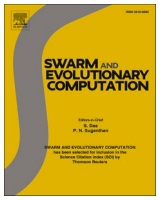
Swarm and Evolutionary Computation 80 (2023) 101338

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

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



A novel Q-learning based variable neighborhood iterative search algorithm for solving disassembly line scheduling problems

Yaxian Ren [a], Kaizhou Gao [a,b,*], Yaping Fu [c,*], Hongyan Sang [a], Dachao Li [a], Zile Luo [b]

[a] School of Computer Science, Liaocheng University, Liaocheng 252000, China
[b] Macau Institute of Systems Engineering, Macau University of Science and Technology, Taipa 999078, Macao
[c] School of Business, Qingdao University, Qingdao 266071, China

ARTICLE INFO

Keywords:
Disassembly line scheduling problem
Q-learning
Smoothing index
Local search

ABSTRACT

This paper addresses disassembly line scheduling problems (DLSP) to minimize the smoothing index with the workstation number threshold. First, a mathematical model is developed to formulate the concerned problems. Second, seven novel neighborhood structures are designed based on the feature of the DLSP and the corresponding local search operators are designed. Third, a novel Q-Learning based variable neighborhood iterative search (Q-VNIS) algorithm is first proposed to solve the DLSP. Q-learning is employed to select the premium local search operator in each iteration. Finally, the effectiveness of Q-learning in the proposed Q-VNIS is verified. To test the performance of the proposed Q-VNIS, 20 cases with different scales are solved and the Friedman test is executed. The experimental results and discussions show that the proposed Q-VNIS competes strongly for solving the DLSP.

1. Introduction

With global competition and environmental governance demands, recycling of end-of-life (EOL) products has attracted widespread attention. Due to technological innovation and product upgrading, product life is shortened. These EOL products waste resources and pollute the environment [1]. EOL product disassembly plays a key role in the recycling economy [2]. Disassembly line planning and scheduling can effectively improve the efficiency and effectiveness of the disassembly procedure [3], which is called disassembly line scheduling problems (DLSP). In recent years, more and more researchers have focused on disassembly line scheduling and planning problems. Kalayci et al. [4] first propose the sequential interdependent disassembly line balance problem and then construct a corresponding mathematical optimization model. Özceylan et al. [5] conducted a review of the literature on disassembly balancing issues. By summarizing and evaluating the existing research on disassembly scheduling, the research achievements and current research situation are analyzed, and the ongoing research directions are discussed. Meng et al. [6] proposed a mixed integer linear programming model to solve the disassembly planning problems for EOL products. A bucket brigade disassembly line balancing optimization method with uncertainty is proposed by Tian et al. [7], in which a cloud model is used to represent the uncertain disassembly time. Wang et al. [8] investigated the new problem of robot parallel disassembly sequence planning and developed a corresponding multi-objective model to minimize the completion time and energy consumption. Guo et al. [9] proposed a lexicographic multi-objective scatter search method to solve the multi-objective optimization model. To solve the problem of disassembly production line balance under limited distributional information, Liu et al. [10] proposed a cutting plane method. More economic benefits are obtained in [9–14] by reducing the cost of the disassembly process or disassembly losses. In [15,16], a two-sided disassembly model has been developed to reduce energy consumption.

To solve the DLSP, many traditional mathematical methods based on different strategies are used [17–20]. Since the disassembly line balancing and scheduling problem is NP-complete [21], precise mathematical methods are not suitable for solving large-scale problems. Various meta-heuristics have been employed and improved for solving various scheduling problems successfully [22–30], including disassembly scheduling problems. Tseng et al. [31] apply a genetic algorithm

* Corresponding authors.
E-mail addresses: gaokaizh@aliyun.com (K. Gao), fuyaping0432@163.com (Y. Fu).

https://doi.org/10.1016/j.swevo.2023.101338
Received 5 November 2022; Received in revised form 21 April 2023; Accepted 18 May 2023
Available online 24 May 2023

to optimize DLSP, aiming to solve the disassembly sequence planning problem more effectively. Since then, some scholars have proposed more meta-heuristics to solve the disassembly scheduling problem [32–34]. A Pareto firefly algorithm is proposed in [35] to solve disassembly problems while considering the hazards involved in disassembly operations. The disassembly problems with multiple objectives are also solved by using meta-heuristics in [36–38]. Wang et al. [14] propose an artificial bee colony algorithm for solving partial disassembly while maximizing efficiency. Guo et al. [39] combined a variable neighborhood descent method with an artificial bee colony algorithm to solve the disassembly line balance problem. Gao et al. [40] summarize the achievements in meta-heuristics for solving disassembly line scheduling, planning, and balancing problems, and some future directions in meta-heuristics for solving DLSP are highlighted.

Meta-heuristics have been successfully employed and improved to solve various scheduling problems [41–43]. In recent years, reinforcement learning is also used to solve scheduling problems or integrated into meta-heuristics to improve meta-heuristics' convergence speed. For reinforcement learning solving scheduling problems, literature [44] summarized the design of states and actions, sorted out the scheduling algorithms based on reinforcement learning, discussed the fusion model of reinforcement learning and meta-heuristics, analyzed the problems in current research, and pointed out the future research directions about this topic. In [45], a hyper-heuristic with Q-learning is proposed to solve distributed blocking flow shop scheduling problems for energy savings. Using the historical information from the low-level heuristic feedback, suitable low-level heuristics are selected from a pre-designed set of low-level heuristics. Literature [46] proposed an artificial bee colony algorithm with Q-learning to solve distributed heterogeneous no-wait flowshop scheduling problems. A Q-learning mechanism is designed to select neighborhood structure through the empirical knowledge gained during the operation. A reinforcement learning based strategy is proposed to improve the performance of brain storm optimization algorithm in [47]. The Q-learning mechanism is introduced to guide strategy selection based on the historical information fed back from the corresponding strategies. In [48], machine learning techniques are integrated into meta-heuristics to solve combinatorial optimization problems and a new effective iterative greedy algorithm is developed based on reinforcement learning. Q-learning is employed to select appropriate perturbation operators in the search process. To solve flow shop scheduling problems, literature [49] combines reinforcement learning and the ABC algorithm. In this study, we combine a reinforcement learning algorithm, Q-learning, and variable neighborhood search, to develop an iterative search algorithm for solving the DLSP.

The scientific contributions of this study are shown as follows:

1) The model of DLSP with consideration of interference time is developed to minimize the smoothing index with the workstation number threshold.
2) Seven neighborhood structures are designed, and corresponding local search operators are developed based on the features of DLSP.
3) A Q-learning based variable neighborhood iterative search (Q-VNIS) algorithm is first proposed to solve the concerned problems.

The rest of this study is organized as follows. The problem is described in Section 2, and a mathematical model is developed for the smoothing index with workstation number thresholds. In Section 3, the proposed Q-VNIS algorithm is presented in detail. Section 4 reports the experimental results and comparisons. Finally, this study is concluded, and some future directions are suggested in Section 5.

2. Problem description and formulation

2.1. Problem description

Disassembly is a complete production process in which components may be output at various workstations. During the disassembly process, there are n tasks that should be assigned to m workstations for disassembly. To reduce the complexity of the problem, it is assumed that all workstations can disassemble any task. All workstations are the same. When the constraints are satisfied, we assign the disassembly tasks to a workstation on the disassembly line according to the disassembly order that satisfies the constraints. Once a job is assigned to a workstation, it cannot be transferred to another and should be completed there. Different disassembly sequences have different disassembly time values. The operating time of one workstation cannot exceed the cycle time of the workstation. The DLSP includes three tasks: 1) task sequencing; 2) assigning tasks to workstations; and 3) adjusting the task sequence to minimize the smoothing index under the workstation number threshold.

In DLSP, there may be some precedence relationships among disassembly tasks. Fig. 1 shows a disassembly task precedence diagram. Each part of a product is represented by node i. Each node represents a disassembly task. The number in the circle indicates the disassembly task i. The number outside the circle is the disassembly time of the task i. Solid arrows indicate task sequential relationships, while dashed arrows indicate mutual interference between tasks. The number next to the dotted line represents the interference time. As shown in Fig. 1, task 1 is the predecessor of tasks 2 and 3, while tasks 2 and 3 are the successors of task 1. Task 1 must be finished before tasks 2 and 3 can be executed. Taking tasks 2 and 3 as an example, if task 2 precedes task 3, task 3 will prevent task 2 from working in the most convenient way, resulting in 1 unit of interference time for task 2, and its actual disassembly time is $5 + 1 = 6$. Conversely, if task 3 is disassembled before task 2, task 2 will prevent task 3 from working in the most convenient way. The actual disassembly time of task 3 will be increased by 10 units of interference time. It makes the actual disassembly time of task 3 increasing to $4 + 10 = 14$. Fig. 2 shows the distribution of tasks in two different disassembly sequences on the workstation. Fig. 2(a) shows the distribution of tasks on the workstations before adjustment, while Fig. 2(b) shows the distribution of adjusted tasks on the workstations. It can be seen that the idle time in Fig. 2(a) is less than that in Fig. 2(b), but the interference time in Fig. 2(b) is shorter, and the number of open workstations is reduced. We have included all possible solutions of the example in Fig. 1 in the supplementary file.

In DLSP, the more workstations that are open, the more interference time there will be. It is important to minimize the number of workstations for a given task. The smoothing index essentially balances the processing time of the workstation, which can balance the workload and ensure the efficient operation of the workstation. The objective of this study is to minimize the smoothing index below the workstation number threshold.

