0093	0.0019	**0.0000**	0.0038	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
15 × 2 × 2	0.0217	0.0117	0.0078	0.0088	**0.0000**	0.0063	0.0153	0.0049	0.0096	0.0201	0.0033	0.0091	0.0045	**0.0000**	0.0070	**0.0044**	**0.0000**	**0.0057**
15 × 2 × 4	0.0166	0.0054	0.0071	0.0083	**0.0000**	0.0076	0.0184	0.0097	0.0056	0.0140	**0.0000**	0.0096	0.0048	**0.0000**	0.0063	**0.0034**	**0.0000**	**0.0039**
15 × 2 × 6	0.0195	0.0057	0.0103	0.0135	0.0054	0.0037	0.0132	0.0001	0.0079	0.0137	0.0029	0.0054	0.0026	**0.0000**	0.0039	**0.0019**	**0.0000**	**0.0022**
15 × 4 × 2	0.0567	0.0196	0.0216	0.0332	0.0033	0.0177	0.0240	0.0063	0.0095	0.0183	0.0055	**0.0072**	0.0063	**0.0000**	0.0075	**0.0054**	**0.0000**	0.0092
15 × 4 × 4	0.0243	0.0040	0.0144	0.0171	**0.0000**	0.0123	0.0136	0.0019	0.0079	0.0110	**0.0000**	0.0072	0.0050	**0.0000**	0.0065	**0.0044**	**0.0000**	**0.0051**
15 × 4 × 6	0.0246	0.0117	0.0080	0.0181	**0.0000**	0.0127	0.0176	0.0089	0.0067	0.0164	**0.0000**	0.0073	0.0061	**0.0000**	0.0073	**0.0013**	**0.0000**	**0.0021**
20 × 2 × 2	0.0186	0.0084	0.0066	0.0105	**0.0000**	0.0084	0.0157	**0.0000**	0.0070	0.0143	**0.0000**	0.0106	0.0051	**0.0000**	**0.0058**	**0.0046**	**0.0000**	0.0059
20 × 2 × 4	0.0211	0.0084	0.0080	0.0153	0.0027	0.0109	0.0248	0.0079	0.0096	0.0179	0.0045	0.0101	0.0024	**0.0000**	**0.0020**	**0.0012**	**0.0000**	0.0026
20 × 2 × 6	0.0192	0.0107	0.0083	0.0157	**0.0000**	0.0111	0.0180	0.0012	0.0087	0.0193	0.0041	0.0100	0.0050	**0.0000**	0.0055	**0.0031**	**0.0000**	**0.0052**
20 × 4 × 2	0.0532	0.0228	0.0147	0.0298	0.0141	0.0164	0.0307	0.0090	0.0136	0.0316	0.0237	0.0101	0.0067	**0.0000**	0.0086	**0.0026**	**0.0000**	**0.0044**
20 × 4 × 4	0.0374	0.0110	0.0159	0.0251	0.0115	0.0118	0.0291	0.0151	0.0113	0.0229	0.0107	**0.0078**	0.0088	**0.0000**	0.0121	**0.0068**	**0.0000**	0.0092
20 × 4 × 6	0.0269	0.0046	0.0146	0.0249	0.0020	0.0119	0.0223	0.0022	0.0095	0.0226	0.0017	0.0107	0.0050	**0.0000**	0.0070	**0.0035**	**0.0000**	**0.0048**
25 × 2 × 2	0.0067	0.0022	0.0035	0.0078	0.0004	0.0055	0.0088	0.0043	0.0044	0.0091	0.0006	0.0051	0.0028	**0.0000**	0.0036	**0.0020**	**0.0000**	**0.0034**
25 × 2 × 4	0.0132	0.0047	0.0055	0.0089	0.0010	0.0066	0.0136	0.0040	0.0055	0.0153	**0.0000**	0.0093	0.0060	**0.0000**	0.0048	**0.0017**	**0.0000**	**0.0033**
25 × 2 × 6	0.0165	0.0040	0.0076	0.0139	0.0021	0.0074	0.0166	0.0097	0.0053	0.0139	**0.0000**	0.0101	0.0057	**0.0000**	0.0053	**0.0016**	**0.0000**	**0.0032**
25 × 4 × 2	0.0306	0.0107	0.0153	0.0219	0.0034	0.0144	0.0175	0.0043	0.0113	0.0216	0.0081	0.0102	0.0073	**0.0000**	0.0082	**0.0021**	**0.0000**	**0.0055**
25 × 4 × 4	0.0356	0.0228	0.0092	0.0217	**0.0000**	0.0095	0.0206	**0.0000**	0.0117	0.0245	0.0064	0.0105	0.0087	**0.0000**	0.0099	**0.0020**	**0.0000**	**0.0029**
25 × 4 × 6	0.0350	**0.0000**	0.0146	0.0231	0.0041	0.0113	0.0263	0.0127	0.0112	0.0217	**0.0000**	0.0124	0.0060	**0.0000**	**0.0077**	**0.0051**	**0.0000**	0.0085
30 × 2 × 2	0.0078	0.0026	0.0035	0.0090	0.0032	0.0033	0.0084	**0.0000**	0.0052	0.0108	0.0038	0.0049	0.0040	**0.0000**	0.0044	**0.0016**	**0.0000**	**0.0029**
30 × 2 × 4	0.0155	0.0076	0.0059	0.0155	0.0072	0.0050	0.0190	0.0010	0.0085	0.0164	0.0064	0.0056	0.0072	**0.0000**	0.0104	**0.0026**	**0.0000**	**0.0043**
30 × 2 × 6	0.0119	0.0061	0.0071	0.0083	**0.0000**	0.0065	0.0086	0.0037	0.0027	0.0065	0.0017	0.0038	0.0023	**0.0000**	**0.0025**	**0.0019**	**0.0000**	0.0030
30 × 4 × 2	0.0439	0.0163	0.0157	0.0340	0.0086	0.0137	0.0352	0.0226	0.0114	0.0255	0.0064	0.0152	0.0091	**0.0000**	0.0086	**0.0031**	**0.0000**	**0.0058**
30 × 4 × 4	0.0316	0.0144	0.0131	0.0277	0.0176	0.0106	0.0266	0.0100	0.0068	0.0250	**0.0000**	0.0131	0.0054	**0.0000**	0.0041	**0.0014**	**0.0000**	**0.0022**
30 × 4 × 6	0.0436	0.0094	0.0173	0.0316	0.0157	0.0105	0.0310	0.0103	0.0099	0.0322	0.0150	**0.0074**	0.0080	**0.0000**	0.0096	**0.0075**	**0.0000**	0.0087
Average	0.0235	0.0082	0.0099	0.0166	0.0035	0.0090	0.0172	0.0053	0.0074	0.0162	0.0039	0.0079	0.0049	**0.0000**	0.0057	**0.0027**	**0.0000**	**0.0042**

