COMPLEX SYSTEM MODELING AND SIMULATION
ISSN 2096-9929 05/06 pp 174−185
Volume 2, Number 2, June 2022
DOI: 10.23919/CSMS.2022.0007

# Optimal Design of Flexible Job Shop Scheduling Under Resource Preemption Based on Deep Reinforcement Learning

Zhen Chen, Lin Zhang*, Xiaohan Wang, and Pengfei Gu

**Abstract:** With the popularization of multi-variety and small-batch production patterns, the flexible job shop scheduling problem (FJSSP) has been widely studied. The sharing of processing resources by multiple machines frequently occurs due to space constraints in a flexible shop, which results in resource preemption for processing workpieces. Resource preemption complicates the constraints of scheduling problems that are otherwise difficult to solve. In this paper, the flexible job shop scheduling problem under the process resource preemption scenario is modeled, and a two-layer rule scheduling algorithm based on deep reinforcement learning is proposed to achieve the goal of minimum scheduling time. The simulation experiments compare our scheduling algorithm with two traditional metaheuristic optimization algorithms among different processing resource distribution scenarios in static scheduling environment. The results suggest that the two-layer rule scheduling algorithm based on deep reinforcement learning is more effective than the meta-heuristic algorithm in the application of processing resource preemption scenarios. Ablation experiments, generalization, and dynamic experiments are performed to demonstrate the excellent performance of our method for FJSSP under resource preemption.

**Key words:** flexible job shop scheduling; resource preemption; deep reinforcement learning; two-level scheduling

## 1 Introduction

In recent years, with the development of industrial production and market demand, flexible job shop system has been widely studied as an important form of intelligent workshop. Flexible job shop scheduling problem turns out to be an NP-hard problem. As an important processing tool in production lines, industrial robots have been widely used in smart workshops in recent years. In many scenarios, robots use resources to process work. However, due to the short supply of processing resources such as space and tools, multiple processing robots often need to share a resource pool, that is, processing under resource constraints. Scheduling jobs in the above environment constitute the flexible job shop scheduling problem under resource preemption (FJSSP-RP). In this paper, we abstract and model FJSSP-RP. In FJSSP-RP, different processing tasks need to be completed by matching robots, and the premise that robots can perform processing a task is to obtain the resource required for processing specific task. In addition, in the actual job shop, the modeling process of the scheduling problem is similar, only the types of processes or the distribution of resources are different, so the generalization of the solution algorithm is also one of the factors to be considered.

The essence of FJSSP-RP still belongs to the category of FJSSP. The improved heuristic algorithm is a mainstream method to solve FJSSP. Zhang[1] proposed a multi-objective optimization algorithm fusion non-dominated sorting genetic algorithm (FNSGA) based on the improved non-dominated sorting genetic algorithm-II (NSGA-II) algorithm. The

• Zhen Chen, Lin Zhang, Xiaohan Wang, and Pengfei Gu are with the School of Automation Science and Eletrical Engineering, Beihang University, Beijing 100191, China. E-mail: czhen@buaa.edu.cn; johnlin9999@163.com; xiaohan wang@buaa.edu.cn; by2003151@buaa.edu.cn.
* To whom correspondence should be addressed.
Manuscript received: 2022-03-04; revised: 2022-03-30; accepted: 2022-05-06

algorithm was used to solve the problems of slow search speed and low efficiency of scheduling scheme in the traditional multi-objective optimization algorithm. In order to optimize the muti-goal of job shop completion time, total machine load, and job shop energy consumption, Zhu[2] proposed an improved strong propagation NSGA-II algorithm in which the chromosomes were divided into strong propagation subgroups and ordinary subgroups according to the different reproductive ability. The two subgroups used different genetic operations according to their own characteristics, so as to promote the efficiency of the algorithm. For minimizing the shop completion time, Ref. [3] introduced a new conversion mechanism based on the differential evolution algorithm, which makes the differential evolution algorithm suitable for solving discrete problems. In Ref. [4], a new hybrid improved genetic algorithm was proposed to solve and optimize the FJSSP. The new genetic algorithm improved the diversity of the population by strengthening the initial population quality, and promoted the search ability by improving the mutation mechanism. According to the standard cuckoo algorithm, a double-layer coding discrete cuckoo algorithm was proposed in Ref. [5]. Teekeng et al.[6] adopted the improved particle swarm optimization algorithm to avoid premature convergence to the local optimal solution by expanding the solution space of FJSSP. And Teekeng et al.[6] used 20 benchmark examples to benchmark enhanced particle swarm optimization (EPSO) to prove the effectiveness of the algorithm. Boyer et al.[7] proposed a generalized flexible job shop scheduling model based on a real seamless rolling ring manufacturing environment and considering various strong constraints such as time delay and holding time. On this basis, a greedy random adaptive search algorithm was proposed to minimize the maximum completion time. Liu et al.[8] proposed a mixed-variable differential evolution approach to solve coordinated charging scheduling of electric vehicles. Zhou et al.[9] proposed a self-adaptive differential evolution algorithm for scheduling a single batch-processing machine with arbitrary job sizes and release times. Zhao et al.[10–12] proposed three new improved algorithms for different flowshop scheduling scenarios. However, in the FJSSP-RP problem, the difficulty and complexity of encoding and decoding using traditional heuristic algorithms are increased due to resource preemption, and the convergence speed is also affected. At the same time, the dynamic problem of job shop cannot be solved by traditional heuristic algorithm. In addition, heuristic algorithms can only be specifically designed for specific problems, and have poor generalization.

In view of the shortcomings of the above solutions, reinforcement learning algorithm has also been applied to solve FJSSP[13] in recent years. In static scheduling environments, Gabel and Riedmilier[14] transformed the classical job shop scheduling into a sequential decision problem, and introduced the neural network to approximate the value function. Martínez et al.[15] combined the heuristic algorithm with reinforcement learning (RL) to find suitable operators and optimize based on that. Chen et al.[16] proposed a self-learning genetic algorithm based on reinforcement learning. The algorithm uses genetic algorithm as the basic optimization method, and its key parameters are intelligently adjusted by reinforcement learning, in order to obtain faster convergence speed and better convergence results. In dynamic scheduling environments, value-based algorithms are widely adopted. Reference [17] modeled the dynamic scheduling problem with new job insertion in FJSSP, took minimizing the completion time as the optimization objective, and solved the problem by using the deep-Q-network (DQN) algorithm. Csáji et al.[10] proposed a triple-level learning mechanism to achieve adaptive behavior and search space reduction. Wang et al.[18] applied multi-agent reinforcement learning to deal with workshop resource constraints scheduling problem.