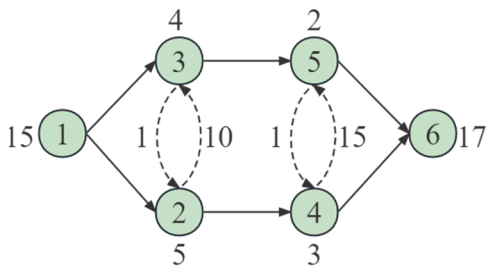


Fig. 1. Disassembly task precedence diagram.

2.2. Problem formulation

In this section, a mathematical model for the DLSP is developed with the goal of minimizing the smoothing index under the workstation number threshold. The following notations are employed to formulate the concerned problems.

Notations	Definition
(1) **Parameters**	
m:	Workstation threshold.
n:	Number of tasks. One disassembly task corresponds to one component, so also indicates the number of tasks.
k:	Index for workstation, $k \in \{1,2,\ldots,m\}$.
i,j:	Index for tasks (components), $i,j \in \{0,1,\ldots,n\}$.
CT:	Cycle time.
t_i:	The standard disassembly time for task i.
sd_{ji}:	Interference time, disassembly time increase for task i before task j.
u_{ij}:	If task i and task j have an interference relationship, then $u_{ij} = 1$, otherwise $u_{ij} = 0$.
D_{ij}:	If task i can be disassembled before task j, $D_{ij} = 1$, otherwise $D_{ij} = 0$.
G:	An infinite positive number.
(1) **Decision variables**	
w_{ij}:	If task i is disassembled before task j, $w_{ij} = 1$ otherwise, $w_{ij} = 0$.
x_{ik}:	If task i is assigned to workstation k, $x_{ik} = 1$, otherwise $x_{ik} = 0$.
y_{ikj}:	If task i is assigned to workstation k and disassembled before task j, $y_{ikj} = 1$, otherwise $y_{ikj} = 0$.
z_k:	If workstation k is opened, $z_k = 1$, otherwise $z_k = 0$.
t_i':	The actual disassembly time of task i, the sum of the standard disassembly time and the interference time.
B_i:	The disassembly start time of task i.
F_i:	The disassembly finish time of task i.
T_k:	The total operating time of workstation k.
IT_k:	The idle time of workstation k.

According to the notations described above, the mathematical model of the DLSP can be formulated as follows:

$$f = \text{Min} \sum_{k=1}^{m} z_k \times IT_k^2, \ \forall k \in \{1,2,\ldots,m\} \tag{1}$$

s.t.

$$w_{ij} + w_{ji} = 1, \ i \neq j, \forall i,j \in \{1,2,\ldots,n\} \tag{2}$$

$$w_{ii} = 0, \ \forall i \in \{0,1,\ldots,n\} \tag{3}$$

$$w_{ij} \leq D_{ij}, \ \forall i,j \in \{0,1,\ldots,n\} \tag{4}$$

$$t'_i \geq t_i + sd_{ji} \times w_{ij} \times u_{ij}, \ \forall i,j \in \{0,1,\ldots,n\} \tag{5}$$

$$F_i = B_i + t'_i, \ \forall i \in \{0,1,\ldots,n\} \tag{6}$$

$$B_j \geq F_i - G \times (1 - w_{ij}), \ \forall i \in \{0,1,\ldots,n\}, \ \forall j \in \{1,2,\ldots,n\} \tag{7}$$

$$\sum_{k=1}^{m} x_{ik} = 1, \ \forall i \in \{1,2,\ldots,n\} \tag{8}$$

$$w_{ij} = \sum_{k=1}^{m} y_{ikj}, \ \forall i,j \in \{0,1,\ldots,n\} \tag{9}$$

$$w_{ij} + x_{ik} \leq 1 + y_{ikj}, \ \forall i \in \{1,2,\ldots,n\}, \ \forall j \in \{0,1,\ldots,n\}, \forall k \in \{1,2,\ldots,m\} \tag{10}$$

$$T_k = \sum_{i=1}^{n} t_i \times x_{ik} + \sum_{i=1}^{n} \sum_{j=1}^{n} sd_{ji} \times u_{ij} \times y_{ikj}, \ \forall k \in \{1,2,\ldots,m\} \tag{11}$$

$$T_k \leq CT, \ \forall k \in \{1,2,\ldots,m\} \tag{12}$$

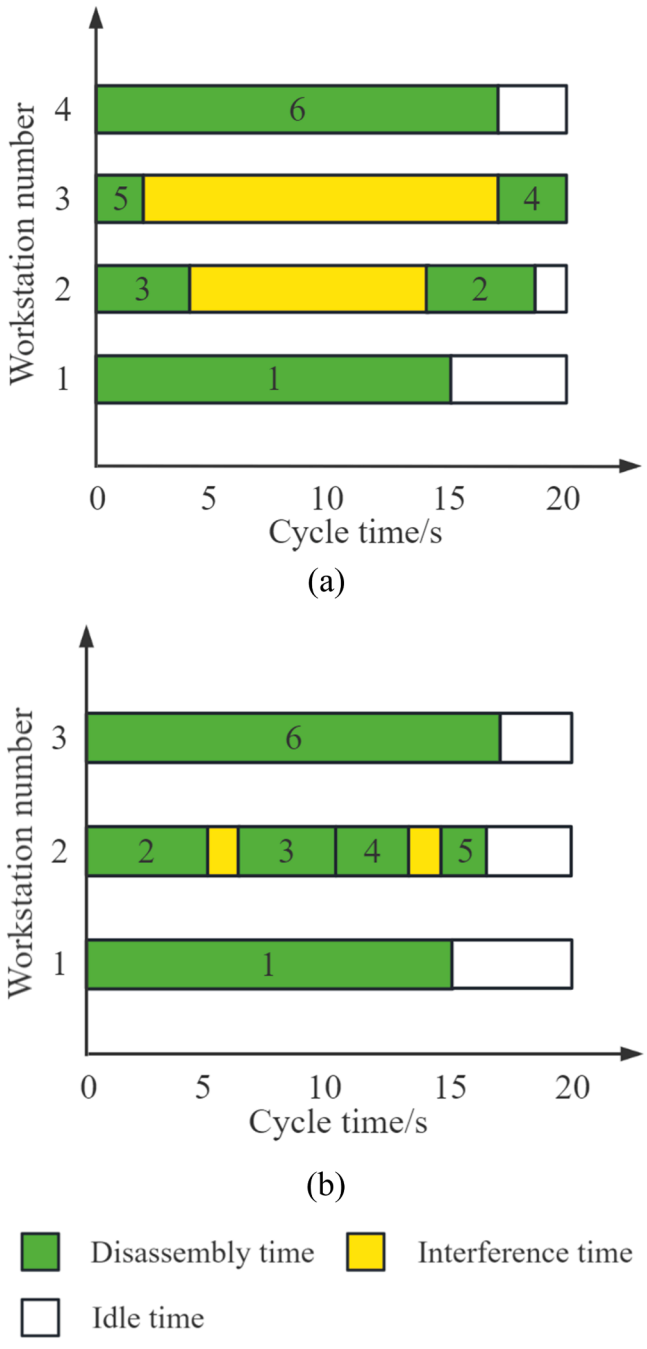


Fig. 2. Task assignment on workstations.

$$F_i - G^*(1 - x_{ik}) \leq k \times CT, \ \forall i \in \{0,1,\ldots,n\}, \ \forall k \in \{1,2,\ldots,m\} \tag{13}$$

$$F_i + G^*(1 - x_{ik}) \geq (k-1) \times CT, \ \forall i \in \{1,2,\ldots,n\}, \ \forall k \in \{1,2,\ldots,m\} \tag{14}$$

$$z_k \geq x_{ik}, \ \forall i \in \{1,2,\ldots,n\}, \ \forall k \in \{1,2,\ldots,m\} \tag{15}$$

$$z_k \leq \sum_{i=1}^{n} x_{ik}, \forall k \in \{1,2,\ldots,m\} \tag{16}$$

$$z_{k+1} \leq z_k, \ \forall k \in \{1,2,\ldots,m-1\} \tag{17}$$

$$IT_k \geq CT \times z_k - T_k, \ \forall k \in \{1,2,\ldots,m\} \tag{18}$$

$$w_{ij} \in \{0,1\},\ \forall i,j \in \{0,1,\ldots,n\} \tag{19}$$

$$x_{ik} \in \{0,1\},\ \forall i \in \{0,1,\ldots,n\},\ \forall k \in \{1,2,\ldots,m\} \tag{20}$$

$$y_{ikj} \in \{0,1\},\ \forall i,j \in \{0,1,\ldots,n\},\ \forall k \in \{1,2,\ldots,m\} \tag{21}$$

$$z_k \in \{0,1\},\ \forall k \in \{1,2,\ldots,m\} \tag{22}$$

$$t_i^{'} \geq 0, B_i \geq 0, B_0 = 0, F_i \geq 0, T_k \geq 0,\ IT_k \geq 0, \forall i \in \{0,1,\ldots,n\}, \forall k \in \{1,2,\ldots,m\} \tag{23}$$

Eq. (1) is the objective function, meaning the minimization of the smoothing index, and it is a nonlinear objective function. The smoothing index is used to balance the idle time of workstations and the number of tasks assigned to each workstation. In Eq. (1), the $\sum_{k=1}^{m} z_k \times IT_k^2$ is the smoothing index. IT_k^2 is the idle time for each workstation when the total idle time is the same. Constraint (2) represents that task i should be processed before task j, or task j should be processed before task i. Constraint (3) does not allow a task to come before or after itself. Constraint (4) indicates that the processing sequence of tasks needs to satisfy the disassembly priority relationship. The $D_{ij} = 1$ represents that task i can be disassembled before task j, and w_{ij} can be either 1 or 0; otherwise, w_{ij} is equal to 0. Constraint (5) means that the actual disassembly time is the sum of the standard time and the interference time. Constraint (6) stipulates that the end time of a task is equal to the sum of the start time and the actual disassembly time of the task. Constraint (7) implies that the start time of the subsequent task must be greater than or equal to the end time of the previous task. Constraint (8) defines that each task can be assigned to only one workstation. Constraints (9) and (10) contact x_{ik}, w_{ij}, and y_{ikj}, denoting that if y_{ikj} is equal to 1, task i is assigned to workstation k, and task j is performed after task i. Constraints (11) and (12) denote the actual running time of the workstation. Constraints (13) and (14) explain the assignment of tasks to different workstations. Constraint (15)-(17) ensures that workstations are opened in sequence. Constraint (18) calculates the idle time for each workstation. Constraints (19) to (23) give the sign restrictions of decision variables.

