China Instrument and Control Society
Instrumentation
INSTRUMENTATION, Vol. 11, No. 1, March 2024 https://doi.org/10.15878/j.instr.202300165

Article

Deep Reinforcement Learning Solves Job-shop Scheduling Problems

Anjiang Cai[1], Yangfan Yu[1,*] and Manman Zhao[2]

[1] School of Mechanical and Electrical Engineering, Xi 'an University of Architecture and Technology, Xi 'an 710055, China

[2] Department of Automation Engineering, Wuxi Higher Vocational and Technical School of Mechanical and Electrical Engineering, Wuxi 214028, China

[*] Corresponding author email: 1732900847@qq.com

Abstract: To solve the sparse reward problem of job-shop scheduling by deep reinforcement learning, a deep reinforcement learning framework considering sparse reward problem is proposed. The job shop scheduling problem is transformed into Markov decision process, and six state features are designed to improve the state feature representation by using two-way scheduling method, including four state features that distinguish the optimal action and two state features that are related to the learning goal. An extended variant of graph isomorphic network GIN++ is used to encode disjunction graphs to improve the performance and generalization ability of the model. Through iterative greedy algorithm, random strategy is generated as the initial strategy, and the action with the maximum information gain is selected to expand it to optimize the exploration ability of Actor-Critic algorithm. Through validation of the trained policy model on multiple public test data sets and comparison with other advanced DRL methods and scheduling rules, the proposed method reduces the minimum average gap by 3.49%, 5.31% and 4.16%, respectively, compared with the priority rule-based method, and 5.34% compared with the learning-based method. 11.97% and 5.02%, effectively improving the accuracy of DRL to solve the approximate solution of JSSP minimum completion time.

Keywords: job shop scheduling problems; deep reinforcement learning; state characteristics; policy network

Citation: Anjiang Cai, Yangfan Yu and Manman Zhao. "Deep Reinforcement Learning Solves Job-Shop Scheduling Problems." *Instrumentation* 11, no. 1 (March 2024). https://doi.org/10.15878/j.instr.202300165

0 Introduction

The Job shop scheduling problem (JSSP) is a popular NP-hard combinatorial optimization problem that can simulate real-world scheduling problems in manufacturing, planning, and engineering. More and more researchers tend to develop approximate methods to obtain satisfactory solutions, which are mainly divided into: meta-heuristic method, heuristic method based on priority rule and learning method[1,2]. With the increase of production scale and operation complexity, it has become very important to study effective scheduling technology to improve production efficiency, and more scholars have begun to study learning-based methods to solve such problems. WANG et al.[3] adopted the DRL learning strategy model to solve JSSP. In the proposed method, PPO[4] was adopted to find the optimal scheduling strategy. Compared with the traditional scheduling method, the proposed method could realize adaptive and real-time production scheduling, and the expression of state features may not be perfect, resulting in multiple optimal solutions for the candidate action set. ZHANG et al.[5] combined graph neural network (GNN)[6] and PPO to solve JSSP by learning the PDR of the task A policy model of the same size can solve problems of more size types through model generalization. Due to the sparse reward problem, the agent needs to explore a large number of state Spaces to find the state features that obtain rewards, and it is easy to fall into local optimality, which ultimately limits the accuracy of the policy model

to solve the approximate solution of JSSP. At present, there are few researches on solving sparse reward problem in JSSP by deep reinforcement learning. YUAN et al.[7] proposed a utilization scheduling rule that maximizes the cumulative rewards of all agents through Boltzmann utilization to avoid local optimality caused by sparse rewards. LEI et al.[8] proposed an end-to-end hierarchical reinforcement learning framework for large-scale flexible job shop scheduling problems. They designed a high-level agent for resscheduling and two low-level agents for job assignment and machine selection, and solved the sparse reward problem by means of multilevel controllers, achieving good results. However, the subtask division of hierarchical reinforcement learning requires a lot of knowledge and experience, and needs to design multiple strategies and learning processes, resulting in high algorithm complexity.

The problems in solving JSSP by using deep reinforcement learning are as follows: (1) The construction of state features is not perfect, and the relationship between state features and learning objectives is not established, which makes the learning time of agents longer and reduces the decision-making efficiency of agents; (2) The deficiency of policy model construction affects the ability of policy network to explore the optimal solution, limits the decision-making ability of agents, and affects the generalization ability of policy model and the ability to solve JSSP; (3) Lack of research on the sparse reward problem makes the agent need to explore a large number of potentially invalid states to find those states that can obtain rewards, and the agent is easy to fall into the local optimal solution or completely unable to learn.

In order to better apply the learning method to JSSP, a deep reinforcement learning framework is proposed to solve the sparse reward problem and improve the accuracy of solving the approximate solution of JSSP. In this paper, GIN++ is adopted to embed the node coding. By introducing an adjacency matrix regularization mechanism, the network can map the model to a specific space. The performance and generalization ability of the model are improved. The bidirectional scheduling method is used to design 6 state features to improve the state feature representation, including 4 state features to distinguish the optimal action and 2 state features related to the learning goal. Taking the quality difference between the predicted completion time and the theoretical completion time as the reward design, the state feature representation proposed in this paper not only satisfies the distinction of state features, but also realizes the connection with the learning goal, which is conducive to the decision-making of the policy network, but also compensates the agent's ability to explore the state space to obtain rewards, and avoids the agent falling into the local optimal or unable to learn problem. In the face of complex combinatory optimization problems, Actor-Critic algorithm will be affected by local optimal solutions and gradient instability, etc. In each iteration of Actor-Critic algorithm improved by IG algorithm, IG can use greedy algorithm to construct new policy parameters, that is, the optimal action based on the current strategy. Doing so helps the Actor-Critic algorithm explore the search space more efficiently and find better strategies. Through continuous iterative improvement, the search ability of Actor-Critic algorithm is improved to obtain better strategy performance. A feasible method for job shop production scheduling is proposed.