Based on the above existing practice in FJSSP, we can conclude that although the heuristic algorithm has good search ability, the generalization of the algorithm and the ability to deal with dynamic events in the production process are insufficient. Therefore, we focus our algorithm on RL-based algorithm. In view of the shortage of resources in the FJSSP-RP problem, it is very important to reasonably select the artifacts participating in the current scheduling action. We apply this idea to the design of the RL algorithm.

In this paper, the resource preemptive flexible job shop scheduling problem is analyzed and optimized. On this basis, a two-layer rule scheduling optimization algorithm based on proximal policy optimization (PPO) algorithm is proposed. The experiments compare our scheduling algorithm with two traditional meta-heuristic optimization algorithms among different processing resource distribution scenarios. The results suggest that the two-layer rule scheduling algorithm based on deep reinforcement learning is more effective

than meta-heuristic algorithm in the application of processing resource preemption scenarios. Furthermore, we test the algorithm under dynamic conditions such as random device failures, and use the control variable method to test the components in the algorithm. Finally, we show the generalization of the trained algorithm to other resource distribution scenarios. The results show that the proposed algorithm has good generalization and dynamic coping ability for FJSSP-RP.

The content of this article is organized as follows: Section 2 introduces the background of the problem and the algorithm. Section 3 describes and formalizes the problem, and gives the detailed design of the algorithm. In Section 4, we compare the performance of the algorithm in this paper with the two traditional algorithms under different processing resource distributions. The generalization, dynamic, and ablation experiments are carried out for the performance of the algorithm proposed in this paper. Section 5 presents the conclusion of this paper.

## 2 Background

### 2.1 Resource preemption in flexible job shop scheduling

As an NP-hard problem[19], flexible job shop scheduling problem is one of the most key problems in manufacturing and process planning[20]. In this problem, a group of workpieces are recorded as $\{J_1, J_2, J_3, ..., J_n\}$ and need to be processed in a group of machines recorded as $\{M_1, M_2, M_3, ..., M_m\}$. The processing of each workpiece includes multiple processes $\{O_{i1}, O_{i2}, O_{i3}, ..., O_{in}\}$, where $O_{ij}$ represents the $j$-th process of the $i$-th workpiece $J_i$. The processes meet certain sequence constraints. At a certain time, a machine can only process one process of one workpiece. The flexibility of flexible job shop scheduling problem is reflected in two aspects. On one hand, the same machine can process different processes of different workpieces at different time. On the other hand, for a workpiece, the processing machine is not fixed, and a production line can be composed of multiple different machines to minimize the processing time of the workpiece.

Flexible job shop can be divided into fully flexible job shop and partial flexible job shop according to Ref. [2]. In a fully flexible job shop, each process can be completed on any machine, that is, the list of processes that can be processed by each machine contains all processes. In partial flexible job shops, processes can only be completed in specific machines. For example, for all $n$ processes, the set of processes that can be processed by machine $M_j$ is $\{O_{j1}, O_{j2}, O_{j3}, ..., O_{jm}\}$, where $m \leqslant n$. In actual production and life, partial flexible job shops are more common[21].

In the general flexible job shop scheduling problem, the process resources in the resource list of each machine are independent of each other, but in some specific scheduling scenarios, there are some restrictions on the resources between each machine. For example, aircraft carrier deck supply scheduling, automobile maintenance scheduling, etc. In other words, the concept of "machine" in the above problem can be understood as a spatial "occupant", each "occupant" can only process one workpiece at a certain time, and some occupants may share resources for processing operations. For example, machine $M_i$ and machine $M_j$ share a set of resource lists $\{O_{ij1}, O_{ij2}, O_{ij3}, ..., O_{ijm}\}$, where $O_{ijr}$ represents the $r$-th shared process resource of $M_i$ and machine $M_j$. When $O_{ijk}$ in the resource list is used by machine $M_i$, machine $M_j$ can only use the remaining resources in the resource list during the same time period. Scheduling the processing of the workpiece in such a scenario constitutes a flexible job shop scheduling problem under resource preemption.

### 2.2 Markov decision process and reinforcement learning

Markov decision process (MDP) is a mathematical modeling of sequential decision events[22]. MDP models decision-making in solving the problem whose results are partly random and partly controlled by decision-makers. And MDP is of great significance to the study of the optimization of dynamic programming. A Markov decision process is represented by a quad $\langle S, A, T, R\rangle$ formed by state set $S$, action set $A$, state transition function $T$, and reward function $R$. Given any state $s \in S$, selecting an action $a \in A$ will cause the environment to enter a new state $s' \in S$, and return the reward $R_a(s, s')$. The new state is determined by the transition probability matrix $T(s', a, s) \in [0, 1]$. Random strategy $\pi : s \to a$ refers to the probability distribution of action in a given state[23].

Reinforcement learning, as a field of machine learning, emphasizes how to interact with the environment and take appropriate actions to maximize the goal. Markov decision process is the most common form of defining reinforcement learning problems.

Reinforcement learning algorithm mainly uses temporal-difference learning method, Monte Carlo method, and dynamic programming method to update its value function or strategy function[24].

Deep reinforcement learning is the product of the combination of deep learning technology and reinforcement learning. The powerful function approximation ability of deep learning enables the agent to learn the value function or strategy function more efficiently, and provides the possibility for end-to-end learning with better performance[25]. Specifically, two mainstream algorithms based on value and strategy are formed. Value-based algorithms generally only work in discrete action spaces, deterministic strategy based strategies generally work in continuous action spaces, and random strategy based strategies support discrete and continuous action spaces[26].

### 2.3 Proximal policy optimization algorithm

Proximal policy optimization (PPO) algorithm is a model free deep reinforcement learning (DRL) method and a classical algorithm in actor-critic architecture. It can be applied to discrete control tasks and continuous control tasks. In aspects of exploration methods and sample management, PPO adopts an off-policy architecture, which means the sampling strategy is different from the strategy to be optimized, so as to improve the sampling efficiency. In order to improve sample usage efficiency, PPO divides a batch of samples into multiple mini-batches and reuses them for many times. In gradient calculation, PPO inherits the idea of confidence region of trust region policy optimization (TRPO) algorithm and increases the stability of training by limiting the range of parameter update.