3. The proposed Q-VNIS

3.1. Solution representation

In this study, a simple solution representation scheme is used. A sequence of integers formed by arranging the serial index of the disassembly tasks is a solution to the concerned problem. But the solution is not always feasible. A feasible solution must satisfy the components' disassembly precedence constraint. An unfeasible solution can be adjusted to a feasible one by using the priority matrix. For example, let $v = (6,3,2,5,4,1)$ be an integer permutation of the case in Fig. 1, and v contains all the tasks index. The order from left to right indicates the order of task disassembly. As shown in Fig. 1, task 1 should be disassembled first, followed by task 2/3, task 2 before task 4, task 3 followed by task 5, and finally, task 6. Hence, v can be adjusted to $v^{'} = (1,3,2,5,4,6)$, where task 1 is disassembled first, then task 3, from left to right, in the order of the operations. After executing task 6, the disassembly process is complete. The above encoding strategy considers operation sequence and priority relationships simultaneously.

Furthermore, a decoding strategy is designed. Under the premise that the workstation operation time does not exceed the cycle time, the tasks in the adjusted sequence are assigned to the workstations in turn. Taking the sequence $v^{'} = (1,3,2,5,4,6)$ as an example, we assume that the cycle time $CT = 20\ s$ (s). The decoding process is as follows: turn on workstation 1 to assign task 1. The total operating time (T_1) of workstation 1 is 15 s, the idle time IT_1 of workstation 1 is 5 s. The following one is task 3, which is disturbed by task 2, and its actual disassembly time $t^{'}_3 = t_3 + sd_{23} = 4 + 10 = 14 > IT_1 = 5$. We need to turn on workstation 2. Task 3 is assigned to workstation 2. The working time of workstation 2 is 14 s, and its idle time is 6 s. Next, task 2 comes, and its disassembly time is 5 s. Task 2 can be disassembled on workstation 2. At this time, the running time of workstation 2 is 19 s and the idle time is 1 s. The disassembly time of task 5 is 2 s, which is greater than the idle time (1 s) of workstation 2. Hence, workstation 3 is turned on, and task 5 is assigned to it. The decoding is finished when all tasks are assigned to workstations. The decoding result is shown in Fig. 2(a), and the workstation number and smoothing index of the disassembly sequence $v^{'}$ are: $m = 4, f = 50$, respectively. The second kind of infeasible solution is if the number of workstations turned on exceeds the workstation threshold, the solution will not be updated.

3.2. Neighborhood structures and local search operators

Based on the features of the DLSP, seven neighborhood structures are designed, and the corresponding local search operators are developed. The details of seven local search operators are described below.

1 Swap: In a solution, two tasks are randomly selected and their positions are swapped. An example is shown in Fig. 3(a), where the positions of tasks 6 and 1 are exchanged.

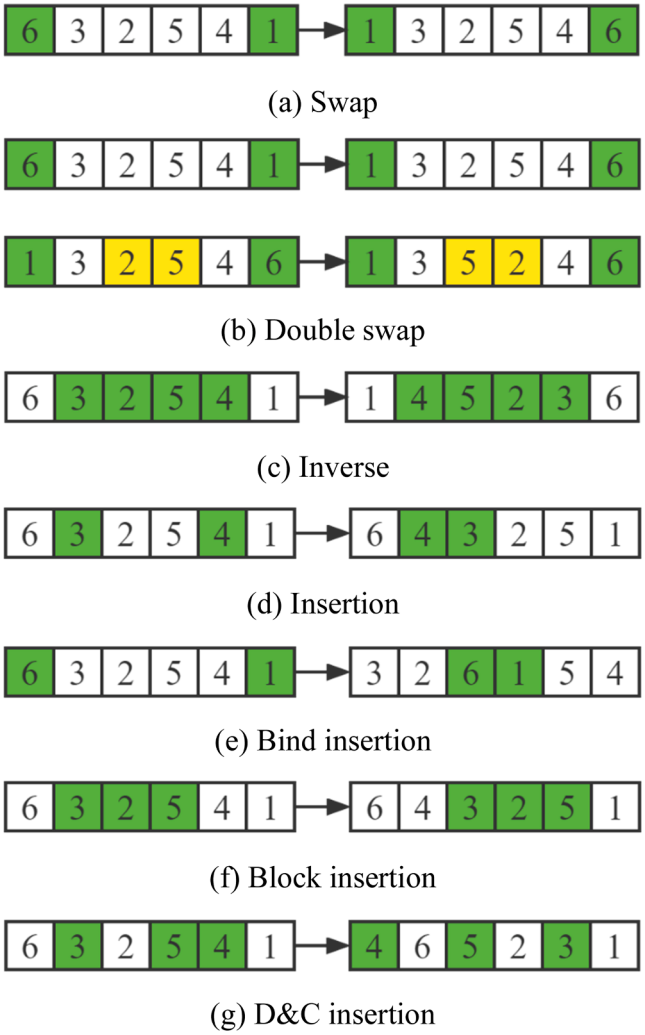


Fig. 3. Local search operators.

2 Double swap: Two swap operations are performed in one solution, as shown in Fig. 3(b).
3 Inverse: Randomly select two tasks in a solution and reverse the order of the two tasks and the intermediate tasks, as shown in Fig. 3 (c).
4 Insertion: Randomly select two tasks, determine the relationship between their positions, insert the latter task into the position of the former task, and move the tasks after the inserted position back by one position in turn, as shown in Fig. 3(d).
5 Bind insertion: Two tasks are selected randomly in a sequence, and they are taken as a whole and inserted into all possible positions in that sequence. The procedure is shown in Fig. 3(e).
6 Block insertion: Randomly select two different positions in a solution, bind the tasks between the two positions as a block, and insert the block into all possible positions in the solution (Fig. 3(f)).
7 D&C insertion: Several tasks are randomly deleted from a solution, and the deleted tasks are randomly inserted one by one into all possible positions in the solution. Fig. 3(g) depicts the sequences before and after D&C insertion.

Some new solutions may be infeasible for local search operators. The first kind of infeasible solution is that the newly generated sequence does not satisfy the precedence, it must be adjusted to meet the constraints (2)-(4). The second kind of infeasible solution is if the number of workstations turned on exceeds the workstation threshold, the solution will not be updated.

3.3. Q-Learning

Q-learning is a type of reinforcement learning [50]. In Q-learning, $Q(s,a)$ is the expected value of the reward that can be obtained after taking action a at state s. The Q-table records the Q-values of all combinations of states and actions. When a state s is given, the action with the largest Q-value is selected from the Q-table. Since the Q-table is randomly initialized, selecting the maximum value for the operation may prevent some values from being selected, making it impossible to update other values. To avoid this problem, ε-greedy is used to select operations. When the agent observes state s, the action corresponding to the largest Q-value is selected inside the Q-table with probability $1-\varepsilon$, and ε of the probability for the agent to randomly explore. Without entirely using a greedy algorithm, there is some chance of random selection, so that all actions in the Q-table are guaranteed to be updated in a sufficient number of iterations. The update strategy of the Q-value for one state action pair is as follows:

$$Q(s,a)\leftarrow Q(s,a)+\alpha[r+\gamma max_{a'}Q(s',a')-Q(s,a)] \tag{24}$$

where, s is the current state, while a represents the actual action. $Q(s,a)$ is the Q-value corresponding to the state-action operation, and each state-action has a unique corresponding $Q(s,a)$. The α refers to the learning rate of the iterative update. The discount factor γ takes values in the range $[0,1]$. $Q(s',a')$ represents the state at the next moment and the Q-value corresponds to the taken action. r is the bonus value obtained from the current episode.

The main advantage of Q-learning is the ability to learn offline using temporal-difference learning. Q-learning accumulates historical experience through continuous exploration of the episode, and the agent strengthens itself through continuous trial and error. The framework of Q-learning is shown in Fig. 4.

3.4. Q-Learning based VNIS

For solving the DLSP, a Q-learning based VNIS is proposed. The core to combine Q-learning and VNIS is to establish the mapping between the environment and agent in Q-learning and the concerned problem. In this study, the solutions are taken as the states, and the local search operators are the actions.

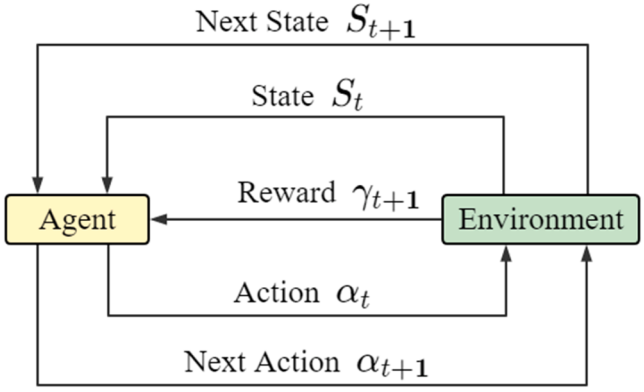


Fig. 4. The framework of Q-learning.

