*2025 International Conference on Intelligent Computing*
*July 26-29, Ningbo, China*
https://www.ic-icc.cn/2025/index.php

# Multi-Agent Reinforcement Learning with Cooperative Mechanism for Dynamic Job Shop Scheduling Problem

Jie Shang[1], Junqing Li[1,2][0000-0002-3617-6708] and Jiake Li[2]

[1] Shandong Normal University, Jinan 250014, China
[2] Yunnan Normal University, Kunming 650500, China
lijunqing@lcu-cs.com

**Abstract.** With the vigorous growth of the manufacturing industry, the complex dynamic scheduling problem has become the focus of enterprise production and cannot be ignored. Therefore, this paper established a mathematical model of the dynamic job shop scheduling problem (DJSP) with uncertain processing time. The optimization objective was to minimize the makespan. Firstly, a multi-agent reinforcement learning (MADRL) method was proposed, including two agents: proximal policy optimization (PPO) and deep Q-network (DQN). The DQN agent employs a collaborative mechanism to co-train with the PPO agent. It prioritizes high-value samples from the experience replay buffer and provides them to the PPO agent for feature extraction. Within the PPO agent, state features are represented by both job attributes and machine resource availability metrics. Different dispatching rules are assigned to feasible machines as action space, with the reward function defined by idle time during scheduling. Finally, we applied our method to a wide range of publicly available static benchmarks, with results demonstrating significant performance improvements. Furthermore, through comparative experiments with heuristic dispatching rules as well as PPO and DQN reinforcement learning algorithms, we further validated the versatility and superiority of our approach across dynamic scenarios of varying scales.

**Keywords:** Dynamic job shop scheduling, Uncertain processing time, Multi-agent reinforcement learning.

## 1 Introduction

With the advancement of manufacturing technologies, smart manufacturing has emerged as a key focus in both academia and industry. As one of its core challenges, production scheduling faces growing complexity due to various uncertain factors in manufacturing processes, leading to an exponential expansion in production data state characteristics and significantly increasing the difficulty of scheduling optimization.

The dynamic job shop scheduling problem (DJSP)has long been a key focus in industrial and manufacturing research. Traditional priority dispatching rules (PDRs) and heuristic algorithms [1] have been widely applied to the DJSP. However, current methods are often limited and singular, unable to effectively handle the dynamic factors in real-world scenarios. This highlights the need for more advanced algorithmic frameworks to tackle these challenges effectively.

To address these challenges, the main contributions of this paper are as follows: (1) An optimization model for the DJSP is developed, with the objective of minimizing makespan. (2) A multi-agent reinforcement learning (MADRL) dynamic scheduling framework based on deep Q-network (DQN) and proximal policy optimization (PPO) agents is proposed, effectively alleviating the complexity introduced by dynamic scheduling. (3) A state representation based on jobs and machines is designed, incorporating scheduling rules into the action space, and continuously optimizing the optimal solution using the reward function. (4) Experimental results indicate that our method is highly effective and generalizable both in static and dynamic scheduling environments. Even for larger instances, the learned policies exhibit strong performance.

The structure of the rest of this paper is as follows. Section 2 provides a literature review. Section 3 presents the model for dynamic job shop scheduling with uncertain processing times. Section 4 introduces the MADRL scheduling method. Section 5 presents the data from static and dynamic experiments, along with a discussion of the comparative results. Finally, Section 6 concludes the paper with a summary and outlook.

## 2 Literature review

In recent years, traditional PDRs and heuristic methods have been extensively explored in job shop scheduling, with heuristic [2] and meta-heuristic algorithms [3, 4] seeing the widest application. Zhang proposed a rescheduling method based on a genetic algorithm (GA) and tabu search for DJSP with random job arrivals and breakdown of machines [5]. Chen proposed a cooperative evolutionary algorithm to solve the flexible job shop scheduling problem (FJSP) [6]. Li introduced a co-evolutionary algorithm for the flexible flow shop scheduling problem (FSP) with robotic transportation [7]. Gao proposed an improved particle swarm optimization algorithm combined with reinforcement learning to solve the flexible job-shop scheduling problem [8]. While these approaches have demonstrated promising results in smaller-scale problems, they may encounter computational and temporal challenges when tackling larger-scale instances.

DRL frameworks have emerged as a powerful approach for job shop scheduling problems, owing to their exceptional learning capabilities and broad applicability. Zhao proposed the estimation of the reinforcement learning based distribution algorithm (RLEDA) to solve the energy-efficient DHFJSP while minimizing the makespan and total energy consumption [9]. Du designed a multi-objective DQN algorithm for the FJSP with crane transportation and setup times [10]. Luo addressed the DJSP with new job insertions using a double DQN algorithm [11]. Shahrabi proposed a scheduling method based on variable neighborhood search (VNS) to address the DJSP [12]. Lin introduced distributed edge computing into a smart factory framework, leveraging a multi-level neural network with a DRL framework to tackle the JSP [13]. Liu used a proximal policy optimization (PPO) algorithm to deal with DJSP with random job arrivals and random machine failures by minimizing the makespan [14]. Wu proposed a PPO algorithm based on experience replay with mixed priorities to solve the DJSP with uncertain processing time [15]. Hammami et al. combined PPO and Actor-Critic algorithms with the event-driven rescheduling strategy to solve the real-time JSP problem under uncertain job arrivals [16].