1 Problem Description

Standard JSSP can be described as: Scheduling n job $J_i=\{J_1, J_2, ..., J_n\}$ to be processed on K machine $M_k=\{M_1, M_2, ..., M_k\}$. Each job has an operation, each job has m operations $O_{i,j}=\{O_{i,1}, O_{i,2},...,O_{i,m}\}$ processed in a certain order, $i$represents the job index of the operation, j represents the machine index of the operation. The processing time of each operation on the machine is $T_{i,j,k}$, the start time of operation processing is $t_{i,j}^{s}$, and the end time of operation processing is $t_{i,j}^{e}$. The optimization objective of this paper is to minimize the completion time, and the assumptions and constraints are as follows:

1.1 Basic Assumptions

(1) Capacity constraint: Each machine can handle at most one operation at the same time.

(2) Non-preemptive: Each machine can only process one operation before processing the next operation.

(3) The machine will not fail when processing the job operation.

(4) In the process of job processing, the preparation and transportation time between operations is ignored.

1.2 Mathematical Model

1.2.1 Objective function

Optimization goal: minimum completion time.

$$T=\min \max_{\substack{1\leqslant i\leqslant n\\ 1\leqslant j\leqslant m}} \{t_{i,j}^{e}\} \tag{1}$$

1.2.2 Decision variables

$$x_{i,j,k}=\begin{cases}1, O_{i,j} \text{is processed on } \mathrm{M_k}\\ 0, \text{others}\end{cases} \tag{2}$$

$$Y_{g,h,i,j}=\begin{cases}-1, O_{g,h} \text{ is processed before } O_{i,j}\\ 1, O_{g,h} \text{ is processed after } O_{i,j}\end{cases} \tag{3}$$

1.2.3 Constraints

1) Each operation can only be processed on one machine.

$$\sum\nolimits_{k\in K} X_{i,j,k}=1, \forall i, j, k \tag{4}$$

2) The start time of each operation on the machine is greater than or equal to zero.

$$(t_{i,j}^{e}-T_{i,j,k})\ X_{i,j,k}\geqslant 0,\ \forall i,\ j,\ k \tag{5}$$

3) Constraints on the sequence of operations.

$$(t_{i,j+1}^{e}-T_{i,j,k}-t_{i,j}^{e})\ X_{i,j,k}\geqslant 0,\ \forall i,\ j,\ k \tag{6}$$

4) Only one operation can be performed on each

machine.

$$\frac{1}{2}\left\{\begin{array}{l}(t^e_{g,h}-t^e_{i,j}-T_{g,h,k})X_{g,h,k}X_{i.j.k}Y_{g,h,i,j}(Y_{g,h,i,j}+1)\\+(t^e_{i,j}-t^e_{g,h}-T_{g,h,k})X_{g,h,k}X_{i.j.k}Y_{g,h,i,j}(Y_{g,h,i,j}-1)\end{array}\right\}\geqslant 0, \quad (7)$$

$\forall(i, j), (g, h), k$

5) The processing time of the operation on the machine is greater than zero.

$$T_{i,j,k}=t^e_{i,j}-t^s_{i,j} \quad (8)$$

$$0<T_{i,j,k} \quad (9)$$

6) The initial processing time of the operation on the machine, where $t^e_{g,h,k}$ represents the completion time of the previous operation on the same machine.

$$t^s_{i,j}=\max\{t^e_{i,j-1},t^e_{g,h}\} \quad (10)$$

7) Machine processing completion time.

$$T^e_k=t^s_{i,j}+T_{i,j,k} \quad (11)$$

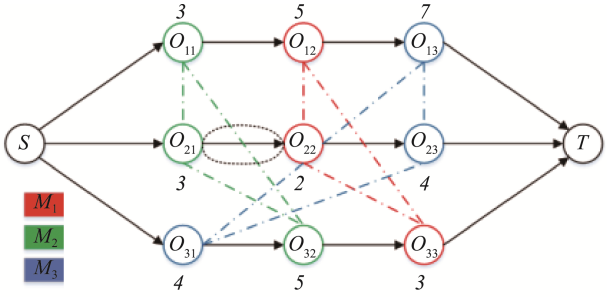
8) Machine processing operation time.

$$T_k=\sum_{\substack{i=1\\j=1}}^{\substack{i=n\\j=m}}T_{i,j,k},\forall i,j,k \quad (12)$$

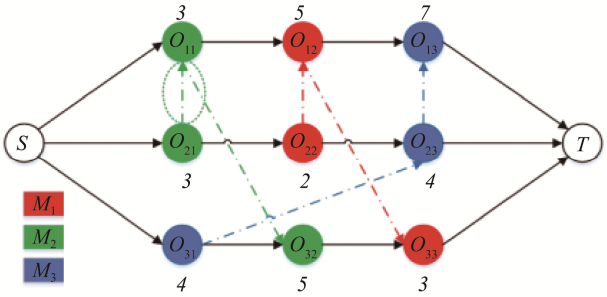
9) Job processing time.

$$T_i=\sum_{\substack{j=1\\k=1}}^{\substack{j=m\\k=K}}T_{i,j,k},\forall\, i,j,k \quad (13)$$

JSSP is expressed as disjunctive graph[9] and modeled as topological structure of state and action, as shown in Fig.1(a) represents a 3×3 JSSP instance. The black arrows are connected arcs, indicating the priority relationship between adjacent operations of the same job. Dotted lines of the same color indicate operations processed on the same machine. Fig.1(b) is a complete solution that determines the disjunction arc direction for each machine. The set $O=\{O_{i,j}\mid\forall i,j\}\cup\{S,T\}$ represents each operation node in the disjunctive graph, and S and T represent the start and end of job processing respectively, which are all virtual values.



(a) JSSP Instance



(b) JSSP complete solution

Fig.1 Disjunctive diagram representation

2 Construct the DRL Framework for Solving JSSP

The JSSP is described by disjunction graph, the state features are embedded by GIN++ network, and the optimal strategy model is trained by PPO algorithm under Actor-Critic framework. The overall deep reinforcement learning framework for solving JSSP is shown in Fig.2.