Table 1 is the Q-table in Q-learning. The rows represent solutions. The columns represent the seven proposed local search operators. We use Q-learning to select the local search operators (i.e., actions) that will be performed in the next step. If a local search operator is selected by a solution and performs well, its reward is added. The Q-value in the Q-table increases, and the probability of being selected also increases. Conversely, if a local search operator makes the solution worse, it is not rewarded. At the same time, its Q-value is reduced, and the selection probability is also reduced. By accumulating experience through continuous search, the solution can filter out more suitable local search operators for each solution to generate better solutions. This paper uses a roulette strategy for selecting local search operators. The greater the fitness, the better the selection probability of the corresponding solution is. Q-learning is employed to select the premium local search operator to improve the exploitation performance of the proposed Q-VNIS.

3.5. The framework of Q-VNIS

The procedures of the Q-VNIS are shown as follows:

Step 1. Initialize solutions and the Q-table. The Q-table is shown in Table 1. One solution corresponds to a state, and one action represents a local search operator. For example, solution 1 is the current state, and the swap represents the actual selected action. $Q(\mathrm{S1},\mathrm{Swap})$ is the state-action value. The values in the Q-table are set to 1 initially.

Step 2. Evaluate the objective values of the initialized solutions.

Step 3. The solutions are sorted in descending order based on their objective values. Select the local search operator in Q-learning. The detailed steps of action selection are shown in Algorithm 1.

Step 4. After executing the selected action, the Q-learning will send out a reward value R, which will update the corresponding Q-value in the Q-table. The detailed information for calculating the reward value is shown in Eq. (25).

Table 1
Q-table.

	Swap	Double swap	…	Block insertion	D&C insertion
S1	Q(S1, Swap)	Q(S1, Double swap)	…	Q(S1, Block insertion)	Q(S1, D&C insertion)
S2	Q(S2, Swap)	Q(S2, Double swap)	…	Q(S2, Block insertion)	Q(S2, D&C insertion)
…	…	…	…		
Sw	Q(Sw, Swap)	Q(Sw, Double swap)	…	Q(Sw, Block insertion)	Q(Sw, D&C insertion)

$$R = \begin{cases} \frac{f - f'}{10}, & f' < f \\ 1, & f' = f \\ 0, & f' > f \end{cases} \tag{25}$$

Step 5. The newly generated solutions are evaluated, and the solutions are updated accordingly.

Step 6. Output the results if the termination conditions are met. Otherwise, return to step 3.

To represent the steps of the proposed Q-VNIS more clearly, its framework is shown in Fig. 5.

4. Experiments and discussions

4.1. Experimental setup

To verify the effectiveness of the proposed Q-VNIS algorithm, 13 instances are solved. Cases 1–8 are obtained from [51]. Case 9 is obtained from [52], case 10 is obtained from [53] for aircraft auxiliary power units, and cases 11 and 12 are from [14]. Case 13 is an aircraft engine disassembly obtained from a company in our project. Cases 14–20 are examples that we randomly generated. The number of tasks in all cases is shown in Table 2. All the compared algorithms are coded in C++. The running platform is a computer with an Intel Core i7–10,700 CPU @ 2.90 GHz and 16 GB of RAM under Microsoft Windows 11.

To make a fair comparison, the running times of all algorithms for the same case are consistent, and all experiments are carried out 20 times independently, the average (Ave), maximum (Max), and minimum (Min) values of the results are counted. The coefficient of variation (CV) is used to evaluate the performance of the algorithms.

$$CV = \frac{SD}{AVE} \times 100\% \tag{26}$$

where, the AVE is the mean value in 20 runs, and the SD is the corresponding standard deviation.

Table 2
Number of tasks in case.

Case	Product name	Number of tasks
1	Case in [46]	6
2	Case in [46]	7
3	Computer mainframe	8
4	Truck	9
5	Case in [46]	10
6	Case in [46]	10
7	Case in [46]	12
8	Phone	25
9	Auxiliary power unit	16
10	Laptop	47
11	LCD TV	36
12	Refrigerator	66
13	Aircraft engine	51
14	New generated	16
15	New generated	27
16	New generated	36
17	New generated	37
18	New generated	47
19	New generated	51
20	New generated	66

4.2. Parameter settings

Q-VNIS has three parameters: the number of solutions (SN), α and γ. In order to obtain a better combination, the Orthogonal Experimental [54] design method is used to test and analyze the influence of these three parameters on the performance of Q-VNIS. Each parameter has four levels, $\in \{10, 20, 30, 40\}$, $\alpha \in \{0.6, 0.7, 0.8, 0.9\}$, $\gamma \in \{0.1, 0.2, 0.3, 0.4\}$. The orthogonal matrix $L_{16}(4^3)$ is shown in Table 3. Each set of parameters is solved by Q-VNIS, and the experimental results are analyzed with the smooth index (SI) results in 20 runs. According to the results in Table 3, the main effects plot is drawn shown in Fig. 6. It can be seen that Q-VNIS produces the best results with the parameters' values, $SN = 20$, $\alpha = 0.8$ and $\gamma = 0.1$, respectively. Therefore, the combination is applied to the following experiments.

4.3. Verification of the mathematical model

In this section, we firstly report the number of binary decision variables (NBVs), continuous decision variables (NCVs), and constraints

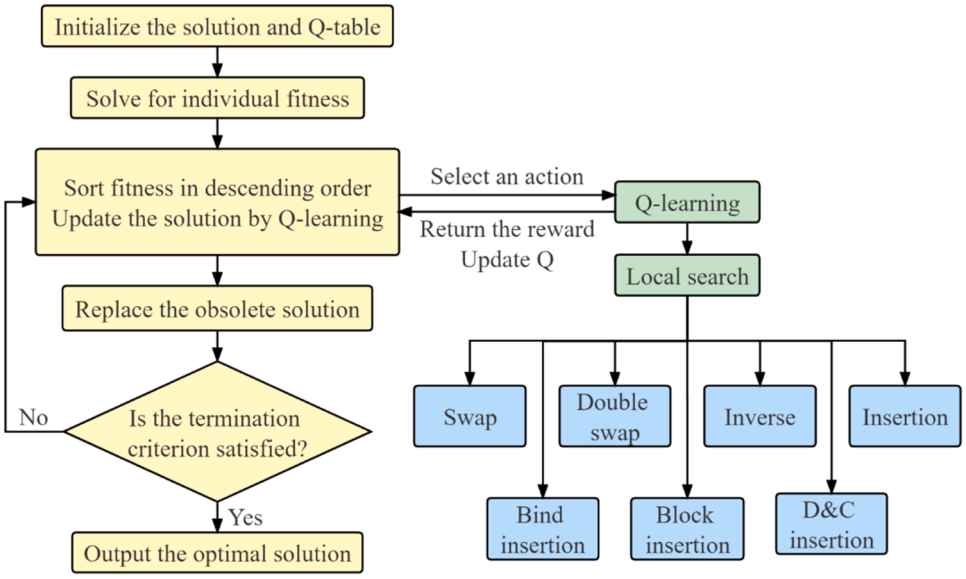


Fig. 5. The framework of the proposed Q-VNIS.

Table 3
Orthogonal experiments for parameter setting.

No.	SN	α	γ	SI
1	10	0.9	0.1	772.8
2	10	0.8	0.2	774.3
3	10	0.7	0.3	777.5
4	10	0.6	0.4	779.7
5	20	0.9	0.2	777.8
6	20	0.8	0.1	769.1
7	20	0.7	0.4	775.9
8	20	0.6	0.3	775.3
9	30	0.9	0.3	838.7
10	30	0.8	0.4	814.1
11	30	0.7	0.1	821.0
12	30	0.6	0.2	819.5
13	40	0.9	0.4	851.9
14	40	0.8	0.3	851.0
15	40	0.7	0.2	859.5
16	40	0.6	0.1	852.3

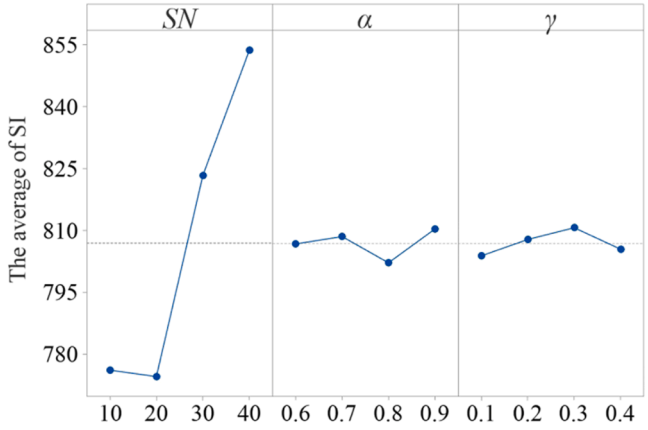


Fig. 6. The main effects plot for Q-VNIS.

(NCs) in established mathematical model, as shown in Table 4.

To verify the validity of the proposed mathematical model, this work employs the Gurobi solver to solve partial cases with different scales and the results are shown in Table 5, where n represents the number of tasks and SI denotes the objective value. The gap is the percentage difference between the solutions obtained by Q-VNIS and Gurobi. As can be seen from Table 5, both Gurobi and Q-VNIS can obtain optimal solutions in a shorter time when the scale of cases is small (cases with $n = 6$ and $n = 12$). As case scale increasing, the running time of Gurobi gradually increases and is much larger than that of Q-VNIS. For the cases with 16 tasks and 25 tasks, the running times of Gurobi are 5689.6 s and 11,364.5 s, respectively while the corresponding values of Q-VNIS are 1.947 s and 7.058 s. The values of gap between Q-VNIS and Gurobi are 27.272% and 19.048%, respectively.