PPO applies the importance sampling method to change the training algorithm of on-policy into off-policy, which not only improves the low sample utilization of on policy algorithm, but also has the characteristics of high stability of on policy. $\frac{p_\theta(\tau)}{p_{\theta'}(\tau)}$ represents the importance weight in displayed equation.

$$\nabla \bar{R}_\theta = E_{\tau \sim p_{\theta'}(\theta)}\left[\frac{p_\theta(\tau)}{p_{\theta'}(\tau)} R(\tau) \nabla \log p_\theta(\tau)\right] \tag{1}$$

where $E$ is the expectation of the reward.

In addition to using KL divergence to punish the probability distribution of constrained actions, another implementation of PPO algorithm is to use $\frac{\pi_{\theta'}(a \mid s)}{\pi_\theta(a \mid s)}$. In this approach, the degree constraint of deviation from 1 replaces the Kullback–Leibler (KL) divergence constraint, and the excess part shall be cut directly[27]. Only the data in the confidence interval can return the gradient to the policy network. For the excess part, the gradient will not be generated due to the truncation operation.

The algorithm flow of PPO is shown in Algorithm 1[28].

## 3 Problem Description and Algorithm Design

### 3.1 Problem description

FJSSP is an NP-hard problem. The occurrence of resource preemption in FJSSP-RP makes some resources dynamically available at some moments, which further increases the state space of FJSSP-RP. The flexible job shop scheduling problem under resource preemption is described as follows: There are $N$ workpieces in the job shop which need to be processed, denoted as $\{J_1, J_2, J_3, ..., J_n\}$. There are $m$ types of all processes for all workpieces, and the complete set is $O = \{O_1, O_2, O_3, ..., O_m\}$. For a certain workpiece $J_k$, the list of processes to be completed is denoted as $\{oj_{1^{(k)}} \to oj_{2^{(k)}} \to \cdots \to oj_{h^{(k)}} \mid oj_i \in O\}$. For the process resource list $OM_p = \{om_1, om_2, \ldots, om_r \mid om_i \in O\} \subset O$, it is shared by the machine or the occupant $M_{i1}$, $M_{i2}$, ..., $M_{in}$, where $n \geqslant 1$. For each process of the workpiece, a corresponding machine is required for processing, and the machine must contain the resources corresponding to the current processing procedure of the workpiece. Each process has a fixed processing time, the processing time of the process $O_p$ is donated as $t_{\text{span}}$. It takes time for the workpiece to transfer between different machines. The transfer time is related to the speed and the distance between the machines. The position of the occupant $M_{ij}$ is donated as $W(M_{ij})$, then the transfer time from $M_{ij}$ to $M'_{ij}$ is $t_{\text{trans}} = (W(M_{ij}) -$

**Algorithm 1 PPO algorithm**

For itration=1, 2, ..., do
  For actor=1, 2, ..., $N$ do
    Run policy $\pi_{\theta_{old}}$ in environment for $T$ timestep
    Compute advantage etimates $\widehat{A_1}, \widehat{A_2}, \ldots, \widehat{A_T}$
  End for
  Optimize surrogate $L$ wrt $\theta$ with $L$ epochs and minibatch size $M \leqslant NT$
  $\theta \to \theta_{old}$
End for

$W(M'_{ij}))/speed$.

Because of the scheduling arrangement, the workpiece may need to wait in the original place before proceeding to the next processing, which is recorded as $t_{\text{delay}}$. Assuming that the workpiece $J_k$ completes the $(d^{(k)})$-th process $oj_{d^{(k)}}$ of its process list $OJ_k$, the transfer time required before processing is $t^{d^{(k)}}_{\text{trans}}$, the waiting time required for scheduling is $t^{d^{(k)}}_{\text{delay}}$, and the processing time to complete is $t^{d^{(k)}}_{\text{span}}$. Under the constraints of production relations, the scheduling objective is to find a scheme with the shortest time to complete all tasks, which is expressed below:

$$\text{minimize}\left(\max_{1\leqslant k\leqslant n}\left(\sum_{d^{(k)}=1}^{d^{(k)}=h^{(k)}}\left(t^{d^{(k)}}_{\text{trans}}+t^{d^{(k)}}_{\text{delay}}+t^{d^{(k)}}_{\text{span}}\right)\right)\right).$$

## 3.2 Design of a PPO-based two-layer rule scheduling algorithm

The block diagram of the PPO-based two-layer rule scheduling optimization algorithm is shown in Fig. 1. In each iteration, according to the actions output by the PPO, the algorithm selects the workpieces that participate in current update, and selects the allocation rules for workpieces. And then the job shop environment is changed according to the above two rules. Finally, the environment state and rewards are provided to the agent, and the next iteration will continue.

One of the main features of this algorithm is that in order to ensure that the output action is as simple as possible to apply to the agent, and to maximize the role of human experience, the upper and lower levels of scheduling action mechanisms are used here to solve the scheduling problem under resource constraints. The specific method is: the upper-level action is that the PPO selects the workpieces to participate in the current round of update according to the current environment information, and the action used in the current round of the update is selected in the lower-level simple action set; the lower-level action is some set of matching rules that have been restricted. The details are shown in Table 1.

The detailed design process of the algorithm is given below. Prior to this, the parameter information used is given in Table 2.

### 3.2.1 State

As the input of the PPO algorithm, the state information needs to reflect the information of the current environment as comprehensively as possible, so that the agent can make better decisions. The state information should include all machine's status information, workpiece information, and process information. The status sequence information is defined as follows:

$$\text{State} = [M_J, M_S, M_L, J_{\text{next}}, J_{\text{now}}, J_{\text{left}}, J_{\text{done}}] \quad (2)$$

### 3.2.2 Action

When the number of processed workpieces is $N$, the output action of the PPO network is $N+1$ dimension. The first $N$ dimensions specify the situation where $N$ workpieces participate in this round of iteration, and the output set of each dimension is $\{0, 1\}$. "1" means participating, and "0" means not. The $(N+1)$-th