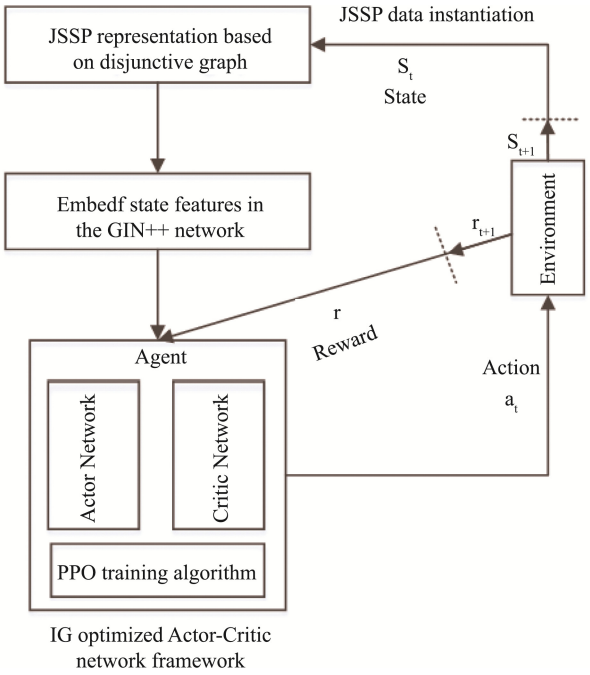


Fig.2 General framework for solving JSSP

2.1 MDP Definition for the Job Shop

The JSSP problem is modeled as a Markov decision model (MDP). The standard MDP consists of (S, a, R, γ, P) five parts, which represent state, action, reward, discount factor and state transition probability in turn.

2.1.1 Status

State feature design should not only distinguish states, but also relate to learning goals. The selection of state features should take into account their impact on the realization of the goal. The selection of state features related to the goal can provide a more informative and effective representation, so as to help the agent better understand the environment, make decisions and optimize strategies. According to the state of the disjunctive graph in the solution, 6 state features are designed:

1) $I_{i,j}(s_t)$: Denotes the scheduling status of node $O_{i,j}$ in the disjunctive graph when state s_t time is obtained, which is 0 when the node is not scheduled and 1 when it is scheduled.

2) $t^{e}_{i,j}(s_t)$: Represents the estimated completion time of $O_{i,j}$.

$$t^{e}_{i,j}(s_t) = t^{s}_{i,j}(s_t) + T_{i,j} \tag{14}$$

Where $t^{s}_{i,j}(s_t)$ represents the estimated start time of $O_{i,j}$ at state s_t, recursively, and if $O_{i,j}$ represents the first operation of the job, it represents the time for the machine to complete the other operations of the job.

3) $t_{M_k}(s_t)$: When the state is s_t, the completion time of the remaining operation on the machine after the machine M_k completes the operation $O_{i,j}$ is calculated as follows:

$$t_{M_k}(s_t) = \sum_{\substack{i=n \\ j=m \\ i,j}} T_{i,j,k}(s_t), \forall k \tag{15}$$

Where i and $j$indicate the indexes of the scheduled operations. If $O_{i,j}$ is from the set of unscheduled operations, it represents the maximum estimated completion time of all unscheduled operations on the same machine, calculated using two-way scheduling. If it is not from the candidate operation set, the value is 0.

4) $t_{J_i}(s_t)$: Represents the completion time of the remaining operations in J_i after the completion of $O_{i,j}$ in state s_t. The formula is as follows:

$$t_{J_i}(s_t) = \sum_{\substack{j=m \\ k=K \\ j,k}} T_{i,j,k}(s_t), \forall i \tag{16}$$

j and k indicate the indexes of the scheduled operations. If $O_{i,j}$ comes from the set of unscheduled operations, it represents the total completion time of all unscheduled operations in the same job, which is calculated using two-way scheduling.If it is not from the candidate operation set, the value is 0.

5) $T_{M_k}(s_t)$: Represents the maximum value of the actual completion time of M_k exceeding the estimated completion time when the state is s_t. The calculation formula is as follows:

$$T_{M_k}(s_t) = \max\{t^{e}_{k}(s_t) + t_{M_k}(s_t) - T_k(s_t)\} \tag{16}$$

When $O_{i,j}$ is completed on the machine, the completion time of the operation is the same as the completion time of the machine, i.e. $t^{e}_{k}(s_t) = t^{e}_{i,j}(s_t)$, where $t^{e}_{k}(s_t) + t_{M_k}(s_t)$ represents the estimated completion time of the machine M_k at state s_t, and $T_k(s_t)$ represents the continuous completion time of all operations on the machine, i.e., the sum of all operation times on the machine M_k.

6) $T_{J_i}(s_t)$: Represents the maximum value of the actual completion time of J_i exceeding the estimated completion time when the state s_t is applied. The formula is as follows:

$$T_{J_i}(s_t) = \max\{t^{e}_{i,j}(s_t) + t_{J_i}(s_t) - T_i(s_t)\} \tag{17}$$

Where $t^{e}_{i,j}(s_t) + t_{J_i}(s_t)$ represents the estimated completion time of job J_i at state s_t, and the last item $T_{J_i}(s_t)$ represents the continuous completion time of job J_i, which is the sum of all operation times in job J_i.

According to the definition of the state features of disjunctive graph nodes, the feature vector of disjunctive graph nodes is constructed, and the state feature information of unscheduled nodes is aggregated by bidirectional scheduling. As shown in Fig.3, the scheduling along the direction of node S to T is forward scheduling, and the scheduling along the direction of node T to S is reverse scheduling. In Fig.3, when the state is s_1, the scheduled node is $O_{3,1}$ and the candidate scheduling set is $\{O_{1,1}, O_{1,2}, O_{1,3}\}$. As shown in Fig.4(a), the state feature 5 of $O_{1,1}$ when the state is s_1 is M_1, M_2, M_3 and the final completion time exceeds the maximum value of the