Table 5
Comparative results of six algorithms for all the instances grouped by P, F and S.

Groups of instances		P					F		S		
		10	15	20	25	30	2	4	2	4	6
SFLA	*aRPD*	0.0121	0.0272	0.0294	0.0229	0.0257	0.0156	0.0313	0.0267	0.0229	0.0209
	bRPD	0.0034	0.0097	0.0110	0.0074	0.0094	0.0062	0.0101	0.0098	0.0095	0.0052
	sRPD	0.0070	0.0115	0.0114	0.0093	0.0104	0.0064	0.0134	0.0108	0.0089	0.0100
QSFLA	*aRPD*	0.0093	0.0165	0.0202	0.0162	0.0210	0.0110	0.0223	0.0182	0.0161	0.0156
	bRPD	0.0004	0.0015	0.0051	0.0018	0.0087	0.0016	0.0054	0.0033	0.0043	0.0029
	sRPD	0.0059	0.0101	0.0118	0.0091	0.0083	0.0065	0.0115	0.0102	0.0087	0.0082
ABC	*aRPD*	0.0070	0.0170	0.0234	0.0172	0.0215	0.0145	0.0199	0.0166	0.0190	0.0161
	bRPD	0.0018	0.0053	0.0059	0.0058	0.0079	0.0038	0.0069	0.0051	0.0059	0.0050
	sRPD	0.0037	0.0079	0.0100	0.0082	0.0074	0.0063	0.0086	0.0082	0.0077	0.0064
QABC	*aRPD*	0.0068	0.0156	0.0214	0.0177	0.0194	0.0137	0.0186	0.0167	0.0167	0.0152
	bRPD	0.0018	0.0020	0.0075	0.0025	0.0056	0.0025	0.0052	0.0054	0.0037	0.0025
	sRPD	0.0039	0.0076	0.0099	0.0096	0.0083	0.0073	0.0084	0.0083	0.0082	0.0071
MPMA	*aRPD*	0.0020	0.0049	0.0055	0.0061	0.0060	0.0042	0.0056	0.0050	0.0052	0.0045
	bRPD	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
	sRPD	0.0023	0.0064	0.0068	0.0066	0.0066	0.0049	0.0066	0.0060	0.0060	0.0053
MPMA-QL	*aRPD*	**0.0012**	**0.0035**	**0.0036**	**0.0024**	**0.0030**	**0.0024**	**0.0031**	**0.0029**	**0.0025**	**0.0029**
	bRPD	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
	sRPD	**0.0021**	**0.0047**	**0.0054**	**0.0045**	**0.0045**	**0.0037**	**0.0047**	**0.0048**	**0.0037**	**0.0041**