Multi-agent reinforcement learning is a new research direction in the field of scheduling. Yuan proposed a dual deep Q-networks (MADDQN) algorithm for FJSP, with multi-agent reinforcement learning application scheduling rules composed of workpiece agents and machine agents [17]. Zhang et al. proposed a multi-agent reinforcement learning system (MARLS) to solve DJSP and adopted an improved contract network protocol (CNP) to guide cooperation and competition among multiple agents [18].

However, their multi-agent approach treating jobs and machines as independent agents fails to account for algorithmic time complexity, potentially leading to model redundancy and state feature incompatibility. Therefore, the dynamic scheduling system urgently requires improved algorithms to analyze these problems effectively.

## 3 Problem description

### 3.1 Problem formulation

The DJSP with uncertain processing time in this paper is defined as follows. The set of $n$ jobs $J = \{J_1, J_2, \ldots, J_n\}$ needs to be processing on $m$ machines $M = \{M_1, M_2, \ldots, M_m\}$. Each job $J_i$ all has $n_i$ operations, among jth jobs named $O_{i,j}$. Every job selects machine $M_k$ to process noted $M_{i,j}(M_{i,j} \in M)$. An operation $O_{i,j}$ may be stranded on the current machine $M_k$ due to factors, and the uncertain processing time is denoted as $UP_{i,j,k}$. The optimization objective is to minimize the makespan, described as Equation (1).

$$Makespan = min\, C_{max} = min \sum C_i (1 \le i \le n) \tag{1}$$

The description of indices is given in Table 1, and the description of variables is given in Table 2. The decision of parameters is defined in Table 3.

**Table 1.** The description of indices.

| Indices | Related description |
|---|---|
| $i, i'$ | The index of jobs, $i, i' = 1, \ldots, n$ |
| $j, j'$ | The index of operations, $j, j' = 1, \ldots, m$ |
| $k$ | The index of machine, $k = 1, \ldots, m$ |

**Table 2.** The description of decision variables.

| Decision variables | Related description |
|---|---|
| $X_{i,j,k} = \begin{cases} 1 \\ 0 \end{cases}$ | If operations $O_{i,j}$ processed on machines $M_k$<br>Otherwise |
| $Y_{i,j,i',j',k} = \begin{cases} 1 \\ 0 \end{cases}$ | If operations $O_{i,j}$ processed on machines $M_k$ before $O_{i',j'}$<br>Otherwise |

**Table 3.** The description of parameters.

| Parameters | Related description |
|---|---|
| $n$ | The number of jobs |
| $m$ | The number of machines |
| $J$ | The set of jobs, $J = \{J_1, J_2, \ldots, J_n\}$ |
| $M$ | The set of machines, $M = \{M_1, M_2, \ldots, M_m\}$ |
| $O$ | The set of operations,$O = \{O_{11}, O_{12}, \ldots, O_{1j}, \ldots, O_{i,j}\}$ |
| $J_i$ | Jobs jth, for $j = 1, \ldots, m$ |
| $M_k$ | Machines $i$th, for $i = 1, \ldots, n$ |
| $C_i$ | The processing time of the jobs $J_i$ |
| $C_{max}$ | Makespan |
| $O_{i,j}$ | Operations $j$th of the jobs $J_i$ |
| $M_{i,j}$ | The processing machines set of operations $O_{i,j}$, $M_{i,j} \in M$ |
| $C_{i,j}$ | The completion time of operations $O_{i,j}$ |
| $S_{i,j}$ | The start processing time of operations $O_{i,j}$ |
| $P_{i,j,k}$ | The processing time of operations $O_{i,j}$ on machines $M_k$ |
| $UP_{i,j,k}$ | The processing time with uncertain time |

The constraints of DJSP are given by the Constraints (3-1) - (3-6).

$$C_{i,j} \geq 0, S_{i,j} \geq 0, C_i \geq 0, \forall i, j \qquad (3-1)$$

$$S_{i,j} \geq C_{i,j-1}, \forall i, \forall j = 2, \ldots, n+1 \qquad (3-2)$$

$$C_{i,j} \geq S_{i,j} + P_{i,j,k} \cdot X_{i,j,k}, \forall i, j, k \qquad (3-3)$$

$$\sum_{k \in M_{i,j}}^{m} X_{i,j,k} = 1, \forall i, j \qquad (3-4)$$

$$S_{i,j} \geq C_{i\prime,j\prime} - Y_{i,j,i\prime,j\prime,k} \times N^*, \forall i, j, i', j', \forall M_k \in M_{i,j} \qquad (3-5)$$

$$S_{i\prime,j\prime} \geq C_{i,j} - \left(1 - Y_{i,j,i\prime,j\prime,k}\right) \times N^*, \forall i, j, i', j', \forall M_k \in M_{i,j} \qquad (3-6)$$

Constraints (3-1) represent the non-negative constraints on the processing time of jobs. Constraint (3-2) represents the processing sequence constraint between different operations of the same jobs. Constraint (3-3) represents the constraint correlation between the start and end times of the operations processing under the same jobs. Constraint (3-4) represents the constraint of machine selection, which is that each operation should only be assigned to one machine for processing within the same machine. Constraints (3-5) and (3-6) represent the processing sequence relationship between different operations of different jobs on the same machine.