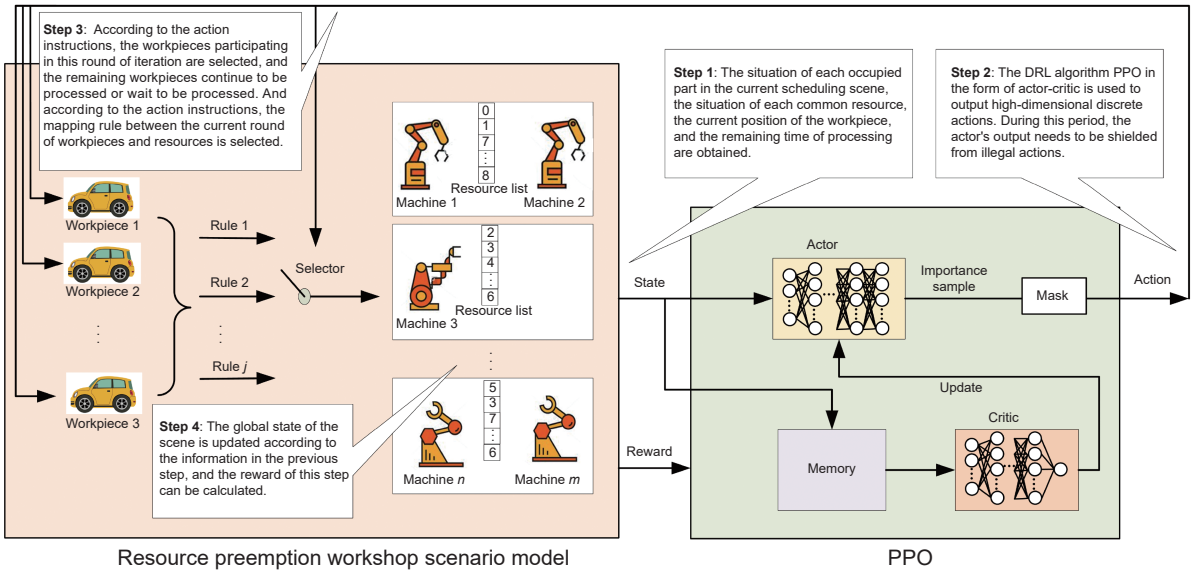


**Fig. 1 PPO-based two-layer rule scheduling algorithm flowchart.**

**Table 1 Lower level action set.**

| No. | Rule name |
|---|---|
| 1 | The shortest processing time in sequence in the next iteration |
| 2 | The longest processing time in sequence in the next iteration |
| 3 | The shortest processing and transfer time in sequence in the next iteration |
| 4 | The longest processing and transfer time in sequence in the next iteration |
| 5 | The shortest transfer time in sequence in the next iteration |
| 6 | The longest transfer time in sequence in the next iteration |

**Table 2 State parameter list.**

| Symbol | Explanation |
|---|---|
| $M_J$ | Set of workpieces currently being processed |
| $M_{\text{sum}}$ | Number of machines in processing |
| $M_S$ | List of current resources of all machines |
| $M_L$ | Remaining time to complete the current process for all machines |
| $OJ_i^{\text{next}}$ | Workpiece $J_i$ next process label |
| $J_{\text{now}}$ | Processing state collection of each workpiece |
| $J_{\text{left}}$ | Number of remaining processes for each workpiece |
| $J_{\text{done}}$ | Number of completed processes for each workpiece |
| $J_{\text{choosed}}$ | Set of workpieces selected to participate in the current iteration |
| $S_{\text{offer}}$ | Available processing resources and machine information |
| $R_{\text{choosed}}$ | The lower rule selected in the current iteration |
| $M$ | Complete set of machines |
| $T_i$ | Remaining time of the current processing procedure of the $i$-th machine |

dimension specifies the lower-level action taken in this iteration. For the first $N$-dimensional output, because of the environment's restrictions on workpieces, it is necessary to shield the illegal action of the $N$-dimensional action before the PPO output and the actual output results are fed back to the network training through the loss function.

### 3.2.3 Reward

The design part of the reward is very important and is related to the convergence speed and convergence effect of the model. The reward design mainly considers two aspects. On one hand, the invalid scheduling action is punished. The invalid scheduling action here is defined as: one workpiece is selected to participate in this iteration, but in fact the environment cannot provide the resources needed for its next step, namely:

$$\exists J_i \in J_{\text{choosed}}, OJ_i^{\text{next}} \notin S_{\text{offer}} \tag{3}$$

On the other hand, the effective use of resources needs to be rewarded. The effective use of resources is measured by the following method: the size of the number of machines in processing $M_{\text{sum}}$. Based on the above considerations, the reward is expressed below:

$$\text{reward} = \begin{cases} -10, \text{ if Eq. (4) is true;} \\ 2, \text{ else if } 3 \leqslant M_{\text{sum}}; \\ -1, \text{ else} \end{cases} \tag{4}$$

### 3.2.4 Environment iteration steps

The environment iteration steps are given below:

**Step 1**: Determine the workpieces set $J_{\text{choosed}}$ participating in this iteration and the rule $R_{\text{choosed}}$ adopted in this iteration according to the PPO output. Calculate the matching relationship between $J_{\text{choosed}}$ and $M_{\text{choosed}}$ according to $R_{\text{choosed}}$.

**Step 2**: Update the remaining processing time of the machine. For machine in complete set of machines $M$, the increased time in this round of iteration is:

$$\Delta t_i = \begin{cases} 0, \text{ if } \quad M_i \notin M_{\text{choosed}}; \\ t_{\text{trans}} + t_{\text{span}}, \text{ if } \quad M_i \in M_{\text{choosed}} \\ \qquad\qquad \text{for } i \in [0, \text{len}(M)] \end{cases} \tag{5}$$

Update the remaining processing time to

$$T_i \leftarrow T_i + \Delta t_i \tag{6}$$

Determine the time step of this iteration as $\min(T_i)$. Subtract the duration of this round to get the processing duration after this iteration:

$$T_i \leftarrow T_i - \min(T_i), \text{ if } T_i < 0 \text{ then let } T_i = 0 \tag{7}$$

**Step 3**: If $T_i = 0$, then update the status of the

relevant workpiece to be idle and machine to be available.

#### 3.2.5 Algorithm iteration steps

Finally, the process of each iteration of the algorithm is given as follows:

**Step 1**: The situation of each occupied part in the current scheduling scene, the situation of each common resource, the current position of the workpiece, and the remaining time of processing are obtained.

**Step 2**: The DRL algorithm PPO in the form of actor-critic is used to output high-dimensional discrete actions. During this period, the actor's output needs to be shielded from illegal actions.

**Step 3**: According to the action instructions, the workpieces participating in this round of iteration are selected, and the remaining workpieces continue to be processed or wait to be processed. And according to the action instructions, the mapping rule between the current round of workpieces and resources is selected.