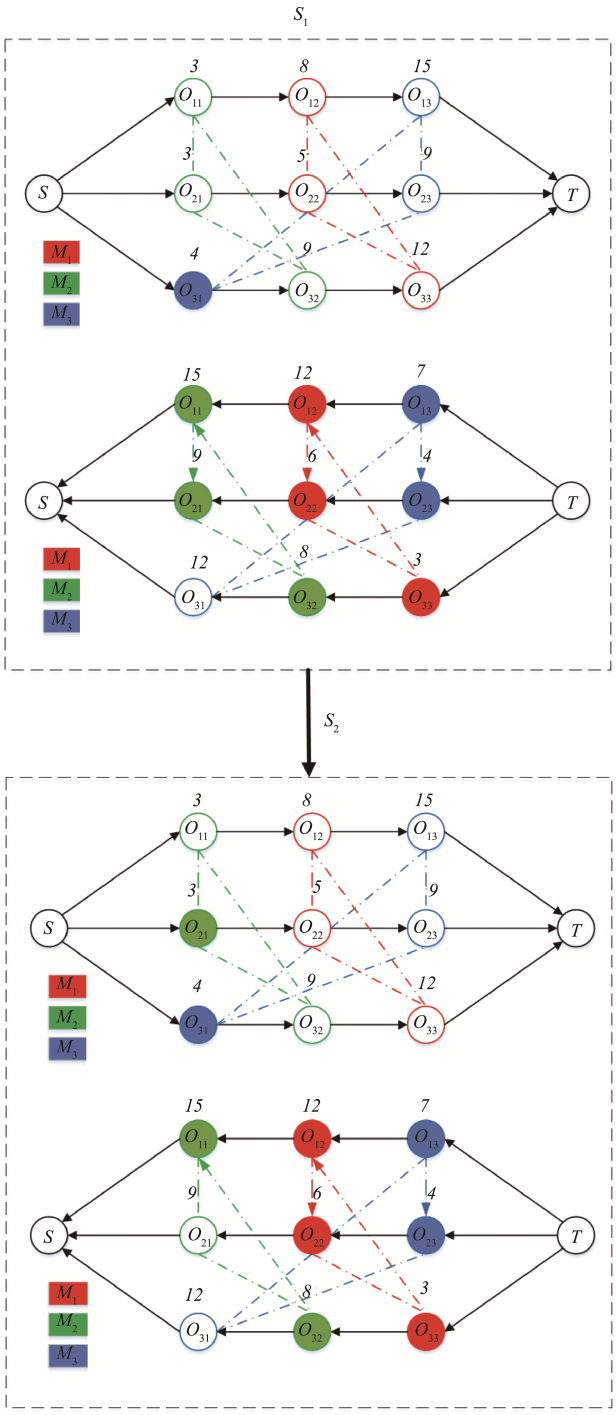


Fig.3 State transition process

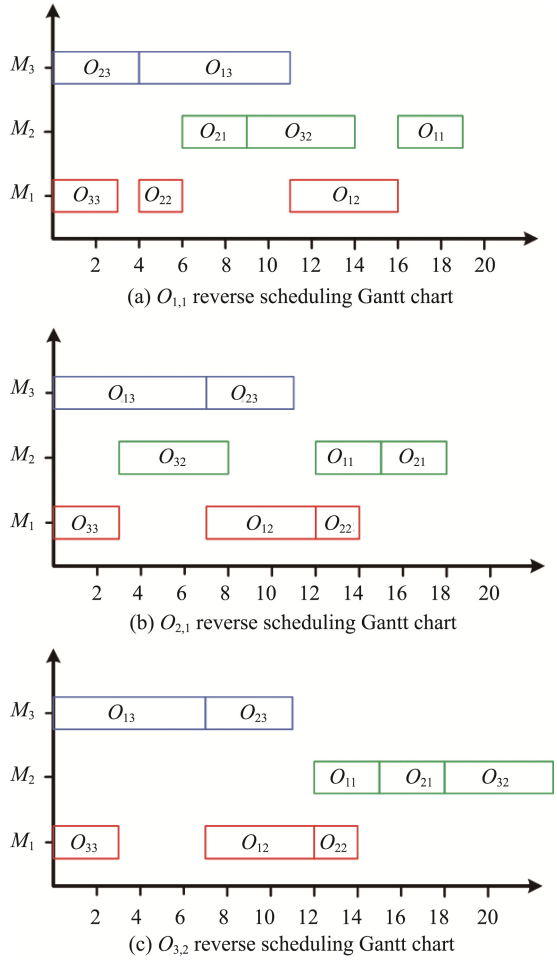


Fig.4 Reverse scheduling Gantt chart of operations at state s_1

continuous processing time, that is, max{16–5–2–3, 19–3–5–3, 11–7–4}=8. State feature 6 is J_1, J_2, J_3 The final completion time exceeds the maximum value of the continuous processing time part, that is, max{19–4–15, 19–9, 14–8}=6, so the state feature vector of $O_{1,1}$ at state s_1 is $O_{1,1}$={0, 3, 12, 9, 8, 6}. Similarly, as shown in Fig.4(b), when $O_{1,2}$ is in state s_1, state feature 5 is max{14–2–5–3, 18–3–3–5, 11–4–7}=7, state feature 6 is max{15–15, 18–7–9, 8–8}=2, and the state feature vector of $O_{2,1}$is $O_{2,1}$=[0, 3, 12, 9, 8, 6] when it is in state s_1. Similarly, as shown in Fig.4(c), the state feature 5 of $O_{3,2}$ in state s_1 is max{14–2–5–3, 23–5–3–3, 11–4–7}=12, the state feature 6 is max{15–15, 18–7–9, 23–8}=15, and the state feature vector of $O_{3,2}$ in state s_1 is $O_{3,2}$=[0, 5, 3, 3, 12, 15]. In state s_1, where $O_{1,1}$ and $O_{2,1}$ state features 1, 2 are the same, it is impossible to distinguish the optimal solution, and the design of state features 3, 4 complements the deficiency of state feature representation.However, state features 1 to 4 only satisfy the distinction state, and state 3 of $O_{1,1}$ and $O_{2,1}$ is different, which has no obvious connection with the minimum completion time goal of DRL, leading to the sparse reward problem. A smaller value of state feature 5 indicates a shorter waiting time for the machine to process, and a smaller value of state feature 6 indicates a shorter waiting time for the job operation to be processed in the buffer. Therefore, the smaller the value of status feature 5 and 6, the earlier the completion time of the job, that is, the higher the priority of the job operation. Therefore, operation $O_{2,1}$ is selected when the status s_2 is used. State features 5 and 6 provide more efficient states and obtain more cumulative rewards, making up for the sparse reward problem caused by insufficient correlation between DRL state features design and learning objectives.

2.1.2 Movements

The number of candidate job sets is $|J|$, and each job provides at most one operation during the decision step to construct the action space A_t, and the size of the action space A_t does not exceed $|J|$ and becomes smaller with the decision process.