(a) P=10 *(b) P=15*

(c) F=2 *(*

Fig. 7. Mean plots of different groups on the test instances regarding *aRPD*, *bRPD*, *sRPD*.

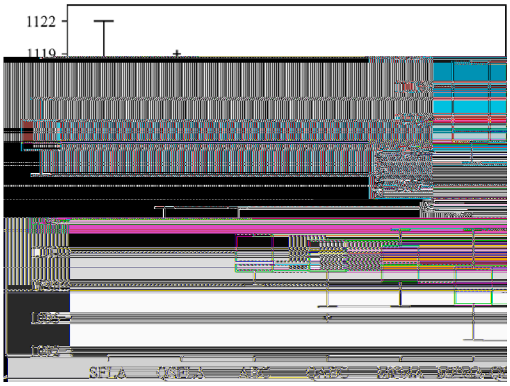

Fig. 8. Boxplot of six algorithms on makespan.

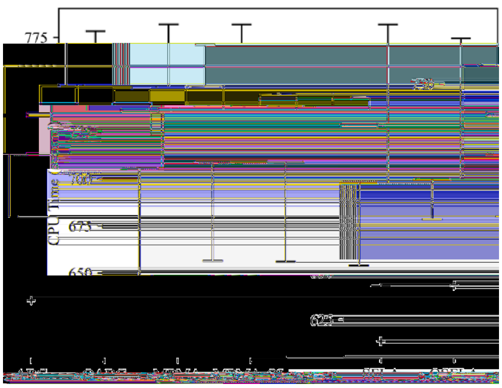

Fig. 9. Boxplot of six algorithms on CPU time.

4.4. Algorithm comparison and analysis for DAHFSP-FPM

Four state-of-the-art optimization algorithms were selected as competitors of MPMA and MPMA-QL, which are shuffled frog-leaping algorithm (SFLA) and SFLA with Q-learning (QSFLA) (Cai et al., 2022), as well as artificial bee colony algorithm (ABC) and ABC with Q-learning (QABC) (Wang, Lei, et al., 2022). Due to the variability of the research questions, some adjustments to comparison algorithms were required. Besides, key parameters of algorithm rivals were re-analyzed to adapt the proposed DAHFSP-FPM. It is worth noting that the problem parameter T has been determined in Section 4.3 to have a significant advantage at 100, and therefore T is fixed to 100 in the following parametric analysis.

SFLA and QSFLA were proposed for solving a DAHFSP without considering machine deterioration and maintenance activities. For dealing with the proposed DAHFSP-FPM, actual processing time under linear deterioration effects instead of normal processing time as well as flexible PM activities were considered in the decoding process of SFLA and QSFLA. There are six key parameters in QSFLA, which covers all the parameters in SFLA. For convenience, the following analysis is performed only for QSFLA parameters. Levels of each key parameter in QSFLA were set as: the population size n in {30, 60, 90, 120, 150}, cluster number $\mathscr{S}$ in {2, 3, 5, 6, 10}, repeat times per search μ in {20, 30, 40, 50, 60}, learning rate in {0.1, 0.2, 0.3, 0.4, 0.5}, discount factor γ in {0.6, 0.7, 0.8, 0.9, 1}, greedy rate ϵ in {0.1, 0.2, 0.3, 0.4, 0.5}. Orthogonal experiment settings under instance $20 \times 2 \times 6$ and the significant rank of parameter combinations are presented in Table A1 and Table A2 in the Appendix, and the factor level trend of parameters is shown as Fig. A1. Hence, the parameter combination of QSFLA is suggested as: $n = 60$, $\mathscr{S} = 10$, $\mu = 60$, $= 0.4$, $\gamma = 0.6$, $\epsilon = 0.4$.