## 3.2 Uncertain processing time

$UP_{i,j,k}$ denotes the uncertain processing time of an operation. Each randomly generated instance contains two sets of distributions. The uncertainty of the processing time was weighted using the normal distribution α in Equation (2) and uniform distribution β in Equation (3) to obtain the value of the uncertainty ratio. Detailed values are outlined in Table 4.

$$F(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{2}$$

$$F(x) = \begin{cases} \frac{1}{b-a}, a \le x \le b \\ 0, else \end{cases} \tag{3}$$

**Table 4.** Uncertain processing time

| Distribution | α | β |
|---|---|---|
| Normal | 1,2,3 | ±25%,50%,75%,100% |
| Uniform | 1,2,3 | ±25%,50%,75%,100% |

## 4 Our method for DJSP

### 4.1 Multi-agent method for scheduling

In multi-agent reinforcement learning (MARL), each agent is required to determine its actions based on its own local observations. MARL method extends agents to utilize global information for policy updates during the training phase. However, during execution, each agent makes decisions based solely on its individual local observations.

The structure of the MARL method is represented in Fig. 1. This method consists of two agents: PPO and DQN. Within the multi-agent collaborative mechanism, DQN and PPO agents share a unified objective function aimed at maximizing rewards derived from a common reward function, thereby achieving global optimization.

Initially, the experience replay buffer samples data from the environment. The DQN agent interacts with the sampled trajectory and replays experienced samples based on value function optimization advantages. Subsequently, the actor network in the PPO agent processes input tuple $(S, A_i, R, S')$ optimizes network weight coefficients $\theta$, and selects action $A$ to pass to Critic network. The Critic network performs action evaluation to obtain $Q(A)$ for policy gradient calculation. Another pair of Actor-Critic networks known as target networks process output next state $S'$. The Actor network in the target network selects action $A_{t+1}$ while the critic network combines output $Q(A_{t+1})$ with output $Q(A_i)$ from PPO agent to calculate Loss function. Finally, PPO agent copies obtained $Q', \theta'$ to target network. Immediate reward $R$ generated by each episode is fed back for optimizing makespan as optimization targets by the multi-agent method for scheduling.

We propose a framework for DJSP based on a MADRL method illustrated in Fig. 2. In this study, we integrate the value function-based DQN agent with the policy gradient-based PPO agent, utilizing the cooperative mechanism between each other for scheduling. Algorithm 1 describes the specific details of the DJSP algorithm based on MADRL.