Table 5
Comparison results of and Gurobi.

n	Gurobi		Q-VNIS		Gap
	SI	Time (in seconds)	SI	Time (in seconds)	
6	50	0.023	50	0.298	0.000%
12	93	67.516	93	1.159	0.000%
16	132	5689.600	168	1.947	27.272%
25	21	11,364.500	25	7.058	19.048%
36	–	–	738	10.817	–
47	–	–	617	15.720	–
66	–	–	755	22.839	–

For the cases with 36 tasks, 47 tasks, and 66 tasks, Gurobi cannot obtain the optimal results within a reasonable computing time. The experimental results (738, 617, and 755) and running time (10.817, 15.720, and 22.839 s) of Q-VNIS for the three cases are listed in Table 5. It can be concluded that the solver ensures to obtain the optimal solutions while the computing cost is unacceptable for large-scale cases. The proposed Q-VNIS can balance the solution quality and the computing cost well while there is a gap between its results and the optimal ones for large-scale cases.

4.4. Experimental results and discussions

4.4.1. Comparisons between VNIS and Q-VNIS

In this section, the effectiveness of Q-learning in VNIS is verified. Table 6 reports the results of VNIS and Q-VNIS for 20 cases. It can be seen from Table 6 that the maximum, minimum, and average values of Q-VNIS are less than or equal to the results of VNIS. For cases 1–5, 9 and

Table 6
Comparison of smoothing index of VNIS and Q-VNIS.

Case	VNIS			Q-VNIS		
	Max	Ave	Min	Max	Ave	Min
1	50	50.0	50	50	50.0	50
2	50	50.0	50	50	50.0	50
3	56	56.0	56	56	56.0	56
4	45	45.0	45	45	45.0	45
5	84	84.0	84	84	84.0	84
6	124	94.4	**91**	**91**	**91.0**	**91**
7	99	93.9	**93**	**93**	**93.0**	**93**
8	42	34.6	27	**30**	**26.8**	**25**
9	32	32.0	32	32	32.0	32
10	877	763.8	625	**693**	**644.5**	**617**
11	838	783.6	746	**780**	**745.0**	**738**
12	905	828.5	787	**797**	**769.1**	**755**
13	5158	4842.3	**4600**	**4840**	**4632.7**	**4600**
14	392	239.6	176	**176**	**168.8**	**168**
15	142	131.8	102	**118**	**105.8**	**98**
16	194	159.7	142	**148**	**141.3**	**138**
17	316	217.8	140	**239**	**174.6**	**130**
18	2393	2130.3	1794	**2090**	**1905.5**	**1742**
19	381	345.7	318	**326**	**304.7**	**298**
20	959	867.6	765	**843**	**800.2**	**759**

Table 4
NBVs, NCVs and NCs of the mathematical model regarding DLSP.

Decision variable	NBVs	Decision variable	NCVs	Constraint set	NCs	Constraint set	NCs
w_{ij}	$(n+1)^2$	t_i'	$n+1$	(2)	n^2	(11)	m
x_{ik}	nm	B_i	$n+1$	(3)	$n(n-1)$	(12)	m
y_{ikj}	$(n+1)^2m$	F_i	$n+1$	(4)	$(n+1)^2$	(13)	$(n+1)m$
z_k	m	T_k	m	(5)	$(n+1)^2$	(14)	nm
		IT_k	m	(6)	$n+1$	(15)	nm
				(7)	$n(n+1)$	(16)	m
				(8)	n	(17)	$m-1$
				(9)	$(n+1)^2$	(18)	m
				(10)	$n(n+1)m$		

14 both algorithms have the same results. For cases 6, 7 and 13, both algorithms can obtain the best results (91, 93 and 4600), respectively. However, the Max and Ave values of Q-VNIS are better than those of VNIS. All Max, Ave, and Min values of Q-VNIS are greater than those of VNIS in cases 8, 11, 12, and 14–20. It can be concluded that Q-learning can significantly improve the performance of the VNIS for large-scale cases (cases 6–8, and cases 10–20) and Q-VNIS has better robustness and stability than VNIS.

4.4.2. Comparing to meta-heuristics

In this section, the proposed Q-VNIS is compared to harmony search (HS) algorithm, artificial bee colony (ABC) algorithm, and genetic algorithm (GA). Table 7 reports the results of the four algorithms in 20 cases. As it can be seen from Table 7, the proposed Q-VNIS can find the best Min values for 15 out of 20 cases. For 16 cases, Q-VNIS obtains better Max and Ave values than its peers. For the rest cases, the results of Q-VNIS are equal to the optimal values by other algorithms. In addition, the average values of the average smoothing index values by HS, ABC, GA, and Q-VNIS for all cases are 573.3, 591.6, 661.0, and 546.0, respectively. It can be concluded that the proposed Q-VNIS has the best competitiveness among all compared algorithms. It means that the proposed Q-VNIS can obtain the best results for all cases among all compared algorithms, while other algorithms can only obtain the best results for partial cases.

To evaluate the stability and robustness of the algorithms, the coefficient of variation (CV) of the results is reported in Table 8. The smaller the CV, the better stability and robustness of the algorithm is. It can be observed from Table 8 that the CV values of Q-VNIS algorithm are obviously smaller than or equal to those by the compared algorithms. It means that the stability and robustness of the Q-VNIS algorithm are better than those of its peers.

4.4.3. Convergence analysis

In this section, the convergence curves of five algorithms are reported. Fig. 7 depicts the curves by HS, ABC, GA, VNIS and Q-VNIS algorithms for four cases with different scales. It is clear that all algorithms converge quickly at the beginning and flatten out as the running time increases. The GA gives poorer results with a slower convergence speed. The HS performance is medium and more stable. The ABC performs well in case 9. The VNIS is less effective in the early stages but fluctuates more significantly later. The Q-VNIS has a faster convergence speed and higher-quality solutions than the VNIS, which verifies the effectiveness of Q-learning for selecting premium local search operators. Compared with other algorithms, the Q-VNIS has both fast convergence curves and the best solutions. It can be observed that Q-VNIS has better efficiency and effectiveness than its peers.

Table 8
Comparison of CV values for five algorithms.

Case	HS	ABC	GA	VNIS	Q-VNIS
1	0.00%	0.00%	0.00%	0.00%	0.00%
2	0.00%	0.00%	0.00%	0.00%	0.00%
3	0.00%	0.00%	0.00%	0.00%	0.00%
4	0.00%	0.00%	0.00%	0.00%	0.00%
5	**0.00%**	**0.00%**	4.01%	**0.00%**	**0.00%**
6	**0.00%**	**0.00%**	10.38%	9.34%	**0.00%**
7	**0.00%**	**0.00%**	3.01%	2.28%	**0.00%**
8	7.46%	23.47%	9.32%	10.07%	**5.85%**
9	0.00%	0.00%	0.00%	0.00%	0.00%
10	7.31%	15.50%	7.18%	10.65%	**3.31%**
11	2.38%	2.20%	5.20%	3.93%	**1.38%**
12	3.53%	4.08%	8.47%	3.24%	**1.83%**
13	2.52%	5.36%	2.96%	3.06%	**1.27%**
14	8.40%	17.12%	25.24%	24.90%	**1.42%**
15	8.16%	12.43%	10.04%	9.88%	**5.44%**
16	4.41%	7.94%	11.41%	9.51%	**2.06%**
17	15.46%	20.03%	28.45%	18.47%	**14.95%**
18	5.43%	8.85%	10.60%	8.82%	**5.27%**
19	3.35%	4.06%	18.81%	6.36%	**3.30%**
20	3.64%	4.42%	8.13%	5.89%	**3.05%**

4.4.4. Statistical test

The Friedman test is used to verify the differences in performance between all of the compared algorithms. The significance level is set at 0.05. Table 9 shows the statistical results of the Friedman test. The value of asymptotic significance is much less than 0.05. It means that the performances of the five algorithms are obviously different. Further, the Nemenyi post-hoc test is executed to rank the five algorithms and the ranks are shown in Fig. 8. It can be seen from Fig. 8 that the Q-VNIS is the best one, with the smallest mean rank value (1.68). In addition, Fig. 9 more intuitively reflects the specific rank distribution of the five algorithms for all cases. As can be seen from Fig. 9, Q-VNIS ranked 1 for 13 cases, with the best performance in these cases, and Q-VNIS didn't

Table 7
Comparison of the smoothing index of four algorithms.