**Step 4**: The global state of the scene is updated according to the information in the previous step, and the reward of this step can be calculated.

## 4 Case Study

In order to verify the effectiveness of the proposed model and algorithm, this paper takes the aircraft carrier deck replenishment scheduling scenario as a research case, and compares the PPO-based two-layer rule scheduling optimization algorithm proposed in this paper with several traditional meta-heuristic algorithms in solving the minimum completion time. The experimental results prove the effectiveness and better generalization of the algorithm proposed in this paper.

### 4.1 Experiment scenario

The issue of aircraft replenishment scheduling on the aircraft carrier deck is an important consideration in the design of the aircraft carrier deck. Due to the limitation of the aircraft carrier deck space, the aircraft's supply resources will be shared by the replenishment sites or machines. Therefore, this scenario constitutes a flexible job scheduling problem under resource preemption.

Under given resource occupancy and limited conditions, the PPO-based two-layer rule scheduling optimization algorithm proposed in this paper is compared with two traditional algorithms under three different machine distributions and different number of workpieces.

We make the following assumptions about the problem.

(1) All resources and machines are not damaged during the processing, and the transfer speed of all workpieces is 20.

(2) Each workpiece starts from the initial position $(0,0)$.

The specific information of the machine and workpiece is as follows.

The machine label and the resources it contains are shown in Table 3, and the resource sharing situation is also given. For example, from Table 3 we can see that machine 6 and machine 8 share resources $(0,1,7,2,8)$. Table 4 shows the position information of the three distributions. Table 5 shows the process information of a variety of workpieces.

The genetic algorithm, ant colony algorithm, and the algorithm proposed in this paper are used to solve and compare the scheduling of the three machine distribution scenarios. In the genetic algorithm, the results obtained due to mutation or crossover operations may not necessarily meet the constraints of resource conflicts. Here we perform decoding and correction operations on each offspring after it is generated to ensure that the correct results are obtained. In the ant colony algorithm, the resource preemption limit can be updated in the taboo table.

### 4.2 Result and analysis

#### 4.2.1 Total makespan

In view of the above-mentioned flexible job shop

**Table 3 Machine and resource information.**

| Machine label | Resource label | Shared machine label (shared resource label) |
|---|---|---|
| 0 | (0−8) | 1(0−8) |
| 1 | (0−8) | 0(0−8) |
| 2 | (0−8) | 3(0−8) |
| 3 | (0−8) | 2(0−8) |
| 4 | (0−8) | − |
| 5 | (0−8) | − |
| 6 | (0,1,7,2,8) | 8(0,1,7,2,8) |
| 7 | (3,4,5,6) | 9(3,4,5,6) |
| 8 | (0,1,7,2,8) | 6(0,1,7,2,8) |
| 9 | (3,4,5,6) | 7(3,4,5,6) |
| 10 | (0,1,7,2,8) | 12(0,1,7,2,8) |
| 11 | (3,4,5,6) | 13(3,4,5,6) |
| 12 | (0,1,7,2,8) | 10(0,1,7,2,8) |
| 13 | (3,4,5,6) | 11(3,4,5,6) |
| 14 | (0,1,7,2,8) | 16(0,1,7,2,8) |
| 15 | (3,4,5,6) | 17(3,4,5,6) |
| 16 | (0,1,7,2,8) | 14(0,1,7,2,8) |
| 17 | (3,4,5,6) | 15(3,4,5,6) |

**Table 4 Three machine distributions' information.**

| Machine label | Distribution 1 | Distribution 2 | Distribution 3 |
|---|---|---|---|
| 0 | (40,13.5) | (22, 4.1) | (7.3,17.2) |
| 1 | (38,14.5) | (36,14.5) | (38,14.5) |
| 2 | (36,15) | (34,15) | (36,15) |
| 3 | (34,15.8) | (32,15.8) | (34,15.8) |
| 4 | (32,16.4) | (5.1, 8.9) | (32,16.4) |
| 5 | (30,17.1) | (30,17.1) | (30,17.1) |
| 6 | (6,16.2) | (6,16.2) | (6,16.2) |
| 7 | (4.2,14) | (4.2,14) | (4.2,14) |
| 8 | (3.6,11.5) | (3.6,11.5) | (3.6,11.5) |
| 9 | (3.1,9.3) | (3.1, 9.3) | (3.1,9.3) |
| 10 | (7,8.4) | (7, 8.4) | (7, 8.4) |
| 11 | (11,7.6) | (11, 7.6) | (11,7.6) |
| 12 | (15, 6.6) | (15, 6.6) | (15, 6.6) |
| 13 | (19, 5.4) | (19, 5.4) | (19, 5.4) |
| 14 | (28.7,17.9) | (7.1,17.35) | (9.25,8.2) |
| 15 | (27.7,19.2) | (27.7,19.2) | (17.1,5.9) |
| 16 | (26.7, 20.6) | (9.17, 17.65) | (5.07, 8.7) |
| 17 | (24.7, 20.8) | (24.7, 20.8) | (13.3,7.08) |

**Table 5 Process information of different parts.**

| Workpiece type | Process list |
|---|---|
| 1 | [0,4,5,3,8,6,7,1,2] |
| 2 | [2,8,4,0,3,7,1,5,6] |
| 3 | [8,6,1,5,7,2,4,0,3] |
| 4 | [3,8,1,0,7,4,6,5,2] |
| 5 | [0,3,8,6,7,2,4,5,1] |
| 6 | [0,2,3,1,6,4,8,5,7] |
| 7 | [2,1,0,4,8,6,5,7,3] |
| 8 | [2,8,0,3,1,6,4,7,5] |
| 9 | [5,4,3,6,2,1,8,0,7] |
| 10 | [3,0,5,7,1,6,8,4,2] |
| 11 | [2,3,0,5,7,1,6,4,8] |
| 12 | [0,2,8,3,7,4,5,6,1] |
| 13 | [6,1,2,8,5,3,4,7,0] |
| 14 | [6,3,1,8,7,0,2,4,5] |
| 16 | [1,8,4,2,3,6,5,0,7] |
| 17 | [5,4,1,8,6,2,7,3,0] |
| 18 | [1,4,2,0,6,8,3,5,7] |
| 19 | [7,5,1,4,3,2,0,6,8] |
| 20 | [7,3,2,0,1,6,5,4,8] |

scheduling problem under resource preemption, it can be seen that the algorithm proposed in this paper is more effective in terms of completion time, and has better generalization of the scene for different machine distribution scenarios.