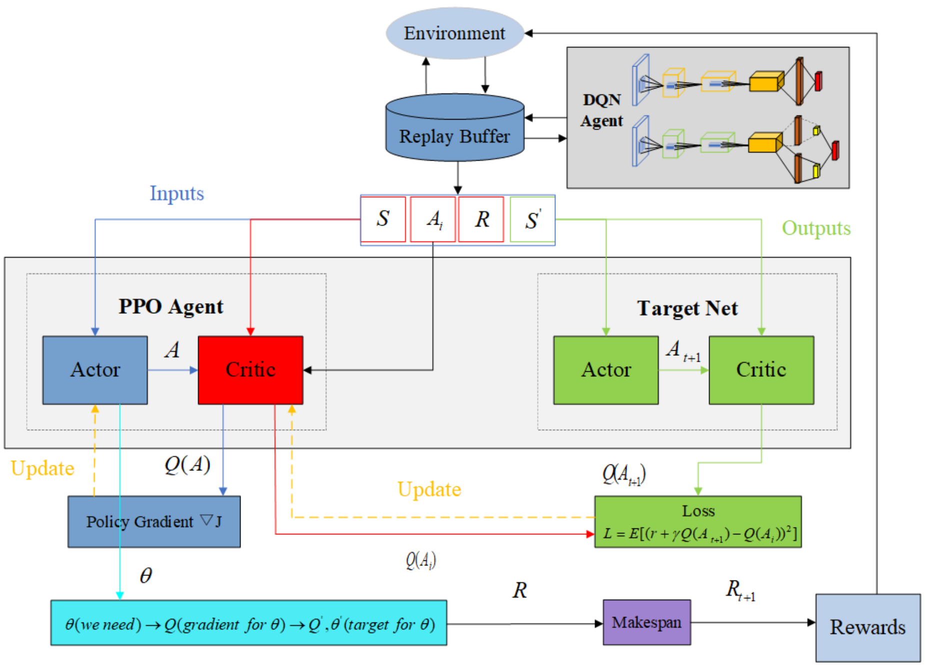

**Fig. 1.** Flow of the MARL algorithm

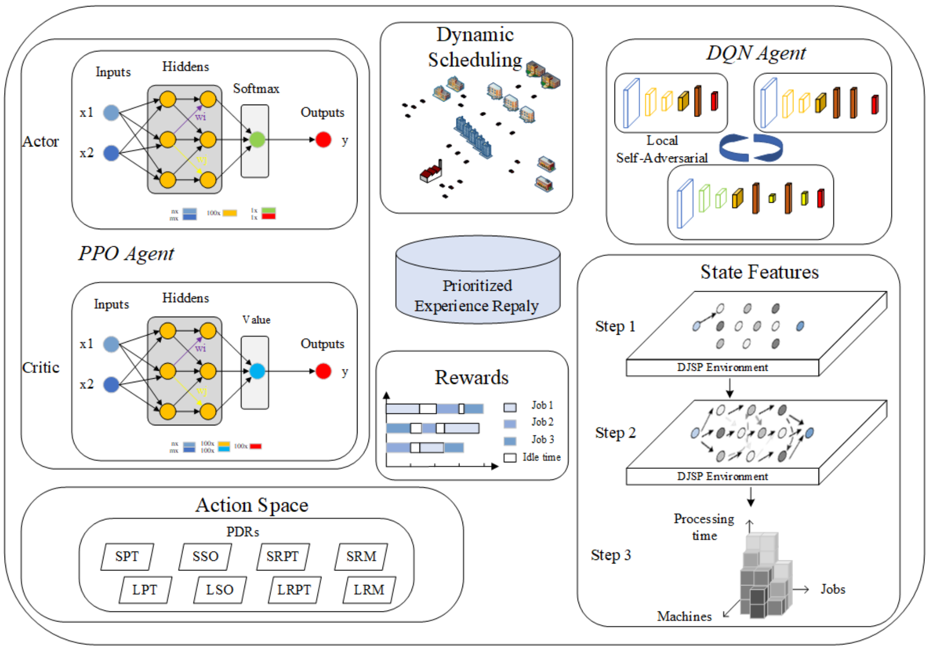

**Fig. 2.** Multi-agent method for scheduling

**Algorithm 1:** The MARL based on DQN and PPO agents for DJSP

```
1:  Initialize experience replay buffer $\mathcal{D}$, batch size $b$, buffer maximum size $\mathcal{N}$,
    epsilon $\varepsilon$, policy network $Q$ with $\theta$, target network $Q'$ with $\theta'$ in DQN agent
2:  Initialize epoch $M$, episode $N$, batch size $b$, mini buffer $m$, trajectories $T$,
    discount factor $\gamma$, Critic network $Q(s,a|\theta^Q)$ and Actor network $Q(s,a|\theta^u)$
    in PPO agent
3:  for $epoch = 1,2,\dots,M$ do
4:    for $episode = 1,2,\dots,N$ do
5:    DQN agent:
6:    Observe state $s_t$, action $a_t$, reward $r_t$ and $x_{t+1}$
7:    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
8:    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
9:    Sample minibatch transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
10:   Compute $y_j$ and perform a gradient descent step on $\left(yj - Q(\phi j, aj; \theta)\right)^2$
11:   Optimize and update $\theta' \leftarrow \theta$
12:   Update policy network $Q$ and target network $Q'$
13:   PPO agent:
14:   for $trajectories = 1,2,\dots,T$ do
15:     Obtain buffer m with $(\phi_j, a_j, r_j, \phi_{j+1})$ from DJSP environment
16:     Run policy $\theta^u$ in DJSP environment for $T$ trajectories
17:     Compute generalized advantage function
            $\widehat{A_t} = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}$,
            $where\ \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$
18:   end for
19:   Optimize surrogate objective $L(\theta)$ with mini buffer $m$, $L(\theta) =
    E_t\left[\min\left(r_t(\theta)\widehat{A_T}, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\widehat{A_T}\right)\right]$
20:   Optimize and update $\theta^Q \leftarrow \theta^u$
21:   Update Actor network $Q(s,a|\theta^u)$ and Critic network $Q(s,a|\theta^Q)$
22:   end for
23: end for
```

## 4.2 State feature

Given the complexity and variability of the DJSP, a 3D vector coordinate system is designed to capture the state features. As illustrated in Figure 3, the jobs and machines are initially represented using a traditional disjunctive graph. Next, the edges corresponding to feasible solutions are connected based on the scheduling relationships, forming the final directed acyclic graph. The critical path is then derived from the 3D coordinates, where the X-axis represents the job sequence number, the Y-axis represents the machine sequence number, and the Z-axis represents the processing time. The state representation is given by Equation (4).

$$[Jobs_n][Machines_m][PT_t] = DR/TOS \tag{4}$$

where $Jobs_n$ and $Machines_m$ denotes the $n$th job and $m$th machine. $PT_t$ indicates the processing time for the current operation, while $DR$ refer to the currently selected dispatching rules. $TOS$ represents the total operations.

## 4.3 Action space

The action space of this study is defined as follows: At each time step $t$, the selection is made based on the current state of the dynamic job shop scheduling environment, which includes the state features of machines and jobs.

The agent probabilistically chose a dispatching rule by the policy network from the action space. For different production states, we define these priority dispatching rules (PDRs) in Table 5 as the shortest processing time (SPT), the longest processing time (LPT), the shortest remaining machine without itself (SRM), the longest remaining machine without itself (LRM), the shortest remaining processing time (SRPT), the longest remaining processing time (LRPT), the shortest processing time with next operation (SSO), and the shortest processing time with next operation (LSO). By selecting these dispatching rules, the objective is to minimize the makespan.

**Table 5.** Action space of PDRs with description

| PDRs | Related description |
|---|---|
| SPT | The next selected job is according to the shortest processing time. |
| LPT | The next selected job is according to the largest processing time. |
| SSO | The next selected job is according to the minimum processing time of the subsequent operation. |
| LSO | The next selected job is according to the largest processing time of the subsequent operation. |
| SRPT | The next selected job is according to the minimum remaining machining time. |
| LRPT | The next selected job is according to the largest remaining machining time. |
| SRM | The next selected job is according to the minimum processing time of the subsequent operation without itself. |
| LRM | The next selected job is according to the largest processing time of the subsequent operation without itself. |

These rules are designed to select eligible processing operations and assign them to the appropriate machines. During the training process, the objective of minimizing the makespan is continuously optimized.