ABC and QABC were used to tackle a three-stage distributed parallel machine scheduling with PM. To solve DAHFSP-FPM by ABC, the encoding representation and decoding procedure of MPMA and search strategies of SFLA were employed. As for QABC, the maximum tardiness metric in the state is replaced with the makespan, and the action set is replaced using the one in QSFLA. Regarding the levels of each key parameter in QABC, the population size n in {20, 40, 60, 80, 100}, local search times R in {35, 45, 55, 65, 75}, $Limit$ in {n, $2n$, $3n$, $4n$, $5n$}, learning rate in {0.1, 0.2, 0.3, 0.4, 0.5}, discount factor γ in {0.6, 0.7, 0.8, 0.9, 1}, greedy rate ϵ in {0.1, 0.2, 0.3, 0.4, 0.5}. Orthogonal experiment settings under instance $20 \times 2 \times 6$ and the significant rank of parameter combinations are given in Table A3 and Table A4, and the factor level trend of parameters is shown as Fig. A2. Therefore, the parameter combination of QABC is determined as: $n = 100$, $R = 75$, $Limit = n$, $= 0.3$, $\gamma = 0.9$, $\epsilon = 0.2$.

To ensure fairness of algorithm competition, the same encoding and decoding methods were used, and the maximum number of fitness evaluations satisfying all algorithm convergence was selected as the

Fig. 10. The optimal schedule found by MPMA-QL under the real-life case.

same termination condition. Comparative results of six algorithms regarding *aRPD*, *bRPD* and *sRPD* are given in Table 4, in which optimal values are marked in bold.

First, it is clear that MPMA-QL outperforms SFLA and QSFLA in terms of *aRPD* and *sRPD* under all the instances. In terms of *bRPD*, SFLA finds the same optimal value as MPMA-QL under three instances, while QSFLA is comparable to MPMA-QL in strength under 13 instances. Second, by comparing MPMA-QL with ABC and QABC in terms of *aRPD* and *bRPD*, it can be seen that MPMA-QL obtained better optimization results under all the instances. ABC showed equivalent performance on only one instance in terms of *aRPD* and on 7 instances in terms of *bRPD*. Besides, QABC exhibited equivalent results on only one instance in terms of *aRPD* and on 12 instances in terms of *bRPD*. In terms of *sRPD*, MPMA-QL revealed its superiority over ABC on 28 out of 30 instances and over QABC on 26 out of 30 instances. The next is the comparison between MPMA and MPMA-QL. In terms of *aRPD*, MPMA is better than MPMA-QL under one instance and is comparable to MPMA-QL in strength under 2 instances. In terms of *bRPD*, both of them achieved the optimum under all the instances. In terms of *sRPD*, MPMA-QL revealed its superiority over MPMA on 24 out of 30 instances, while MPMA achieved better results on the remaining 6 instances as well as exhibited equivalent results on two other instances.

In general, the average *aRPD* values of all the instances obtained by SFLA, QSFLA, ABC, QABC, MPMA, and MPMA-QL are 0.0235, 0.0166, 0.0172, 0.0162, 0.0049, and 0.0027 respectively; the corresponding average *bRPD* values are 0.0082, 0.0035, 0.0053, 0.0039, 0.0000 and 0.0000 respectively; the corresponding average *sRPD* values are 0.0099, 0.0090, 0.0074, 0.0079, 0.0057, and 0.0042 respectively. Besides, all the instances were grouped by P, F and S to analyze the experimental results in further, as shown in Table 5, in which optimal values are marked in bold. For more intuitive comparison, Fig. 7 shows mean plots of four groups of $P = 10$, $P = 15$, $F = 2$, $S = 2$ in terms of *aRPD*, *bRPD* and *sRPD*. Obviously, it can be concluded that MPMA-QL has an excellent performance over five other competing algorithms.

From the above statistics, some additional conclusions are given as follows. On the one hand, Q-learning can assist the original metaheuristic algorithm to find better solutions and improve the stability of the algorithm under most scenarios. On the other hand, the performance

heavily on the performance of the metaheuristic algorithm. Therefore, it is still crucial to design efficient metaheuristic algorithms in combination with problem features.