Case	HS			ABC			GA			Q-VNIS		
	Max	Ave	Min	Max	Ave	Min	Max	Ave	Min	Max	Ave	Min
1	50	50.0	50	50	50.0	50	50	50.0	50	50	50.0	50
2	50	50.0	50	50	50.0	50	50	50.0	50	50	50.0	50
3	56	56.0	56	56	56.0	56	56	56.0	56	56	56.0	56
4	45	45.0	45	45	45.0	45	45	45.0	45	45	45.0	45
5	**84**	**84.0**	**84**	**84**	**84.0**	**84**	100	84.8	**84**	**84**	**84.0**	**84**
6	**91**	**91.0**	**91**	**91**	**91.0**	**91**	115	100.4	**91**	**91**	**91.0**	**91**
7	**93**	**93.0**	**93**	**93**	**93.0**	**93**	101	94.3	**93**	**93**	**93.0**	**93**
8	39	33.9	28	39	34.2	28	80	43.7	34	**30**	**26.8**	**25**
9	32	32.0	32	32	32.0	32	32	32.0	32	32	32.0	32
10	829	740.3	653	917	775.2	697	1269	934.3	713	**693**	**644.5**	**617**
11	812	758.0	**738**	820	760.5	742	902	830.3	758	**780**	**745.0**	**738**
12	867	811.9	761	891	825.7	777	1105	950.2	799	**797**	**769.1**	**755**
13	5029	4778.2	**4600**	5128	4858.6	**4600**	5878	5317.5	4840	**4840**	**4632.7**	**4600**
14	213	201.9	176	320	214.4	176	452	301.6	213	**176**	**168.8**	**168**
15	130	106.4	98	140	118.9	98	142	131.4	100	**118**	**105.8**	98
16	162	149.2	140	186	157.4	142	228	190.5	146	**148**	**141.3**	**138**
17	240	191.7	**130**	326	229.6	146	489	297.1	159	**239**	**174.6**	**130**
18	2188	2028.0	1840	2477	2157.3	1842	2868	2398.8	1870	**2090**	**1905.5**	**1742**
19	350	336.3	318	387	351.6	338	610	405.1	342	**326**	**304.7**	**298**
20	889	829.1	771	907	847.5	795	1083	908.1	779	**843**	**800.2**	**759**
Average	612.5	573.3	537.7	652.0	591.6	544.1	782.8	661.0	562.7	**579.1**	**546.0**	**528.5**

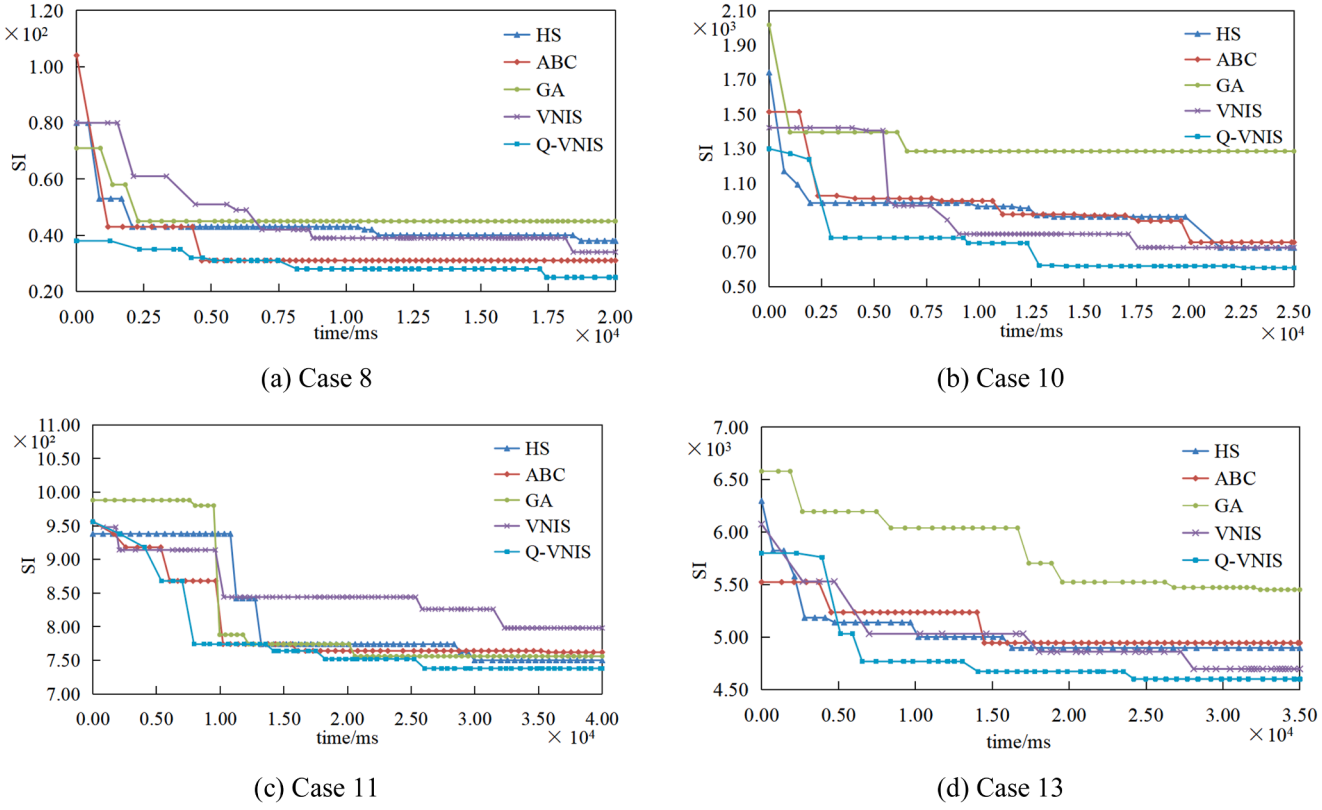


Fig. 7. Convergence curves of all compared algorithms for different-scale cases.

Table 9
Test Statistics of AVE on the cases.

Test Statistics	
N	20
Chi-Square	52.48
df	4
Asymp. Sig.	0.000

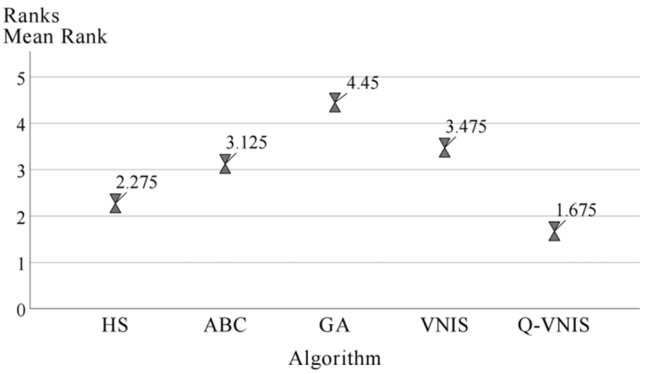


Fig. 8. The Nemenyi post-hoc test of five algorithms.

get the worst results in any case. Other algorithms are not available at rank 1. HS, and ABC have individual cases ranked in second place. No matter in terms of quantity or quality, Q-VNIS is more dominant. It further verifies the strong competitiveness of Q-VNIS.

For comparing the proposed Q-VNIS to its peers one by one, Table 10 shows the results by executing Wilcoxon Signed Ranks Test. It can be seen from Table 10 that the p-values of all algorithm pairs are less than 0.05. Compared with other algorithms, Q-VNIS has a different number of negative ranks. The positive ranks are all 0. It means that Q-VNIS is obviously better than other algorithms.

4.5. Case study: aircraft engine disassembly

Fig. 10 is the disassembly priority diagram of the aircraft engine disassembly case in our project. The aircraft engine consists of 51 components, and the task sequence number, disassembly time, precursor task, successor task, and interference relationship are given in Table 11. For example, $sd_{3,2}$= 18 means that task 2 is disassembled before task 3, and task 3 will interfere with the disassembly time of task 2 by 18 time units. The best result and the corresponding sequence obtained by Q-VNIS with a minimized smoothing index under the workstation threshold are shown in Table 12. The Gantt chart of the best solution for the aircraft engine disassembly case is shown in Fig. 11.

Table 10
Wilcoxon Signed Ranks Test.

Contrast algorithm	p-value	Negative Ranks	Positive Ranks	Ties	Total
Q-VNIS - HS	0.002	12	0	8	20
Q-VNIS - ABC	0.002	12	0	8	20
Q-VNIS - GA	0.001	15	0	5	20
Q-VNIS - VNIS	0.001	14	0	6	20

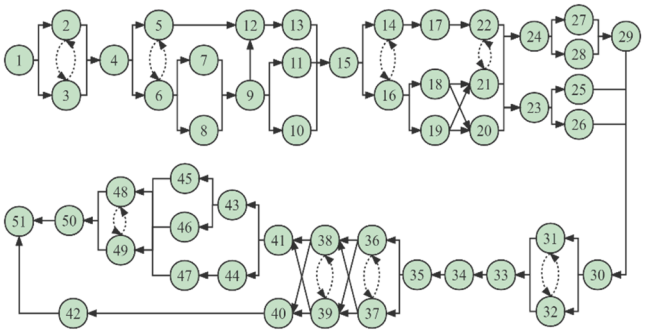


Fig. 10. Precedence graph of the aircraft engine disassembly case.

5. Conclusions and future directions

In this work, a mathematical model for DLSP is established, considering the sequence relationship and the interference time constraints. The goal is to minimize the smoothing index under the workstation number threshold. Based on the features of the concerned problems, a variable neighborhood iterative search algorithm with Q learning is proposed. Experiments are executed for solving 20 instances with different scales. The developed model is verified by the Gurobi solver. The effectiveness of Q-learning is tested. The performance of the proposed algorithm is analyzed by comparing it to HS, ABC, GA. The results and discussions show that the Q-learning based variable neighborhood iterative search algorithm has strong competitiveness.

In the future, the following directions will be pursued: 1) establish a multi-objective model for DLSP with considering more constraints, 2) develop more Q-learning based VNIS and meta-heuristics, and 3) apply

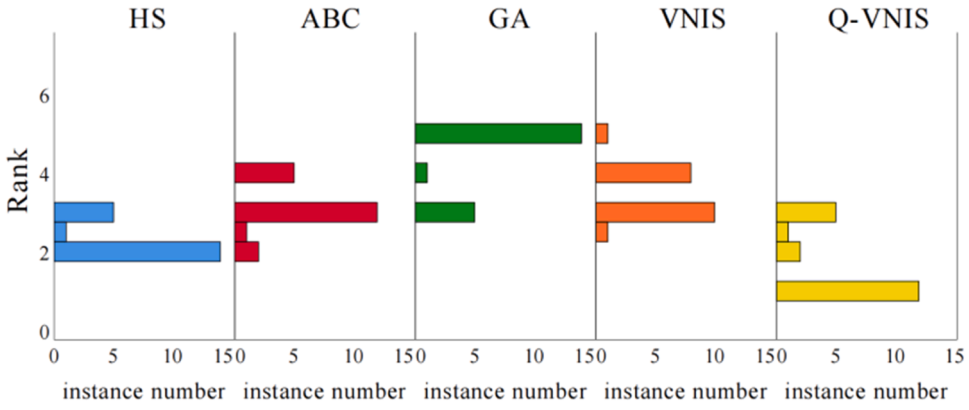


Fig. 9. Friedman's Two-Way Analysis of Variance by Ranks.