## 4.4 Reward function

In RL, the ultimate task of an agent is to maximize the cumulative reward. In scheduling optimization problems, however, the objective is typically to minimize the makespan for all jobs. Therefore, we align the reward function with the makespan based on machine idle time, as shown in Equation (5).

$$Reward = -Idle\ time = \sum C_i (1 \le i \le n) - M * Makespan \qquad (5)$$

where $Reward$ represents the total reward after algorithm training, $Idle\ time$ denotes the total idle time across all machines, $C_i$ is the completion time of the ith job, $n$ is the total number of jobs, $M$ represents the number of machines, and $Makespan$ refers to the actual minimum completion time of all jobs.

Specifically, after each scheduling solution is completed, the idle time of the machines is compared to the makespan. By optimizing the objective function, the cumulative reward during the network training process is effectively maximized, ensuring consistency between the optimal solution of the problem and the reinforcement learning reward function.

## 5 Experiments and Results

### 5.1 Model and configurations

For the PPO agent, the epoch is set to 100 with 10 episodes per epoch. The actor network's input layer corresponds to the state space dimensions, followed by two hidden layers with 100 nodes each. Action probabilities are normalized using a softmax function and mapped to a 1-D vector. The critic network has a similar structure but outputs a state-value function. The learning rates for both the actor network and the critic network are set to 0.001. The clipping ratio coefficient for the actor network is 0.2, and the number of hidden layers corresponds to the dimensions of the training instances, specifically the number of jobs multiplied by the number of machines. In the experience replay buffer, the priority weight factor for the prioritized experience replay mechanism is set to 0.6, the memory size is configured to 10, and the target network is updated every 10 iterations. The capacity of the experience replay buffer is determined by the memory size multiplied by the dimensionality of the state space. The batch size is proportional to the scale of the problem instance.

For the DQN agent, the memory capacity of the experience replay mechanism is set to 10. Each sampled instance is iterated for 1000 episodes, with each episode consisting of 10 timesteps. The batch size is configured to 128, and the exploration factor for the policy is set to 0.05. The target network is updated every 100 episodes, and the discount factor for the reward function is set to 0.8. Certain network layers are shared, while personalized policy layers are retained. This design enables the agent to capture global environmental dynamics while optimizing its policy through local interactions.

All remaining parameters are set to the default values in Pytorch. The selected hyperparameters strike a balance between convergence speed and training stability. Experiments were conducted on a system equipped with an Intel Core i7-11800 CPU and a single Nvidia GeForce RTX 3060 GPU.

### 5.2 Static experiments

We first selected a set of benchmark instances from the OR-Library, including la01, la02, ..., la40 [19], and some instances from ta21, ta22, ..., ta51, ta52 [20], representing different scales. These instances were used as a static test set to evaluate our method. Specifically, we chose instances of the following sizes for training and testing: 10 × 5, 15 × 5, 20 × 5, 10 × 10, 15 × 10, 20 × 10, 30 × 10, 15 × 15, 20 × 20, 30 × 15, 30 × 20, and 50 × 15.

As shown in Fig. 3, we present the makespan obtained after training, comparing it with the known optimal solutions of the benchmark instances. We calculate the accuracy as shown in Equation (6).

$$Accuracy = \left(1 - \frac{|C_{Ours} - C_{Opt}|}{C_{Opt}}\right) * 100\% \qquad (6)$$

where the $C_{Ours}$ represents the makespan obtained by our method. The $C_{Opt}$ refers to the optimal solution for the given benchmark case. The green values in Fig. 3 illustrates the accuracy between the $C_{Ours}$ and the $C_{Opt}$, which represented the precision of the optimal and worst makespan obtained by our method.

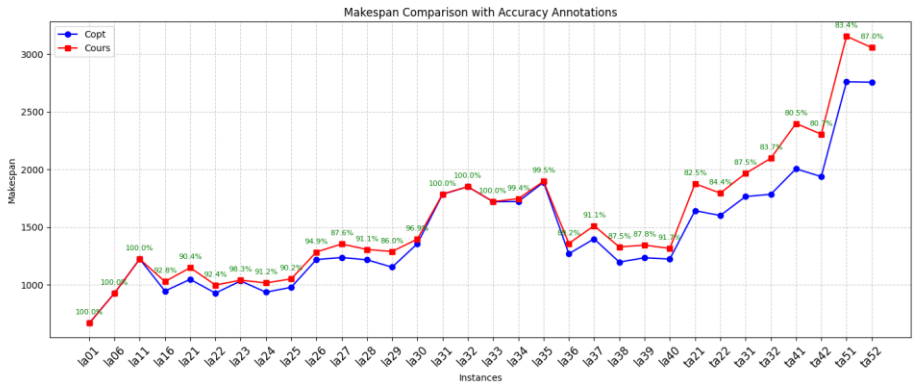

**Fig. 3.** Comparison with the Opt

The method refines the dispatching rules by utilizing the action space, and it selects the appropriate scheduling strategy for job sequencing based on the optimization of the policy gradient. By considering machine idle times, our approach ultimately produces more effective scheduling solutions in static experiments.