A real-world scenario from a furniture company given by Cai et al. (2022) was introduced to test the performance of six algorithms on DAHFSP without PM. This real-life example is described in detail as follows. There are two factories that collaborate to manufacture four different types of cabinets. Each cabinet is constructed from the respective 20 components when they are processed and transferred to the assembly machine. During the component production phase, there are five stages including punching, bending, welding, power pressing and drilling, and each stage consists of 2 to 3 parallel machines. All relevant data are fully referenced to Cai et al. (2022).

When the deterioration factors $_1$, $_2$ and $_3$ are set 0.2 and h $7m$ hom c q

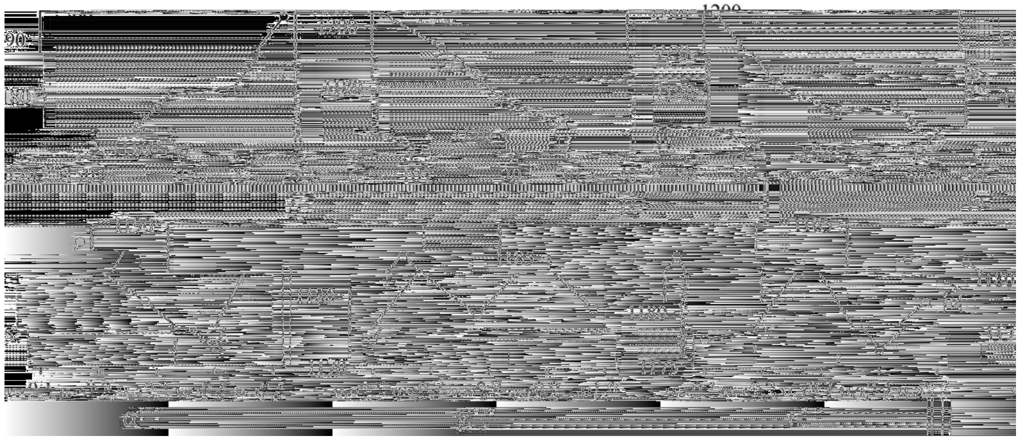

Fig. A1. Factor level trend of QSFLA for each key parameter.

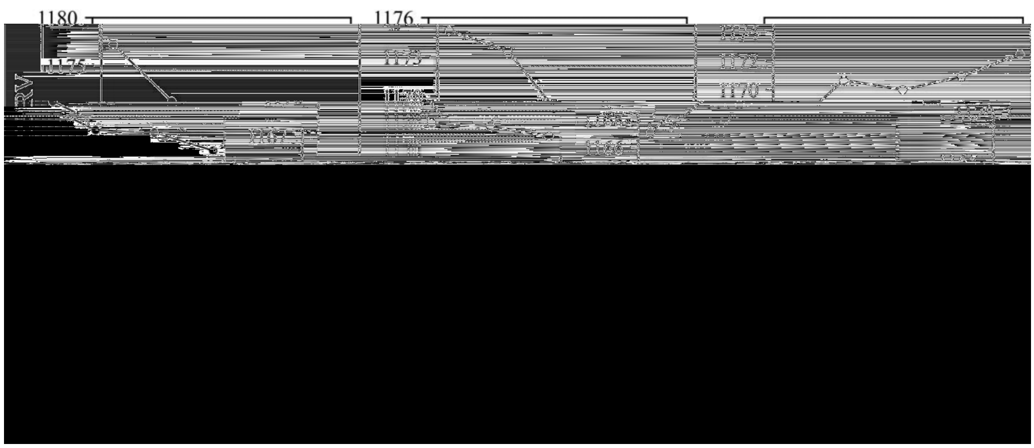

Fig. A2. Factor level trend of QABC for each key parameter.

Table A1
Orthogonal experiment settings of QSFLA.