It can be seen from Figs. 2−4 that the genetic

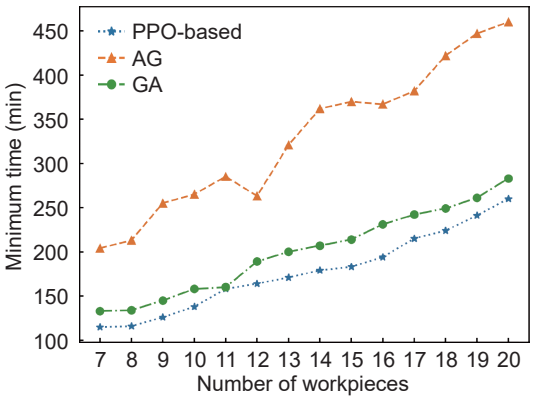


**Fig. 2 Distribution 1 scheduling result.**

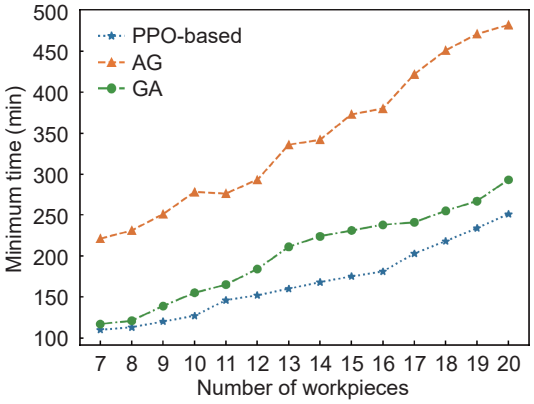


**Fig. 3 Distribution 2 scheduling result.**

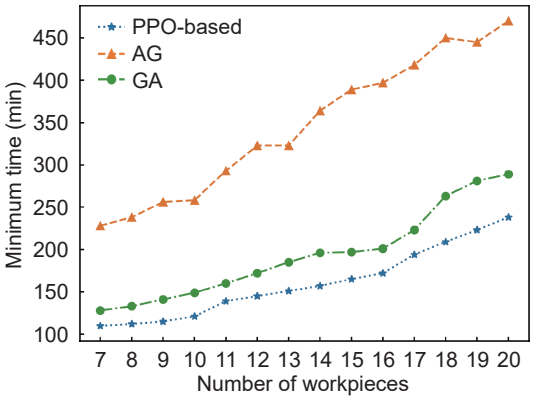


**Fig. 4 Distribution 3 scheduling result.**

algorithm and the PPO-based two-layer rule scheduling optimization algorithm perform better than the ant colony algorithm. When the number of workpieces is small, the algorithm proposed in this paper and the genetic algorithm basically have the same results. When the number of workpieces is large, the PPO-based two-layer rule scheduling optimization algorithm can find a better solution. The effect of particle swarm optimization is much worse. An important reason for

the above results is that the situation where the machine is unavailable at different time due to the limitation of resource preemption is more complicated, and it is more difficult for the traditional heuristic algorithm to search for the correct solution space. The algorithm proposed in this paper uses the mask to correct this problem, and returns the result of this mask to the network for training, so that such problems are avoided to a certain extent.

In the aspect of time-consuming, the genetic algorithm achieves a result similar to that of DRL and the number of iterations has reached 60 000 steps, which is time-consuming. Although the DRL-based algorithm takes much time to train, it is very fast when predicting specific scenarios. At the same time, due to the generalization of the deep reinforcement learning network, it is not sensitive to changes in the process. After a little further training on a trained model, a better solution can be obtained. But for traditional algorithms, it needs to be recalculated as long as the problem changes.

Figure 5 shows the optimal scheduling result gantt chart of the PPO-based two-layer rule scheduling optimization algorithm when the number of workpieces is 8.

#### 4.2.2 Ablation experiment

To determine the role of the components of the algorithm proposed in this paper, we conducted ablation experiments in distribution 1. Firstly, all the workpieces selected for participation in each iteration in Step 3 in Fig. 1 are selected as all acquisitions, that is, the agent does not select the workpieces to participate in this round of iterations. As long as the workpieces can be acquired, the scheduling will be arranged. The curve obtained by repeating the experiment several times is shown in the label of Ablation Test 1 (RT1) in Fig. 6. Secondly, we keep the agent to select the participating artifacts in each iteration and reduce the rule to a fixed one: rule 3 in Table 1, and repeat the experiment many times, the obtained curve is as shown in the label of Ablation Test 2 (RT2) in Fig. 6. It can be found from Fig. 6 that the minimum solution time of the two ablation test groups is not as good as the original one. And in the experiments we found that the algorithm converges very easily to the optimal rule in test 1, and in test 2 the algorithm easily converges to the case of involving all the workpiece in each iteration. It can be found that the combination of workpiece scheduling and rule selection in the algorithm proposed in this paper ensures that the algorithm can achieve a better solution.

#### 4.2.3 Model generalization

We apply the trained model base-model with 8 artifacts under distribution 1 to the new 4 distributions for brief training. This approach is similar to pre-training in natural language processing. Base-model is pre-training model, which can get relatively good results with only short training in new scenarios. The results obtained by multiple predictions on the new scene are shown in Fig. 7a. It can be seen that the algorithm proposed in this paper has good generalization.

#### 4.2.4 Dynamic test

In order to explore the impact of machine downtime on the scheduling results, in the scenario where the number of workpieces in Scenario 1 is 8, some resources are unavailable at the 30-th step of each simulation step, and the failure rates are set to 1%, 5%, and 10%. The results of multiple experiments are shown in Fig. 7b. It can be found that the proposed algorithm can still complete the scheduling task for dynamic situations, and basically does not affect the scheduling results when the failure rate is low.

## 5 Conclusion

The flexible job shop scheduling problem in a resource preemption environment makes it more difficult to solve this type of problem due to its stronger constraint. This article proposes a new method based on the DRL.