## 5.3 Dynamic experiments

Subsequently, we conducted dynamic experiments using 36 instances randomly generated from three different scales: 20 × 10, 30 × 10, and 30 × 20. These instances of uncertain processing time were generated with normal and uniform distributions, as described in Section 3.2.

The Fig. 4 and Fig. 5 respectively illustrate the change in makespan and loss during training on different scale instances. For the small-scale instances of 20 × 10, we can observe the evolution of the current makespan and the optimal makespan after training the model and fine-tuning the hyperparameters. The fluctuations indicate that the agent is exploring different learning policies, ultimately converging to the optimal makespan. And the loss gradually converges to 0 after training. In the PPO agent, the loss change of the actor represents the core loss function, which governs the stability and efficiency of policy updates. This loss is computed using a gradient ascent approach.

Our model achieves performance on the medium-scale instances comparable to the excellent results obtained on the small-scale instances. The effectiveness of our method was validated on the 30 × 20 large-scale instances. Although the training time increased, the learning capability was significantly enhanced, and the convergence speed of the loss improved. When compared to the results from previous instances, the model demonstrated consistent applicability across different scales.

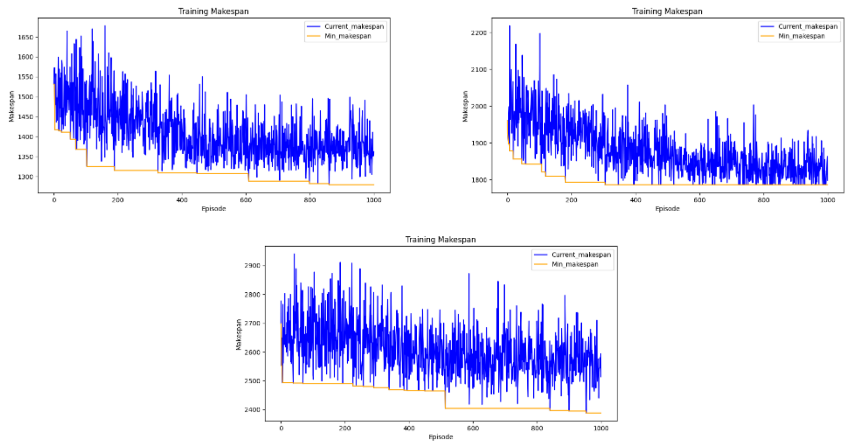

**Fig. 4.** Makespan of different instances

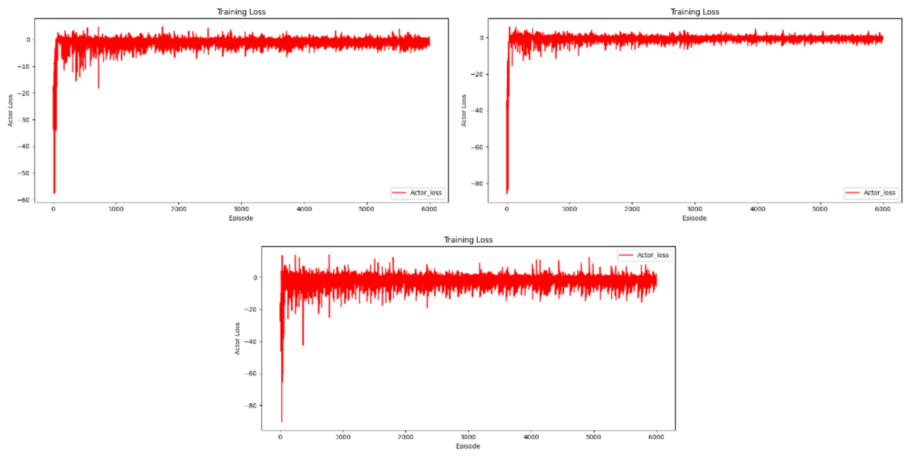

**Fig. 5.** Loss of different instances

Finally, we compared our method with five priority dispatching rules (PDRs), PPO, and DQN algorithms across various scales, as presented in Table 6. The results demonstrate that: (1) Solutions derived from PDRs exhibit inferior quality with notable limitations, particularly manifested through excessive machine idle time. (2) Our method achieves significantly better makespan performance than the optimal PDR solutions across all test instances. (3) Superiority over DRL Baselines: Our method demonstrates stronger optimization capability than both PPO and DQN algorithms. Specifically, PPO achieves an intermediate balance, with solutions consistently falling between those of the proposed method and DQN. DQN exhibits high solution variance across the same-scale instances, indicating instability in its convergence behavior.