Trial number	Factor level						RV
	n	$\mathscr{S}$	μ		γ	[illegible]	
1	30	2	20	0.1	0.6	0.1	1209.58
2	30	3	40	0.4	1	0.2	1171.55
3	30	5	60	0.2	0.9	0.3	1166.56
4	30	6	30	0.5	0.8	0.4	1159.05
5	30	10	50	0.3	0.7	0.5	1151.13
6	60	2	60	0.4	0.8	0.5	1175.05
7	60	3	30	0.2	0.7	0.1	1189.46
8	60	5	50	0.5	0.6	0.2	1159.14
9	60	6	20	0.3	1	0.3	1180.94
10	60	10	40	0.1	0.9	0.4	1151.60
11	90	2	50	0.2	1	0.4	1198.43
12	90	3	20	0.5	0.9	0.5	1224.71
13	90	5	40	0.3	0.8	0.1	1165.09
14	90	6	60	0.1	0.7	0.2	1159.35
15	90	10	30	0.4	0.6	0.3	1155.59
16	120	2	40	0.5	0.7	0.3	1213.76
17	120	3	60	0.3	0.6	0.4	1170.05
18	120	5	30	0.1	1	0.5	1179.21
19	120	6	50	0.4	0.9	0.1	1162.87
20	120	10	20	0.2	0.8	0.2	1178.58
21	150	2	30	0.3	0.9	0.2	1222.79
22	150	3	50	0.1	0.8	0.3	1208.95
23	150	5	20	0.4	0.7	0.4	1203.40
24	150	6	40	0.2	0.6	0.5	1174.48
25	150	10	60	0.5	1	0.1	1160.58

Table A2
Response and rank of parameters for QSFLA.

Level	n	$\mathscr{S}$	μ		γ	ϵ
1	1171.57	1203.92	1199.44	1181.74	1173.77	1177.52
2	1171.24	1192.94	1181.22	1181.50	1183.42	1178.28
3	1180.63	1174.68	1175.30	1178.00	1177.34	1185.16
4	1180.89	1167.34	1176.10	1173.69	1185.71	1176.50
5	1194.04	1159.49	1166.32	1183.45	1178.14	1180.92
Delta	22.80	44.43	33.12	9.76	11.94	8.65
Rank	3	1	2	5	4	6

Table A3
Orthogonal experiment settings of QABC.

Trial number	Factor level						RV
	n	R	*Limit*		γ	ϵ	
1	20	35	n	0.1	0.6	0.1	1178.52

References

Cai, J., Lei, D., Wang, J., & Wang, L. (2022). A novel shuf f ed frog-leaping algorithm with reinforcement learning for distributed assembly hybrid f ow shop scheduling. *International Journal of Production Research, 61*, 1233–1251.

Du, Y., Li, J., Li, C., & Duan, P. (2022). A reinforcement learning approach for f exible job shop scheduling problem with crane transportation and setup times. *IEEE Transactions on Neural Networks and Learning Systems*, 1–15.

Framinan, J. M., Perez-Gonzalez, P., & Fernandez-Viagas, V. (2019). Deterministic assembly scheduling problems: A review and classi f cation of concurrent-type scheduling models and solution procedures. *European Journal of Operational Research, 273*, 401–417.

Fu, Y., Hou, Y., Wang, Z., Wu, X., Gao, K., & Wang, L. (2021). Distributed scheduling problems in intelligent manufacturing systems. *Tsinghua Science and Technology, 26*, 625–645.

Guo, L., Zhuang, Z., Huang, Z., & Qin, W. (2020). Optimization of dynamic multi-objective non-identical parallel machine scheduling with multi-stage reinforcement learning. *IEEE International Conference on Automation Science and Engineering*, 1215–1219.

Komaki, G., Sheikh, S., & Malakooti, B. (2019). Flow shop scheduling problems with assembly operations: A review and new trends. *International Journal of Production Research, 57*, 2926–2955.

Lee, J.-H., & Kim, H.-J. (2022). Reinforcement learning for robotic f ow shop scheduling with processing time variations. *International Journal of Production Research, 60*, 2346–2368.

Lei, D., Su, B., & Li, M. (2021). Cooperated teaching-learning-based optimisation for distributed two-stage assembly f ow shop scheduling. *International Journal of Production Research, 59*, 7232–7245.

Li, H., Gao, K., Duan, P., Li, J., & Zhang, L. (2022). An improved arti f cial bee colony algorithm With Q-Learning for solving permutation f ow-shop scheduling problems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 1–10.

Li, J.-Q., Song, M.-X., Wang, L., Duan, P.-Y., Han, Y.-Y., Sang, H.-Y., & Pan, Q.-K. (2019). Hybrid arti f cial bee colony algorithm for a parallel batching distributed f ow-shop problem with deteriorating jobs. *IEEE Transactions on Cybernetics, 50*, 2425–2439.

Li, R., Gong, W., & Lu, C. (2022). A reinforcement learning based RMOEA/D for bi-objective fuzzy f exible job shop scheduling. *Expert Systems with Applications, 203*, Article 117380.