Table 11
Disassembly tasks of the aircraft engine.

Task	Time	Precursor	Succeed	Interference	Task	Time	Precursor	Succeed	Interference
1	6	/	2,3	/	27	6	24	29	/
2	6	1	4	$sd_{3,2}$=18	28	6	24	29	/
3	6	1	4	$sd_{2,3}$=12	29	6	27,28	30	/
4	6	2,3	5,6	/	30	6	25,26,29	31,32	/
5	30	4	12	$sd_{6,5}$=12	31	12	30	33	$sd_{32,31}$=12
6	6	4	7,8	$sd_{5,6}$=18	32	9	30	33	$sd_{31,32}$=6
7	6	6	9	/	33	6	31,32	34	/
8	6	6	9	/	34	18	33	35	/
9	6	7,8	10,11,12	/	35	6	34	36,37	/
10	6	9	15	/	36	12	35	38,39	$sd_{37,36}$=6
11	6	10	15	/	37	6	35	38,39	$sd_{36,37}$=12
12	6	5,9	13	/	38	3	36,37	40,41	$sd_{39,38}$=6
13	6	12	15	/	39	6	36,37	40,41	$sd_{38,39}$=3
14	6	15	17	$sd_{16,14}$=12	40	120	38,38	42	/
15	30	10,11,13	14,16	/	41	12	38,39	43,44	/
16	6	15	18,19	$sd_{14,16}$=18	42	6	40	51	/
17	6	14	22	/	43	18	41	45,46	/
18	18	16	20,21	/	44	6	41	47	/
19	12	18	20,21	/	45	1	43	48,49	/
20	60	18	23,24	/	46	6	43	48,59	/
21	60	18,19	23,24	$sd_{22,21}$=18	47	12	44	49	/
22	30	17	23,24	$sd_{21,22}$=30	48	12	45,46,47	50	$sd_{49,48}$=3
23	6	20,21,22	25,26	/	49	12	45,46,47	50	$sd_{48,49}$=6
24	12	20,21,22	27,28	/	50	12	48,49	51	/
25	18	23	30	/	51	3	50	/	/
26	24	23	30	/					

Table 12
The detail data of the best solution by Q-VNIS for the aircraft engine disassembly case.

Algorithm	Smoothing index	Workstation threshold	Sequence
Q-VNIS	4600	4	1,2,3,4,6,8,7,9,10,5,12,13,11,15,16,19,14,17,22,18,21,20,24,27,23,26,28,25,29,30,31,32,33,34,35,37,36,38, 39,41,43,40,44,47,46,45,49,48,42,50,51

Algorithm 1
Q-learning based local search.

```
r←Random
Calculate fitness f;
for action = 1→7 do
                                        Sum += Q-value;
                                        p = action Q-value/ sum;
end for
while p < r;
                                        do action←next action;
end while
Calculate fitness f';
if fitness f' < f
                                        reward = f − f'/10;
                                        if f' = f
                                                                        reward = 1;
                                        else
                                                                        reward = 0;
                                        end if
end if
Update Q-table;
Update selection action probability;
```

the proposed algorithms to other combinatorial optimization and scheduling problems [55–57].

CRediT authorship contribution statement

Yaxian Ren: Investigation, Methodology, Writing – original draft. **Kaizhou Gao:** Formal analysis, Writing – review & editing, Supervision, Funding acquisition. **Yaping Fu:** Writing – review & editing. **Hongyan Sang:** Writing – review & editing. **Dachao Li:** Investigation, Methodology. **Zile Luo:** Investigation, Methodology.

Declaration of Competing Interest

The authors declare no conflict of interest.

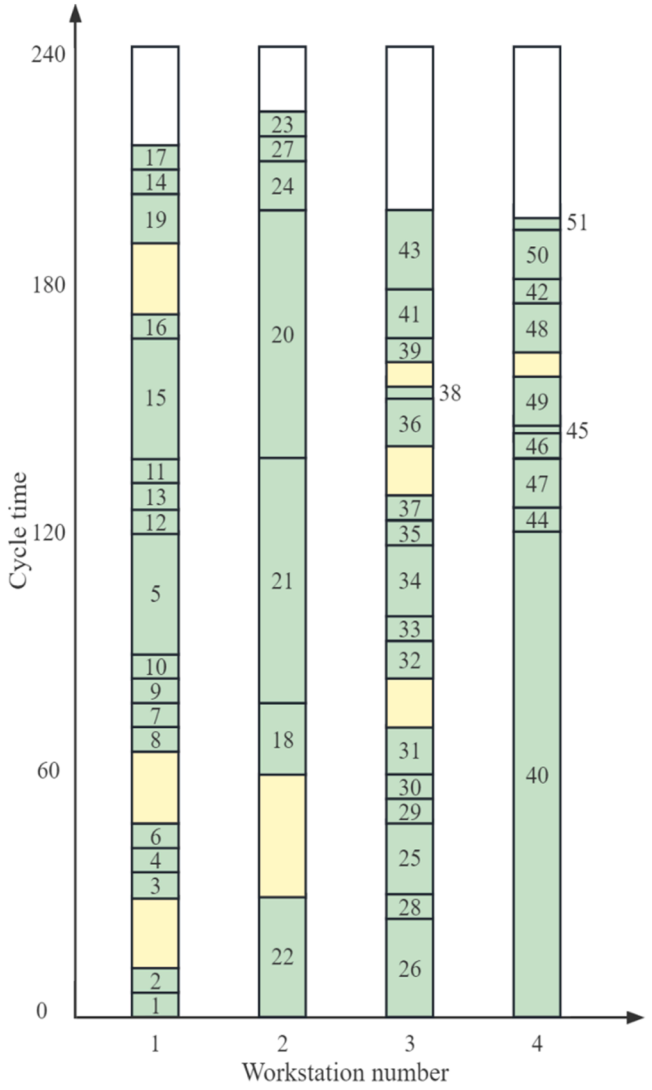


Fig. 11. Gantt charts of the best solution for the aircraft engine disassembly schemes, where green rectangles represent the disassembly time, while the yellow ones represent the interference time.

Data availability

Data will be made available on request.

Acknowledgements

This study is partially supported by the Science and Technology Development Fund (FDCT), Macau SAR, under Grant 0019/2021/A, the National Natural Science Foundation of China under Grant 62173356, Zhuhai Industry-University-Research Project with Hongkong and Macao under Grant ZH22017002210014PWC, the Guangdong Basic and Applied Basic Research Foundation (2023A1515011531), and the Key Technologies for Scheduling and Optimization of Complex Distributed Manufacturing Systems (22JR10KA007).