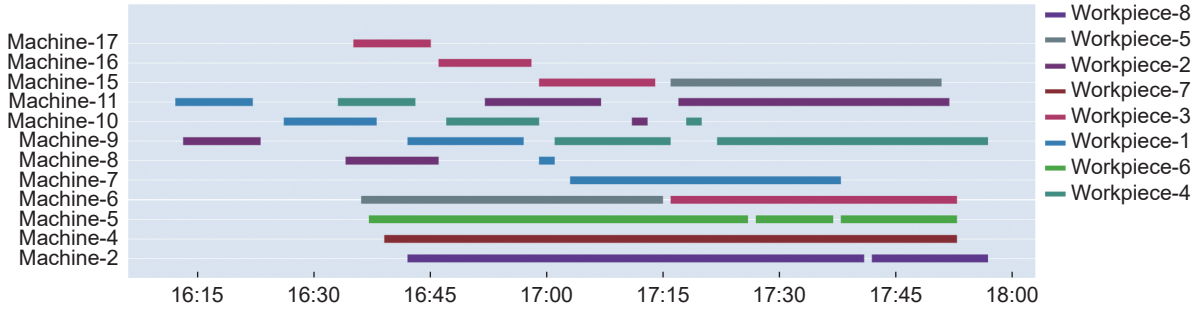


**Fig. 5 Distribution-1, 8 workpieces scheduling gantt chart (Dec 19, 2021).**

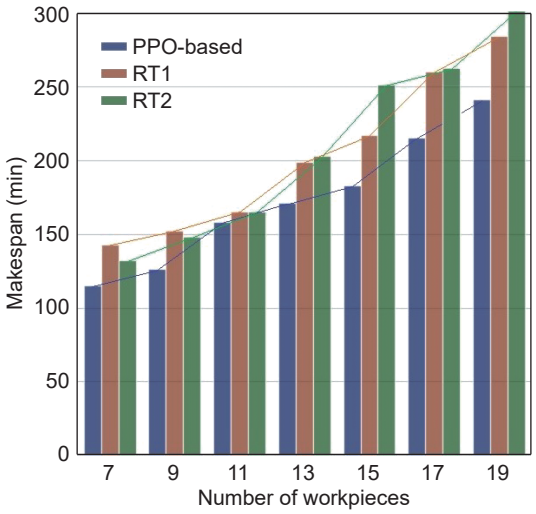


**Fig. 6 Ablation experiment.**

First, a scheduling model for resource preemption with minimum completion time is established. Then, based on the built model, a PPO-based two-layer rule scheduling algorithm is put forward. In our method, the algorithm satisfies the constraint relationship through a special solution sequence, and the artificial experience in the lower-level rules makes the model have a faster convergence speed and better optimization results. The results show that the algorithm proposed in this paper performes better and has better scenario generalization in the case of resource preemption, and it can solve the scheduling problem under dynamic conditions such as equipment damage.

## References

[1] L. Zhang, Research on flexible job shop scheduling based on multi-objective optimization algorithm, MA dissertation, College of Information Science and Engineering, Shandong Agricultural University, Taian, Shangdong, 2019.

[2] X. Zhu, Flexible job shop scheduling multi-object optimization based on strong reproduction NSGA-Ⅱ algorithm, *Modular Machine Tool and Automatic Manufacturing Technology*, no. 9, pp. 180–184, 2021.

[3] Y. Yuan and H. Xu, Flexible job shop scheduling using hybrid differential evolution algorithms, *Computers & Industrial Engineering*, vol. 65, no. 2, pp. 246–260, 2013.

[4] Q. Yu, L. Zhao, and S. Pan, A scheduling optimization of flexible job-shop using genetic algorithm, *Modular Machine Tool and Automatic Manufacturing Technology*, no. 4, pp. 32–34, 2004.

[5] H. Luo and D. Pan, Two-layer coding discrete cuckoo algorithm for solving flexible workshop scheduling problem, *Computer and Digital Engineering*, vol. 49, no. 7, pp. 1281–1285, 2021.

[6] W. Teekeng, A. Thammano, P. Unkaw, and J. Kiatwuthiamorn, A new algorithm for flexible job shop scheduling problem based on particle swarm optimization,

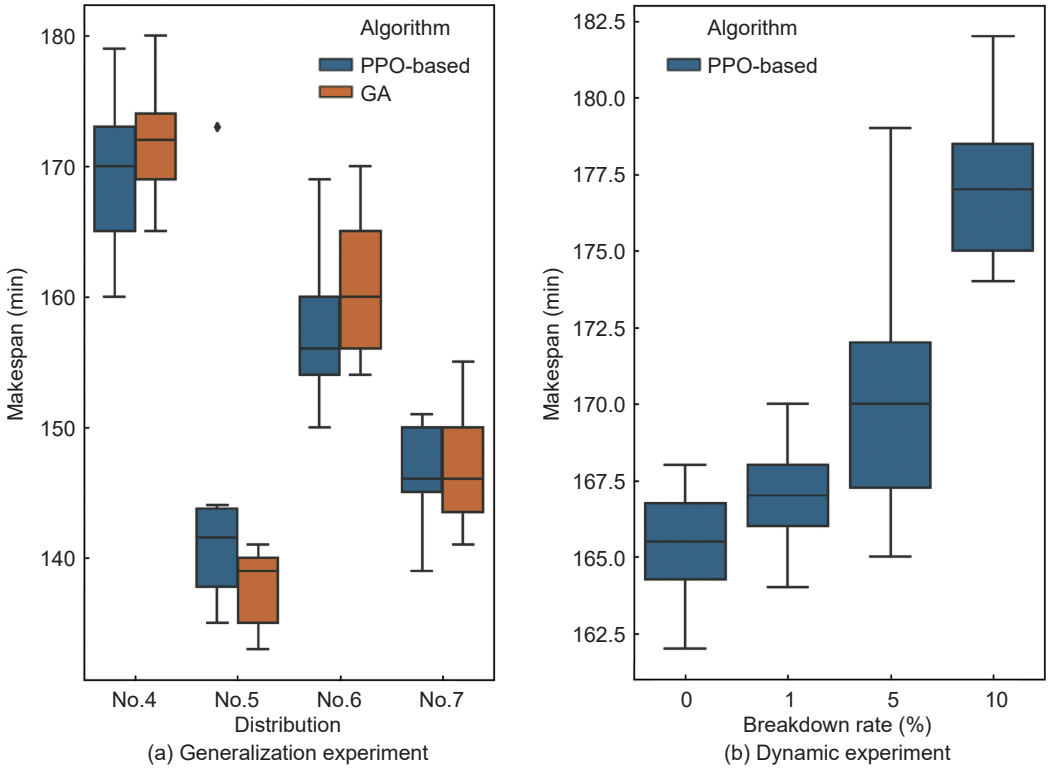


**Fig. 7 Generalization and dynamic experiments.**

*Artificial Life and Robotics*, vol. 21, pp. 18–23, 2016.