**Table 6.** Comparison of dynamic experiments

| Instances | SPT | LPT | SRM | LRM | SRPT | PPO | DQN | Ours | Time |
|---|---|---|---|---|---|---|---|---|---|
| 20x10_25%_1 | 1986 | 1742 | 2182 | 1538 | 2020 | 1371 | 1634 | 1279 | 149.3s |
| 20x10_25%_2 | 1913 | 1692 | 2088 | 1512 | 2130 | 1379 | 1727 | 1280 | 149.8s |
| 20x10_25%_3 | 2137 | 1810 | 2180 | 1489 | 2043 | 1391 | 1784 | 1295 | 152.5s |
| 20x10_50%_1 | 1881 | 1663 | 2114 | 1543 | 2029 | 1372 | 1785 | 1289 | 150.0s |
| 20x10_50%_2 | 1967 | 1691 | 2044 | 1506 | 2049 | 1398 | 1630 | 1277 | 152.0s |
| 20x10_50%_3 | 2046 | 1840 | 2193 | 1469 | 2056 | 1388 | 1826 | 1289 | 151.7s |
| 20x10_75%_1 | 2077 | 1720 | 2141 | 1511 | 2081 | 1385 | 1712 | 1286 | 151.1s |
| 20x10_75%_2 | 1949 | 1628 | 2248 | 1521 | 2039 | 1382 | 1833 | 1281 | 152.9s |
| 20x10_75%_3 | 1933 | 1743 | 2104 | 1536 | 2048 | 1353 | 1669 | 1310 | 172.7s |
| 20x10_100%_1 | 2161 | 1719 | 2081 | 1475 | 2113 | 1356 | 1665 | 1285 | 202.7s |
| 20x10_100%_2 | 1890 | 1657 | 2166 | 1459 | 2095 | 1363 | 1757 | 1291 | 208.9s |
| 20x10_100%_3 | 2200 | 1869 | 2085 | 1507 | 2166 | 1336 | 1641 | 1294 | 207.7s |
| 30x10_25%_1 | 2417 | 2410 | 2797 | 1929 | 2643 | 1831 | 2169 | 1786 | 348.8s |
| 30x10_25%_2 | 2414 | 2352 | 2737 | 1924 | 2848 | 1819 | 2191 | 1782 | 341.3s |
| 30x10_25%_3 | 2452 | 2337 | 2595 | 2015 | 2981 | 1830 | 2242 | 1776 | 286.3s |
| 30x10_50%_1 | 2425 | 2419 | 2684 | 1951 | 2808 | 1803 | 2131 | 1787 | 281.1s |
| 30x10_50%_2 | 2332 | 2324 | 2814 | 1943 | 2793 | 1822 | 2185 | 1781 | 276.6s |
| 30x10_50%_3 | 2273 | 2562 | 2709 | 2004 | 2852 | 1810 | 2241 | 1776 | 265.3s |
| 30x10_75%_1 | 2366 | 2406 | 2579 | 2000 | 2791 | 1830 | 2223 | 1789 | 273.8s |
| 30x10_75%_2 | 2623 | 2353 | 2876 | 1983 | 2775 | 1803 | 2218 | 1768 | 278.7s |
| 30x10_75%_3 | 2593 | 2239 | 2911 | 1926 | 2981 | 1797 | 2197 | 1777 | 277.9s |
| 30x10_100%_1 | 2415 | 2457 | 2780 | 1957 | 2829 | 1794 | 2349 | 1788 | 286.1s |
| 30x10_100%_2 | 2360 | 2418 | 2853 | 1998 | 2869 | 1808 | 2393 | 1785 | 275.0s |
| 30x10_100%_3 | 2376 | 2521 | 2685 | 1937 | 2806 | 1823 | 2284 | 1777 | 290.1s |
| 30x20_25%_1 | 3075 | 3145 | 3533 | 2735 | Null | 2452 | 3297 | 2388 | 845.8s |
| 30x20_25%_2 | 3060 | 3240 | 3285 | 2806 | 3393 | 2520 | 3391 | 2379 | 796.2s |
| 30x20_25%_3 | 3115 | Null | 3278 | Null | Null | 2502 | 3393 | 2420 | 826.8s |
| 30x20_50%_1 | 3189 | 3397 | 3432 | 2749 | 3595 | 2465 | 3172 | 2403 | 960.9s |
| 30x20_50%_2 | 3236 | 3344 | Null | 2810 | 3283 | 2479 | 3484 | 2419 | 772.0s |
| 30x20_50%_3 | 3269 | Null | 3201 | 2829 | 3262 | 2450 | 3201 | 2384 | 824.1s |
| 30x20_75%_1 | 3159 | 3336 | 3324 | 2692 | Null | 2506 | 3126 | 2383 | 834.5s |
| 30x20_75%_2 | 3243 | Null | 3386 | 2756 | 3343 | 2477 | 3254 | 2396 | 735.9s |
| 30x20_75%_3 | 3346 | Null | 3294 | Null | Null | 2479 | 3343 | 2388 | 728.2s |
| 30x20_100%_1 | 3153 | 3119 | 3432 | 2688 | 3237 | 2522 | 3148 | 2394 | 726.2s |
| 30x20_100%_2 | 3306 | Null | 3475 | Null | Null | 2529 | 3237 | 2392 | 744.1s |
| 30x20_100%_3 | 3165 | Null | 3392 | 2742 | 3366 | 2465 | 3367 | 2383 | 758.8s |

(4) Under varying uncertain processing time conditions across different test groups, our method maintains consistent makespan performance with minimal deviation (<5% fluctuation). This stability demonstrates remarkable robustness against dynamic variations in processing time. (5) Notably, the proposed method demonstrates enhanced generalization capability in large-scale instances.

These results comprehensively validate the superiority and effectiveness of the multi-agent learning strategy for dynamic scheduling problems, offering a novel solution paradigm for optimization challenges in complex production environments.