References

[1] A. Gungor, S.M. Gupta, Issues in environmentally conscious manufacturing and product recovery: a survey, Comput. Ind. Eng. 36 (4) (1999) 811–853.
[2] D.A. Paterson, W.L. Ijomah, J.F Windmill, End-of-life decision tool with emphasis on remanufacturing, J. Clean Prod. 148 (2017) 653–664.
[3] A. Güngör, S.M Gupta, Disassembly line in product recovery, Int. J. Prod. Res. 40 (11) (2002) 2569–2589.
[4] C.B. Kalayci, S.M Gupta, Artificial bee colony algorithm for solving sequence-dependent disassembly line balancing problem, Expert Syst. Appl. 40 (18) (2013) 7231–7241.
[5] E. Özceylan, C.B. Kalayci, A. Güngör, et al., Disassembly line balancing problem: a review of the state of the art and future directions, Int. J. Prod. Res. 57 (15–16) (2019) 4805–4827.
[6] L. Meng, B. Zhang, Y. Ren, et al., Mathematical Formulations for Asynchronous Parallel Disassembly Planning of End-of-Life Products, Mathematics 10 (20) (2022) 3854.
[7] G. Tian, C. Zhang, A.M. Fathollahi-Fard, et al., An enhanced social engineering optimizer for solving an energy-efficient disassembly line balancing problem based on bucket brigades and cloud theory, IEEE Trans. Ind. Inf. (2022).
[8] K. Wang, L. Gao, X. Li, et al., Energy-efficient robotic parallel disassembly sequence planning for end-of-life products, IEEE Trans. Autom. Sci. Eng. 19 (2) (2021) 1277–1285.
[9] X. Guo, M. Zhou, S. Liu, et al., Lexicographic multiobjective scatter search for the optimization of sequence-dependent selective disassembly subject to multiresource constraints, IEEE Trans. Cybern. 50 (7) (2019) 3307–3317.
[10] M. Liu, X. Liu, F. Chu, et al., An exact method for disassembly line balancing problem with limited distributional information, Int. J. Prod. Res. 59 (3) (2021) 665–682.
[11] S. Mete, Z.A. Çil, E. Celik, et al., Supply-driven rebalancing of disassembly lines: a novel mathematical model approach, J. Clean Prod. 213 (2019) 1157–1164.
[12] F. Pistolesi, B. Lazzerini, M. Dalle Mura, et al., EMOGA: a hybrid genetic algorithm with extremal optimization core for multiobjective disassembly line balancing, IEEE Trans. Ind. Inf. 14 (3) (2017) 1089–1098.
[13] K. Wang, X. Li, L. Gao, Modeling and optimization of multi-objective partial disassembly line balancing problem considering hazard and profit, J. Clean Prod. 211 (2019) 115–133.
[14] K. Wang, X. Li, L. Gao, et al., A discrete artificial bee colony algorithm for multiobjective disassembly line balancing of end-of-life products, IEEE Trans. Cybern. (2021).
[15] J. Liang, S. Guo, B. Du, et al., Minimizing energy consumption in multi-objective two-sided disassembly line balancing problem with complex execution constraints using dual-individual simulated annealing algorithm, J. Clean Prod. 284 (2021), 125418.
[16] Y. Zhang, Z. Zhang, C. Guan, et al., Improved whale optimisation algorithm for two-sided disassembly line balancing problems considering part characteristic indexes, Int. J. Prod. Res. 60 (8) (2022) 2553–2571.
[17] F.T. Altekin, C. Akkan, Task-failure-driven rebalancing of disassembly lines, Int. J. Prod. Res. 50 (18) (2012) 4955–4976.
[18] F.T. Altekin, L. Kandiller, N.E Ozdemirel, Profit-oriented disassembly-line balancing, Int. J. Prod. Res. 46 (10) (2008) 2675–2693.
[19] M.L. Bentaha, O. Battaïa, A. Dolgui, An exact solution approach for disassembly line balancing problem under uncertainty of the task processing times, Int. J. Prod. Res. 53 (6) (2015) 1807–1818.
[20] A. Koc, I. Sabuncuoglu, Erel E. Two exact formulations for disassembly line balancing problems with task precedence diagram construction using an AND/OR graph, IIE Trans. 41 (10) (2009) 866–881.
[21] S.M. Mcgovern, S.M Gupta, Uninformed and probabilistic distributed agent combinatorial searches for the unary NP-complete disassembly line balancing problem, Environmentally Conscious Manuf. V (2005) 81–92.
[22] Y. Fu, Y. Hou, Z. Wang, et al., Distributed scheduling problems in intelligent manufacturing systems, Tsinghua Sci. Technol. 26 (5) (2021) 625–645.
[23] K. Gao, Z. Cao, L. Zhang, et al., A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems, IEEE/CAA J. Automatica Sinica 6 (4) (2019) 904–916.
[24] K. Gao, Y. Huang, A. Sadollah, et al., A review of energy-efficient scheduling in intelligent production systems, Complex Intell. Syst. 6 (2) (2020) 237–249.
[25] K. Gao, F. Yang, M. Zhou, et al., Flexible job-shop rescheduling for new job insertion by using discrete Jaya algorithm, IEEE Trans. Cybern. 49 (5) (2018) 1944–1955.
[26] L. Meng, K. Gao, Y. Ren, et al., Novel MILP and CP models for distributed hybrid flowshop scheduling problem with sequence-dependent setup times, Swarm Evol. Comput. 71 (2022), 101058.
[27] L. Meng, C. Zhang, Y. Ren, et al., Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem, Comput. Ind. Eng. 142 (2020), 106347.
[28] Y. Pan, K. Gao, Z. Li, et al., Solving Biobjective Distributed Flow-Shop Scheduling Problems With Lot-Streaming Using an Improved Jaya Algorithm, IEEE Trans. Cybern. (2022).
[29] P.W. Shaikh, M. El-Abd, M. Khanafer, et al., A review on swarm intelligence and evolutionary algorithms for solving the traffic signal control problem, IEEE Trans. Intell. Transp. Syst. 23 (1) (2020) 48–63.
[30] Y. An, X. Chen, K. Gao, et al., A hybrid multi-objective evolutionary algorithm for solving an adaptive flexible job-shop rescheduling problem with real-time order acceptance and condition-based preventive maintenance, Expert Syst. Appl. 212 (2023), 118711.
[31] H.-E. Tseng, C.-C. Chang, S.-C. Lee, et al., A block-based genetic algorithm for disassembly sequence planning, Expert Syst. Appl. 96 (2018) 492–505.
[32] Y. Ren, H. Jin, F. Zhao, et al., A multiobjective disassembly planning for value recovery and energy conservation from end-of-life products, IEEE Trans. Autom. Sci. Eng. 18 (2) (2020) 791–803.

[33] Y. Ren, C. Zhang, F. Zhao, et al., An asynchronous parallel disassembly planning based on genetic algorithm, Eur. J. Oper. Res. 269 (2) (2018) 647–660.
[34] Y. Tian, X. Zhang, Z. Liu, et al., Product cooperative disassembly sequence and task planning based on genetic algorithm, Int. J. Adv. Manuf. Technol. 105 (5) (2019) 2103–2120.
[35] L. Zhu, Z. Zhang, Y. Wang, A Pareto firefly algorithm for multi-objective disassembly line balancing problems with hazard evaluation, Int. J. Prod. Res. 56 (24) (2018) 7354–7374.
[36] Y. Ren, C. Zhang, F. Zhao, et al., An MCDM-based multiobjective general variable neighborhood search approach for disassembly line balancing problem, IEEE Trans. Syst., Man, Cybern.: Systems 50 (10) (2018) 3770–3783.
[37] X. Xia, H. Zhu, Z. Zhang, et al., 3D-based multi-objective cooperative disassembly sequence planning method for remanufacturing, The International Journal of Advanced Manufacturing Technology 106 (9) (2020) 4611–4622.
[38] L. Zhu, Z. Zhang, Y. Wang, et al., On the end-of-life state oriented multi-objective disassembly line balancing problem, J Intell Manuf 31 (6) (2020) 1403–1428.
[39] H. Guo, L. Zhang, Y. Ren, et al., Optimizing a stochastic disassembly line balancing problem with task failure via a hybrid variable neighborhood descent-artificial bee colony algorithm, Int. J. Prod. Res. (2022) 1–15.
[40] K.-Z. Gao, Z. He, Y. Huang, et al., A survey on meta-heuristics for solving disassembly line balancing, planning and scheduling problems in remanufacturing, Swarm Evol. Comput. 57 (2020), 100719.
[41] S. Chen, Q.-K. Pan, L. Gao, et al., A population-based iterated greedy algorithm to minimize total flowtime for the distributed blocking flowshop scheduling problem, Eng. Appl. Artif. Intell. 104 (2021), 104375.
[42] Y.-Y. Huang, Q.-K. Pan, L. Gao, et al., A two-phase evolutionary algorithm for multi-objective distributed assembly permutation flowshop scheduling problem, Swarm Evol. Comput. 74 (2022), 101128.
[43] Z.-Y. Wang, Q.-K. Pan, L. Gao, et al., An effective two-stage iterated greedy algorithm to minimize total tardiness for the distributed flowshop group scheduling problem, Swarm Evol. Comput. 74 (2022), 101143.
[44] L. Wang, Z. Pan, J. Wang, A review of reinforcement learning based intelligent optimization for manufacturing scheduling, Complex Syst. Model. Simul. 1 (4) (2021) 257–270.
[45] F. Zhao, S. Di, L. Wang, A hyperheuristic with q-learning for the multiobjective energy-efficient distributed blocking flow shop scheduling problem, IEEE Trans. Cybern. (2022).
[46] F. Zhao, Z. Wang, L. Wang, A Reinforcement Learning Driven Artificial Bee Colony Algorithm for Distributed Heterogeneous No-Wait Flowshop Scheduling Problem With Sequence-Dependent Setup Times, IEEE Trans. Autom. Sci. Eng. (2022).
[47] F. Zhao, X. Hu, L. Wang, et al., A reinforcement learning brain storm optimization algorithm (BSO) with learning mechanism, Knowl. Based Syst. 235 (2022), 107645.
[48] M. Karimi-Mamaghan, M. Mohammadi, B. Pasdeloup, et al., Learning to select operators in meta-heuristics: an integration of Q-learning into the iterated greedy algorithm for the permutation flowshop scheduling problem, Eur. J. Oper. Res. 304 (3) (2023) 1296–1330.
[49] H. Li, K. Gao, P.-Y. Duan, et al., An Improved Artificial Bee Colony Algorithm With Q-Learning for Solving Permutation Flow-Shop Scheduling Problems, IEEE Trans. Syst., Man, Cybern.: Systems (2022).
[50] Watkins C J C H. Learning from delayed rewards, 1989.
[51] J. Liu, Research On Sequence-Dependent Disassembly Line Balancing Problem, University of Electronic Science and Technology of China, 2017.
[52] C.B. Kalayci, A. Hancilar, A. Gungor, et al., Multi-objective fuzzy disassembly line balancing using a hybrid discrete artificial bee colony algorithm, J. Manuf. Syst. 37 (2015) 672–682.
[53] H. Wang, Civil Aviation Repair Personnel Scheduling Research and Application Based On Task Matching, Chongqing University, 2015.
[54] D. Montgomery, Design and Analysis of Experiments, John Wiley, Hoboken: NJ, 2005.
[55] Z. Wang, Y.-S. Ong, H. Ishibuchi, On scalable multiobjective test problems with hardly dominated boundaries, IEEE Trans. Evol. Comput. 23 (2) (2018) 217–231.
[56] Z. Wang, Q. Zhang, Y.-S. Ong, et al., Choose appropriate subproblems for collaborative modeling in expensive multiobjective optimization, IEEE Trans. Cybern. 53 (1) (2021) 483–496.
[57] Z. Wang, H.-L. Zhen, J. Deng, et al., Multiobjective optimization-aided decision-making system for large-scale manufacturing planning, IEEE Trans. Cybern. 52 (8) (2021) 8326–8339.