2.1.3 Status Change

The state transition depends on the environment, and the generation of a new state is equivalent to updating the disjunctive arc direction in the disjunctive graph. When the priority of a job operation in a certain state is determined, the operation to be scheduled is scheduled on the machine according to the earliest available time of the machine, and the disjuncting arc direction on the machine is updated. As shown in Fig.3, after the agent completes an action, the environment simulator updates the state from s_t to s_{t+1} and generates the corresponding disjunction diagram in the new state.

2.1.4 Reward

With the goal of minimizing the completion time, the processing order of each job operation on the machine is gradually arranged according to the priority of the job operation. The reward function $R(s_t, a_t)$ is the difference between the pairs of two states s_t and s_{t+1}. Formula (18), where $\max t^e_{i,j}(s_t) = t^e_{i,j-1} + T_{i,j,k}$ calculates the completion time of J_i at state s_t recursively. The discount factor is γ=1, the cumulative reward is R, and the formula is (19), where $T_i(s_0)$ is the constant representing the theoretical completion time of the job, and $s_{|O|-2}$ represents the set of node states except for {S, T} nodes. The calculation of R is related to the representation of state feature 6, and the reward in this paper is calculated using state feature 6.

$$R(s_t, a_t) = \max\{t^e_{i,j}(s_t) + t_{J_i}(s_t)\} - \max\{t^e_{i,j}(s_{t+1}) + t_{J_i}(s_{t+1})\} \tag{18}$$

$$R = \sum_{t=0}^{|O|-3} R(s_t, a_t) = \max T_i(s_0) - \max t^e_{i,j}(s_{|O|-2}) \tag{19}$$

The disjunctive graph in the constructed MDP presents the priority constraints of the job operations, the processing sequence of the job operations on the corresponding machine, and the time of the machine to process each pending job operation, which reflects the scheduling state of the JSSP, the disjunctive graph structure, and the job data. Therefore, it is very important to extract all the state information contained in the disjunction graph and use it to complete effective scheduling.

2.2 Disjunctive Graph Coding

Therefore, an extended variant of graph isomorphic

network (GIN)[10], GIN++, is used to encode disjoint graphs, which can map models into specific Spaces by introducing an adjacency matrix regularization mechanism, thereby improving model performance and generalization ability. According to the definition of disjunctive graph, each node of disjunctive graph is encoded by several layers of GIN++ to achieve the embedding of state features. Compared with GIN, GIN++ superimposes the new feature vector of the current node with the previous feature vector to obtain the final feature representation of the current node GIN++ does l iterative updates on each node, computes p-dimensional embeddings, and the update at iteration l is expressed as:

$$h_v^{(l)} = MLP_{\theta_l}^{(l)}((1+\varepsilon^l)h_v^{(l-1)} + \sum_{a\in N(v)} h_u^{(l-1)}) + h_v^{l+1} \tag{20}$$

Where $MLP_{\theta_l}^{(l)}$ is a multi-layer perceptron (MLP) with parameter θ_l for iterating l and then normalizing, ε is a learnable arbitrary parameter, $N(v)$ is a neighborhood of v, h_v^l is a representation of node v when iterating l, and h_v^0 represents the original feature of the input. After l iterations of updating, a global representation of the entire graph can be obtained using the pooling function L, which takes the embeddings of all nodes as input and outputs the p-dimensional vector $h_G \in R^p$, with average pooling.

2.3 Optimization of Policy Network

MLP is used to construct a deep neural network for state representation and strategy learning. The strategy network actor-critic in this paper consists of two parts, in which the Actor network is used to output the probability distribution of the action; The Critic network is used to output estimates of action values. In order to improve the exploration ability and learning efficiency of the strategy network, Iterated Greedy Algorithm (IG)[11,12] is adopted to optimize the Actor-Critic algorithm. The specific steps are as follows:

1) Initialize the policy: Use the IG algorithm to select the random policy from the policy space as the initial policy.

2) Actor-Critic policy update: Initial policy parameters are used to update the Actor-Critic algorithm through the gradient rise method to optimize the policy network and improve the current policy.

3) IG generates new policy parameters: IG algorithm is used to calculate information entropy, select actions with maximum information gain for expansion, and find the optimal policy parameters.

4) Evaluate new policy parameters: Test the newly generated policy parameters in the environment and calculate its performance in the environment.

5) Update the policy and value function: According to the evaluation results, use the Actor-Critic algorithm to update the policy and value function. The parameters of policy network and value function network are optimized to approximate better policy and value function.

6) Iterative update: Repeat steps 3-5 to continuously improve and search for better strategies through multiple iterations. In each iteration, IG generates new policy parameters, which are evaluated and updated by Actor-Critic.

3 Experimental Verification

According to the number of jobs and machines in the training set data, policy models of different sizes are trained, including 10×10, 10×15, 10×20, 15×10, 15×15, 20×10, 20×15, 20×20, 30×10, 30×10, 30×15, 30×20. DUM example and LA example were used to evaluate the policy model.

3.1 Experimental Data and Evaluation Indicators

3.1.1 Data Set

Experimental data includes training, validation, and test datasets. The training data set and the experimental data set are combined, and the common benchmark instance is the test data set. The operation processing time range is {1, 90} or {1, 199}, and the number of random seeds is 200.

3.1.2 Evaluation Indicators

In terms of performance evaluation, the relative deviation percentage (RPD)[13] is used as a metric, also known as the optimality gap. Literature [5], [14] and[15] also adopted RPD index as the main evaluation index:

$$RPD = \frac{T_{\max} - T_{\max}^*}{T_{\max}^*} \times 100\% \tag{21}$$

Where $T_{\max}^*$ represents the optimal solution or approximate optimal solution obtained by an exact method, and $T_{\max}$ represents the optimal solution obtained by an approximate method.

3.2 Parameter Settings

Program in Windows 11 64bit computer (CPU: AMD Ryzen 5 5600 6-Core Processor 3.50 GHz,GPU: NVIDIA GTX 1080Ti 16GB memory), based on Python3.10.9, Pytorch2.0.1 environment.

The learning rate l_r of PPO training algorithm is 2×10^{-5}, the clipping parameter ε is 0.2, the discount factor γ is 1, the critical loss coefficient C_v is 1, the strategy loss coefficient C_p is 2, and the entropy loss coefficient C_e is 0.01.