## 6 Conclusion

In this study, we investigate the DJSP with uncertain processing time based on the MADRL algorithm. A mathematical optimization model of mixed integer linear programming is proposed. The MADRL scheduling algorithm, combining the DQN agent and PPO agent is established. The state features, action space, and reward function in dynamic scheduling are designed. Experimental results demonstrate that MADRL outperforms heuristic scheduling rules, PPO, and DQN algorithms, and shows remarkable performance on different scale instances.

MADRL is an emerging DRL method for scheduling problems, offering significant research potential. In future work, we will continue to develop more effective network models to further enhance the efficiency and convenience of using MARL for solving scheduling optimization problems.

**Acknowledgments.** This research is partially supported by the National Science Foundation of China under Grant (62473331, 62173216), Key projects of Yunnan Province Basic Research Program (202401AS070036), Yunnan Key Laboratory of Modern Analytical Mathematics and Applications (No. 202302AN360007)

**Disclosure of Interests.** The authors have no competing interests to declare that are relevant to the content of this paper.

## References

1. Ziaee, M.: A heuristic algorithm for solving flexible job shop scheduling problem. The International Journal of Advanced Manufacturing Technology 71, 519-528 (2014)
2. Ge, H., Sun, L., Liang, Y., Qian, F.: An effective PSO and AIS-based hybrid intelligent algorithm for job-shop scheduling. IEEE transactions on systems, man, and cybernetics-part a: systems and humans 38, 358-368 (2008)
3. Wang, Y.: A new hybrid genetic algorithm for job shop scheduling problem. COMPUT OPER RES 39, 2291-2299 (2012)
4. Tseng, S., Tsai, C., Chen, J., Chiang, M., Yang, C.: Job shop scheduling based on ACO with a hybrid solution construction strategy.: 2011 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2011). 2922-2927. IEEE (2011)
5. Zhang, L., Gao, L., Li, X.: A hybrid genetic algorithm and tabu search for a multi-objective dynamic job shop scheduling problem. INT J PROD RES 51, 3516-3531 (2013)

6. Chen, X., Li, J., Wang, Z., Li, J., Gao, K.: A genetic programming based cooperative evolutionary algorithm for flexible job shop with crane transportation and setup times. APPL SOFT COMPUT 112614 (2024)
7. 7. Li, J., Li, R., Li, J., Yu, X., Xu, Y.: A multi-dimensional co-evolutionary algorithm for multi-objective resource-constrained flexible flowshop with robotic transportation. APPL SOFT COMPUT 170, 112689 (2025)
8. Gao, Y., Shang, Q., Yang, Y., Hu, R., Qian, B.: Improved particle swarm optimization algorithm combined with reinforcement learning for solving flexible job shop scheduling problem.: International Conference on intelligent computing. 288-298. Springer (2023)
9. Zhao, F., Li, M.: Reinforcement Learning-Based Estimation of Distribution Algorithm for Energy-Efficient Distributed Heterogeneous Flexible Job Shop Scheduling Problem.: International Conference on Intelligent Computing. 183-195. Springer (2024)
10. Du, Y., Li, J., Li, C., Duan, P.: A reinforcement learning approach for flexible job shop scheduling problem with crane transportation and setup times. IEEE T NEUR NET LEAR 35, 5695-5709 (2022)
11. Luo, S.: Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. APPL SOFT COMPUT 91, 106208 (2020)
12. Shahrabi, J., Adibi, M.A., Mahootchi, M.: A reinforcement learning approach to parameter estimation in dynamic job shop scheduling. COMPUT IND ENG 110, 75-82 (2017)
13. Lin, C., Deng, D., Chih, Y., Chiu, H.: Smart manufacturing scheduling with edge computing using multiclass deep Q network. IEEE T IND INFORM 15, 4276-4284 (2019)
14. Liu, C., Huang, T.: Dynamic job-shop scheduling problems using graph neural network and deep reinforcement learning. IEEE Transactions on Systems, Man, and Cybernetics: Systems 53, 6836-6848 (2023)
15. Wu, X., Yan, X., Guan, D., Wei, M.: A deep reinforcement learning model for dynamic job-shop scheduling problem with uncertain processing time. ENG APPL ARTIF INTEL 131, 107790 (2024)
16. Hammami, N.E.H., Lardeux, B., B. Hadj-Alouane, A., Jridi, M.: Design and calibration of a DRL algorithm for solving the job shop scheduling problem under unexpected job arrivals. FLEX SERV MANUF J 1-32 (2024)
17. 17. Yuan, M., Huang, H., Li, Z., Zhang, C., Pei, F., Gu, W.: A multi-agent double deep-Q-network based on state machine and event stream for flexible job shop scheduling problem. ADV ENG INFORM 58, 102230 (2023)
18. Zhang, Y., Zhu, H., Tang, D., Zhou, T., Gui, Y.: Dynamic job shop scheduling based on deep reinforcement learning for multi-agent manufacturing systems. ROBOT CIM-INT MANUF 78, 102412 (2022)
19. Lawrence, S.: Resouce constrained project scheduling: An experimental investigation of heuristic scheduling techniques (Supplement). Graduate School of Industrial Administration, Carnegie-Mellon University (1984)
20. Taillard, E.: Benchmarks for basic scheduling problems. EUR J OPER RES 64, 278-285 (1993)