[7] V. Boyer, J. Vallikavungal, X. C. Rodríguez, and M. A. Salazar-Aguilar, The generalized flexible job shop scheduling problem, *Computers & Industrial Engineering*, vol. 160, p. 107542, 2021.

[8] W. Liu, Y. Gong, W. Chen, Z. Liu, H. Wang, and J. Zhang, Coordinated charging scheduling of electric vehicles: A mixed-variable differential evolution approach, *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 12, pp. 5094–5109, 2019.

[9] S. Zhou, L. Xing, X. Zheng, N. Du, L. Wang, and Q. Zhang, A self-adaptive differential evolution algorithm for scheduling a single batch-processing machine with arbitrary job sizes and release times, *IEEE Transactions on Cybernetics*, vol. 51, no. 3, pp. 1430–1442, 2019.

[10] B. C. Csáji, M. László, and B. Kádár, Reinforcement learning in a distributed market-based production control system, *Advanced Engineering Informatics*, vol. 20, no. 3, pp. 279–288, 2006.

[11] F. Zhao, X. He, and L. Wang, A two-stage cooperative evolutionary algorithm with problem-specific knowledge for energy-efficient scheduling of no-wait flow-shop problem, *IEEE Transactions on Cybernetics*, vol. 51, no. 11, pp. 5291–5303, 2020.

[12] F. Zhao, L. Zhao, L. Wang, and H. Song, An ensemble discrete differential evolution for the distributed blocking flowshop scheduling with minimizing makespan criterion, *Expert Systems with Applications*, vol. 160, p. 113678, 2020.

[13] L. Wang, Z. Pan, and J. Wang, A review of reinforcement learning based intelligent optimization for manufacturing scheduling, *Complex System Modeling and Simulation*, vol. 1, no. 4, pp. 257–270, 2021.

[14] T. Gabel and M. Riedmiller, Scaling adaptive agent-based reactive job-shop scheduling to large-scale problems, in *Proc. 2007 IEEE Symposium on Computational Intelligence in Scheduling*, Honolulu, HI, USA, 2007, pp. 259–266.

[15] Y. Martínez, A. Nowé, J. Suárez, and R. Bello, A reinforcement learning approach for the flexible job shop scheduling problem, in *Proc. 5th International Conference on Learning and Intelligent Optimization*, Rome, Italy, 2011, pp. 253–262.

[16] R. Chen, B. Yang, S. Li, and S. Wang, A self-learning genetic algorithm based on reinforcement learning for flexible job shop scheduling problem, *Computers and Industrial Engineering*, vol. 149, p. 106778, 2020.

[17] S. Luo, Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning, *Applied Soft Computing*, vol. 91, p. 106208, 2020.

[18] X. Wang, L. Zhang, T. Lin, C. Zhao, K. Wang, and Z. Chen, Solving job scheduling problems in a resource preemption environment with multi-agent reinforcement learning, *Robotics and Computer-Integrated Manufacturing*, vol. 77, p. 102324, 2022.

[19] C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and C. Xu, Learning to dispatch for job shop scheduling via deep reinforcement learning, arXiv preprint arXiv: 2010.12367, 2020.

[20] I. A. Chaudhry and A. A. Khan, A research survey: Review of flexible job shop scheduling techniques, *International Transactions in Operational Research*, vol. 23, no. 3, pp. 551–591, 2016.

[21] J. Peng, M. Liu, M. Zhang, and X. Zhang, Review on Scheduling Algorithms for MOFJSP, *China Mechanical Engineering*, vol. 23, pp. 3244–3254, 2014.

[22] M. V. Otterlo and M. Wiering, Reinforcement learning and markov decision processes, in *Reinforcement Learning*, M. Wiering and M. V. Otterlo, eds. Berlin, Germany: Springer-Verlarg, 2012, pp. 3–42.

[23] L. Graesser and W. L. Keng, *Foundations of Deep Reinforcement Learning: Theory and Practice in Python*. Boston, MA, USA: Addison-Wesley Professional, 2019.

[24] M. Cui, J. Wang, and M. Yue, Machine learning-based anomaly detection for load forecasting under cyberattacks, *IEEE Transactions on Smart Grid*, vol. 10, no. 5, pp. 5724–5734, 2019.

[25] T. Zhao and M. Eskenazi, Towards end-to-end learning for dialog state tracking and management using deep reinforcement learning, arXiv preprint arXiv: 1606.02560, 2016.

[26] N. Wei, *A Guide to the Implementation of Deep Reinforcement Learning* (in Chinese). Beijing, China: Publishing House of Electronics Industry, 2021.

[27] Y. Wang, H. He, and X. Tan, Truly proximal policy optimization, in *Proc. 35th Uncertainty in Artificial Intelligence*, Tel Aviv, Israel, 2019, pp. 113–122.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, Proximal policy optimization algorithms, arXiv preprint arXiv: 1707.06347, 2017.

**Zhen Chen** is currently pursuing the PhD degree at the School of Automation Science and Eletrical Engineering, Beihang University, Beijing, China. His research interests include deep reinforcement learning, jop shop scheduling problem, scheduling, and service recommendations in cloud manufacturing.

**Xiaohan Wang** is currently pursuing the PhD degree at the School of Automation Science and Eletrical Engineering, Beihang University, Beijing, China. His research direction is discrete simulation, multi-intelligence body systems, and reinforcement learning.

**Lin Zhang** received the BS degree from Nankai University, Tianjin, China, in 1986, and the MS and PhD degrees from Tsinghua University, Beijing, China, in 1989 and 1992, respectively. He is a professor with Beihang Unversity, Beijing. He authored and coauthored 200 papers, 18 books, and chapters. His research interests include service-oriented modeling and simulation, model engineering, cloud manufacturing and simulation and their applications in health, etc. He is a member of IEEE. He served as the president of the Society for Modeling and Simulation International (SCS) (2015−2016). He is a fellow of the SCS and Federation of Asian Simulation Societies (ASIASIM), and the executive vice president of the China Simulation Federation.

**Pengfei Gu** is currently pursuing the PhD degree at the School of Automation Science and Electrical Engineering in Beihang University. He received the master degree from University of Science and Technology Beijing in 2019. His research interests include modeling and simulation, system engineering, and evolutionary game theory.