3.3 Validity Verification

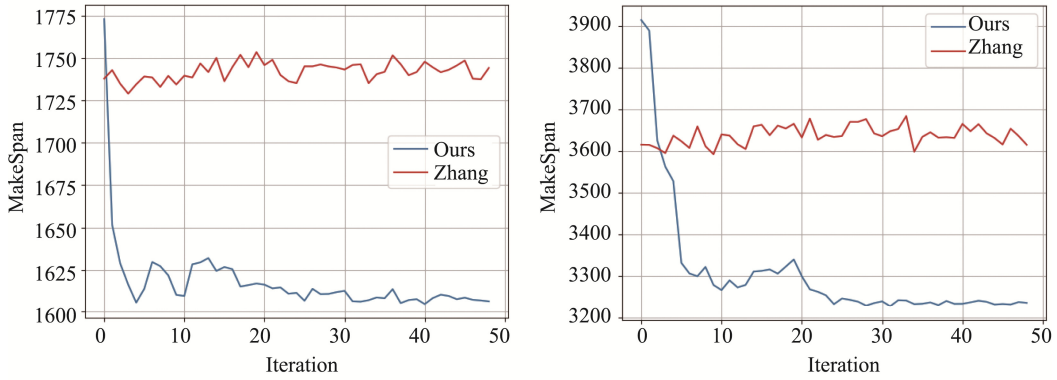
3.3.1 Impact of Introducing IG Algorithm

The improved policy model was compared with the policy model in literature [5] to verify the effectiveness of the introduced IG algorithm. As shown in Fig.5(a) and (b), the training results of the two policy models on the 20×15, {1, 99} and 20×15, {1, 199} data sets are shown. After the introduction of IG algorithm, the learning effect of the strategy model is not good at the initial training stage, but its exploration ability is enhanced and it rapidly converges with the increase of training times. Moreover, the learning

effect reaches the best when the iteration is about 5 times, and the learning effect reaches the best and tends to be stable when the iteration is about 25 times. When facing complex combinatorial optimization problems, the Actor-Critic algorithm adopted in [5] will be affected by problems such as local optimal solution and gradient instability. In this paper, IG improves Actor-Critic algorithm. In each iteration, IG can use greedy algorithm to construct new policy parameters, that is, the optimal action based on the current strategy. This will help Actor-Critic algorithm to explore more efficiently in the search space and find better strategies. Through continuous iterative improvement, the search ability of Actor-Critic algorithm is improved to obtain better strategy performance. At the same time, the learning speed has been significantly improved, and the problem of poor solving effect of policy network caused by sparse rewards has been improved, and the decision-making ability of policy model has been effectively improved.

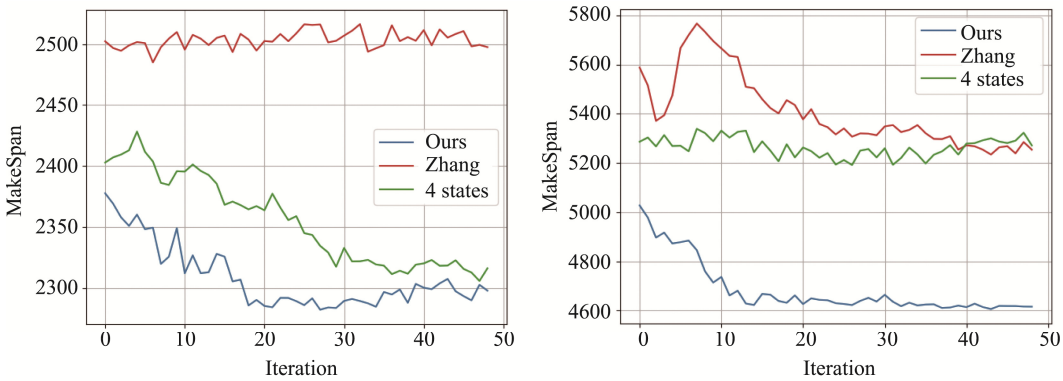
3.3.2 Design the Influence of State Features Related to Learning Objectives

The state feature representation proposed in this paper is compared with the feature representation not related to learning objectives and the state feature representation in literature [5] to verify the validity of the new state feature representation proposed. As shown in Fig.6(a) and (b), when the state feature representation does not design the state feature related to the learning goal, the training effect of the policy model is significantly improved on the 30x20, {1,99} data set, but not on the 30x20, {1,199} data set. Therefore, when the state feature representation does not design the state feature related to the learning goal, the solution effect of the strategy model is not significantly improved, and the improvement effect is not stable. The state feature representation method of [5] only satisfies the distinction of different state features, and does not further design state features related to learning objectives. The state feature representation proposed in this paper not only satisfies the distinction of state features, but also designs two state features related to learning goals on this basis, which makes up for the agent's ability to continue exploring state space to obtain rewards, and avoids the agent falling into local optimal or unable to learn problems. This paper provides a more informative and effective representation by selecting a state feature representation method related to the goal, which helps agents better understand the environment and make decisions and optimize strategies.



(a) 20×15, {1, 99}　　(b) 20×15, {1, 99}

Fig.5 Training effect of optimized strategy model on different data sets
(a)The strategy model is trained on the 20×15, {1, 99} data set
(b)The strategy model is trained on the 20×15, {1, 199} data set



(a) 30×20, {1, 99}　　(b) 30×20, {1, 99}

Fig.6 Validity of the new state feature representation
(a) The effect of solving the new state feature on a 30x20, {1,99} data set
(b) The effect of solving the new state feature on a 30x20, {1,199} data set

3.4 Generalization of Policy Model

Table 1, Table 2 and Table 3 show the generalization ability of policy models of different sizes on TA, DMU and LA test sets under different training data sets and different data set sizes. As shown in Table 1, the 15×15 and 20×15 policy models are suitable for solving medium and small size test datasets (15×15, 20×15, 20×20, 30×15), while the 20×20, 30×15 and 30×20 policy models are more suitable for medium and large size test datasets (30×20, 50×15, 30×15). 50×20, 100×20). As shown in Table 2, the policy model with the size of 20×15 has the strongest solving ability and generalization ability for instances in DMU data sets, while the policy model with the size of 30×20 has stronger solving ability for 40×20 DMU data sets than the policy model with other sizes. As shown in Table 3, policy models with sizes 10×10, 15×15, 20×15, and 20×20 have better solving power and generalization ability. To sum up, the solving ability and generalization ability of policy models with different sizes are affected by the size of the training data set. The data scale of the training data set in this paper is certain, and different policy models cannot achieve the best effect on each test instance. In Table 1, Table 2 and Table 3, 15×15, {1, 99}, 20×15, {1, 99} and 20×15, {1,199} with small data sets have better generalization ability.