Li, Y.-Z., Pan, Q.-K., Ruiz, R., & Sang, H.-Y. (2022). A referenced iterated greedy algorithm for the distributed assembly mixed no-idle permutation f owshop

scheduling problem with the total tardiness criterion. *Knowledge-Based Systems, 239*, Article 108036.
Lohmer, J., & Lasch, R. (2021). Production planning and scheduling in multi-factory production networks: A systematic literature review. *International Journal of Production Research, 59*, 2028–2054.
Mao, J.-Y., Pan, Q.-K., Miao, Z.-H., & Gao, L. (2021). An effective multi-start iterated greedy algorithm to minimize makespan for the distributed permutation flowshop scheduling problem with preventive maintenance. *Expert Systems with Applications, 169*, Article 114495.
Neufeld, J. S., Schulz, S., & Buscher, U. (2022). A systematic review of multi-objective hybrid flow shop scheduling. *European Journal of Operational Research, 309*, 1–23.
Ruiz Rodríguez, M. L., Kubler, S., de Giorgio, A., Cordy, M., Robert, J., & Le Traon, Y. (2022). Multi-agent deep reinforcement learning based Predictive Maintenance on parallel machines. *Robotics and Computer-Integrated Manufacturing, 78*, Article 102406.
Shao, W., Shao, Z., & Pi, D. (2020). Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem. *Knowledge-Based Systems, 194*, Article 105527.
Song, H.-B., Yang, Y.-H., Lin, J., & Ye, J.-X. (2023). An effective hyper heuristic-based memetic algorithm for the distributed assembly permutation flow-shop scheduling problem. *Applied Soft Computing, 135*, Article 110022.
Srai, J. S., Kumar, M., Graham, G., Phillips, W., Tooze, J., , … Ford, S., et al. (2016). Distributed manufacturing: Scope, challenges and opportunities. *International Journal of Production Research, 54*, 6917–6935.
Wang, H., Sarker, B. R., Li, J., & Li, J. (2021). Adaptive scheduling for assembly job shop with uncertain assembly times based on dual Q-learning. *International Journal of Production Research, 59*, 5867–5883.
Wang, H., Yan, Q., & Zhang, S. (2021). Integrated scheduling and flexible maintenance in deteriorating multi-state single machine system using a reinforcement learning approach. *Advanced Engineering Informatics, 49*, Article 101339.
Wang, J., Lei, D., & Cai, J. (2022). An adaptive artificial bee colony with reinforcement learning for distributed three-stage assembly scheduling with maintenance. *Applied Soft Computing, 117*, Article 108371.
Wang, X., Ren, T., Bai, D., Ezeh, C., Zhang, H., & Dong, Z. (2022). Minimizing the sum of makespan on multi-agent single-machine scheduling with release dates. *Swarm and Evolutionary Computation, 69*, Article 100996.
Yang, S., Wang, J., & Xu, Z. (2022). Real-time scheduling for distributed permutation flowshops with dynamic job arrivals using deep reinforcement learning. *Advanced Engineering Informatics, 54*, Article 101776.
Zhang, Z., & Tang, Q. (2021). Integrating flexible preventive maintenance activities into two-stage assembly flow shop scheduling with multiple assembly machines. *Computers and Industrial Engineering, 159*, Article 107493.
Zhang, Z.-Q., Hu, R., Qian, B., Jin, H.-P., Wang, L., & Yang, J.-B. (2022). A matrix cube-based estimation of distribution algorithm for the energy-efficient distributed assembly permutation flow-shop scheduling problem. *Expert Systems with Applications, 194*, Article 116484.
Zhao, F., Di, S., Wang, L., Xu, T., Zhu, N., et al. (2022). A self-learning hyper-heuristic for the distributed assembly blocking flow shop scheduling problem with total flowtime criterion. *Engineering Applications of Artificial Intelligence, 116*, Article 105418.
Zhao, F., Xu, Z., Wang, L., Zhu, N., Xu, T., & Jonrinaldi. (2022). A population-based iterated greedy algorithm for distributed assembly no-wait flow-shop scheduling problem. *IEEE Transactions on Industrial Informatics*, 1–12.
Zhao, Z., Zhou, M., & Liu, S. (2021). Iterated greedy algorithms for flow-shop scheduling problems: A tutorial. *IEEE Transactions on Automation Science and Engineering, 19*, 1941–1959.