Table 1 Average optimality deviation value and average time on TA test set

n×*m*	15×15, {1, 99}		20×15, {1, 99}		20×20, {1, 99}		30×15, {1, 99}		30×20, {1, 99}	
	Gap	Time(s)	Gap	Time(s)	Gap	Time(s)	Gap	Time(s)	Gap	Time(s)
15×15	17.36%	0.83	19.11%	0.81	19.01%	0.68	18.65%	0.70	19.64%	0.69
20×15	23.14%	1.17	20.00%	1.21	20.65%	1.06	24.03%	1.06	20.88%	1.02
20×20	19.56%	1.57	18.93%	1.63	19.03%	1.57	21.56%	1.50	19.25%	1.57
30×15	19.72%	1.94	21.92%	1.78	21.73%	1.81	23.72%	1.83	20.81%	1.71
30×20	23.78%	2.58	23.40%	2.84	24.21%	2.41	23.54%	2.61	22.57%	2.80
50×15	15.05%	3.12	14.89%	3.67	15.13%	3.60	15.04%	3.52	13.82%	3.83
50×20	15.72%	5.06	16.16%	5.18	15.44%	4.73	17.01%	4.85	16.28%	5.49
100×20	7.62%	18.21	7.11%	17.14	7.05%	17.18	6.90%	18.83	6.94%	18.04
Average Gap	17.74%	4.31	17.69%	4.28	17.78%	4.13	18.81%	4.36	17.52%	4.39

Table 2 Average optimality deviation value and average time on DMU test set

n×*m*	20×15, {1, 199}		20×20, {1, 199}		30×15, {1, 199}		30×20, {1, 199}	
	Gap	Time(s)	Gap	Time(s)	Gap	Time(s)	Gap	Time(s)
20×15	**24.40%**	1.13	25.00%	1.02	26.21%	1.08	26.04%	1.10
20×20	**23.44%**	1.47	24.29%	1.54	24.27%	1.53	23.57%	1.47
30×15	**29.63%**	1.75	31.12%	1.71	31.87%	1.70	30.51%	1.70
30×20	**28.45%**	2.46	30.59%	2.91	28.95%	2.60	29.07%	2.44
40×15	**25.30%**	2.54	26.95%	2.50	26.55%	2.65	25.98%	2.59
40×20	29.24%	3.56	30.66%	3.70	30.63%	3.83	**28.70%**	3.65
50×15	**24.88%**	3.53	27.60%	3.50	25.92%	3.47	25.30%	3.40
50×20	**27.32%**	5.35	30.93%	5.25	30.07%	5.91	27.98%	5.08
Average Gap	**26.58%**	2.72	28.39%	2.76	28.06%	2.84	27.14	**2.67**

Table 3 Average optimality deviation value and average time on LA test set

n×*m*	10×10, {1, 99}		15×15, {1, 99}		20×15, {1, 99}		20×20, {1, 99}		30×15, {1, 99}		30×20, {1, 99}	
	Gap	Time(s)	Gap	Time(s)	Gap	Time(s)	Gap	Time(s)	Gap	Time(s)	Gap	Time(s)
10×5	12.77%	0.18	14.54%	0.19	12.55%	0.19	**11.26%**	0.18	21.30%	0.19	12.00%	0.20
15×5	3.70%	0.25	3.88%	0.26	**1.97%**	0.26	3.03%	0.26	13.38%	0.28	2.19%	0.27
20×5	**2.99%**	0.35	4.36%	0.38	3.96%	0.38	3.96%	0.34	5.17%	0.38	3.55%	0.38
10×10	12.95%	0.34	16.88%	0.33	**10.89%**	0.34	13.10%	0.35	19.92%	0.36	12.87%	0.36
15×10	13.94%	0.52	17.15%	0.55	13.31%	0.52	**13.21%**	0.57	18.23%	0.57	14.11%	0.54
20×10	18.56%	0.74	**14.86%**	0.72	16.17%	0.83	16.91%	0.84	19.16%	0.82	16.85%	0.84
30×10	**3.39%**	1.44	4.25%	1.51	5.15%	1.55	4.62%	1.60	7.13%	1.59	3.73%	1.53
15×15	15.05%	0.89	**14.38%**	0.92	17.39%	0.85	15.53%	0.88	18.59%	0.93	16.25%	0.90
Average Gap	10.42%	**0.58**	11.29%	0.60	**10.17%**	0.61	10.20%	0.62	15.36%	0.64	10.19%	0.62

3.5 Comparison of Approximate Solution Accuracy

The baseline comparisons were SPT, FIFO, MWKR, [5], [14] and [16]. [5] GNN is used to embed the node code in the disjunction graph in fixed dimension, predict the state representation and reward calculation of the minimum completion time, and use Actor network to explore the state space and generate strategies and output actions.[14] GNN was used to learn node features embedded in JSSP spatial structure, and scheduling policies were derived to map embedded node features to scheduling actions. Node state changes were used as state features and reward calculations, and Actor networks were used to explore and generate strategies for state space and output actions.[16] A DRL framework called DGERD Transformer is proposed, which integrates disjunctive graph embeddings and attention mechanisms in DNN to solve JSSP based on attention mechanism and disjunctive graph embeddings. Priority constraints of job operations are used as state representations. Minimize the inverse of the maximum completion time as a reward setting.

As shown in Table 4, the average solving gap of the strategy model trained by the proposed method is superior to other methods, increasing by 3.49% compared with the minimum average gap based on priority rules and 5.34% compared with the minimum average gap based on learning methods. As shown in Fig.7, among the 80 groups of data in TA test data set, the results of this paper are superior to those of other control groups in 69 groups, and the optimal results account for 86.25%. Most current

Table 4 Comparison of TA test set results

n×*m*	SPT	FIFO	MWKR	[5]	[14]	[16]	Ours
15×15	25.88%	23.11%	21.19%	25.96%	20.13%	35.38%	**17.36%**
20×15	32.81%	30.02%	22.84%	30.03%	24.95%	32.09%	**20.00%**
20×20	27.75%	27.67%	23.98%	31.61%	29.25%	28.33%	**18.93%**
30×15	35.27%	30.22%	23.96%	33.00%	24.70%	36.43%	**19.72%**
30×20	34.44%	30.90%	25.68%	33.62%	32.00%	34.67%	**22.57%**
50×15	24.10%	20.11%	17.79%	22.38%	15.92%	31.86%	**13.82%**
50×20	25.53%	23.18%	18.41%	26.51%	21.29%	28.04%	**15.44%**
100×20	14.40%	12.75%	8.81%	13.61%	9.24%	17.89%	**6.90%**
Average Gap	27.52%	24.74%	20.33%	27.09%	22.18%	30.59%	**16.84%**

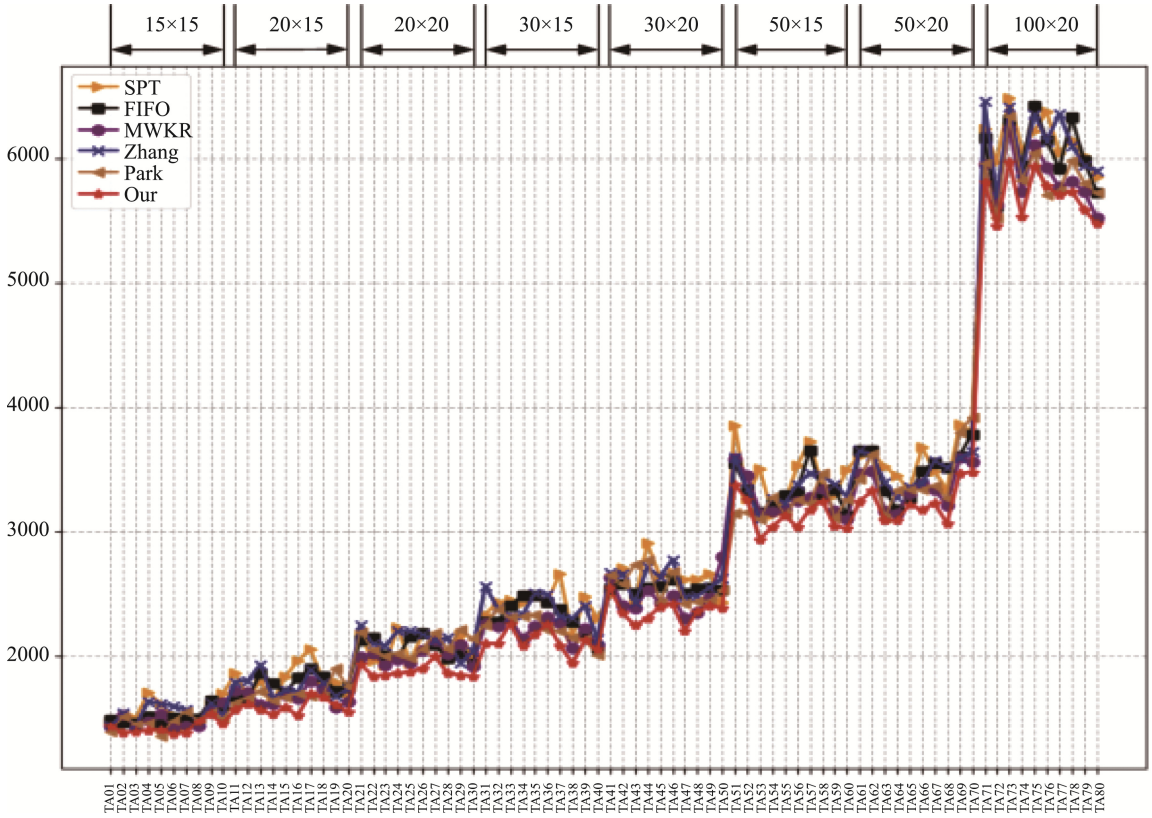


Fig.7 Comparison of TA instance test sets

learning-based methods use TA test sets to evaluate the performance of their strategy models, which also proves the effectiveness of the proposed methods. As shown in Table 5, the average solving gap of the strategy model trained by the proposed method is superior to other methods, which is 5.31% higher than the minimum average gap based on priority rules and 11.97% higher than the minimum average gap based on learning methods. As shown in Fig.8, among the 80 groups of data in the DMU test data set, our results were superior to those of other control groups in 74 groups, and the optimal results accounted for 92.50%. As shown in Table 6, the strategy model trained by the proposed method is better than other methods in solving the average gap, which is 4.16% higher than the minimum average gap based on priority rules and 5.02% higher than the minimum average gap based on learning methods. Literature [14] has better generalization ability on 15x5 and 20x5. As shown in Fig.9, among the 40 groups of data in the LA test data set, the results of this paper are superior to those of other control groups in 29 groups, and the optimal results account for 72.50%. In general, the average RPD value of the solution results of DMU data set is 26.51%, which is the largest compared with the average RPD value of the solution results of TA and LA data sets, possibly because the processing time range of each operation in DMU data set is larger.

Table 5 Comparison of DMU test set results

m×*n*	SPT	FIFO	MWKR	[5]	Ours
20×15	64.13%	37.18%	30.49%	38.95%	**24.40%**
20×20	64.56%	32.43%	26.35%	37.74%	**23.44%**
30×15	62.56%	39.29%	34.79%	41.86%	**29.63%**
30×20	65.91%	36.57%	32.18%	39.48%	**28.45%**
40×15	55.87%	35.08%	31.16%	35.38%	**25.30%**
40×20	63.00%	39.72%	33.24%	39.38%	**28.70%**
50×15	50.37%	34.74%	31.04%	36.20%	**24.88%**
50×20	62.19%	41.38%	35.34%	38.86%	**27.32%**
Average Gap	61.07%	37.04%	31.82%	38.48%	**26.51%**

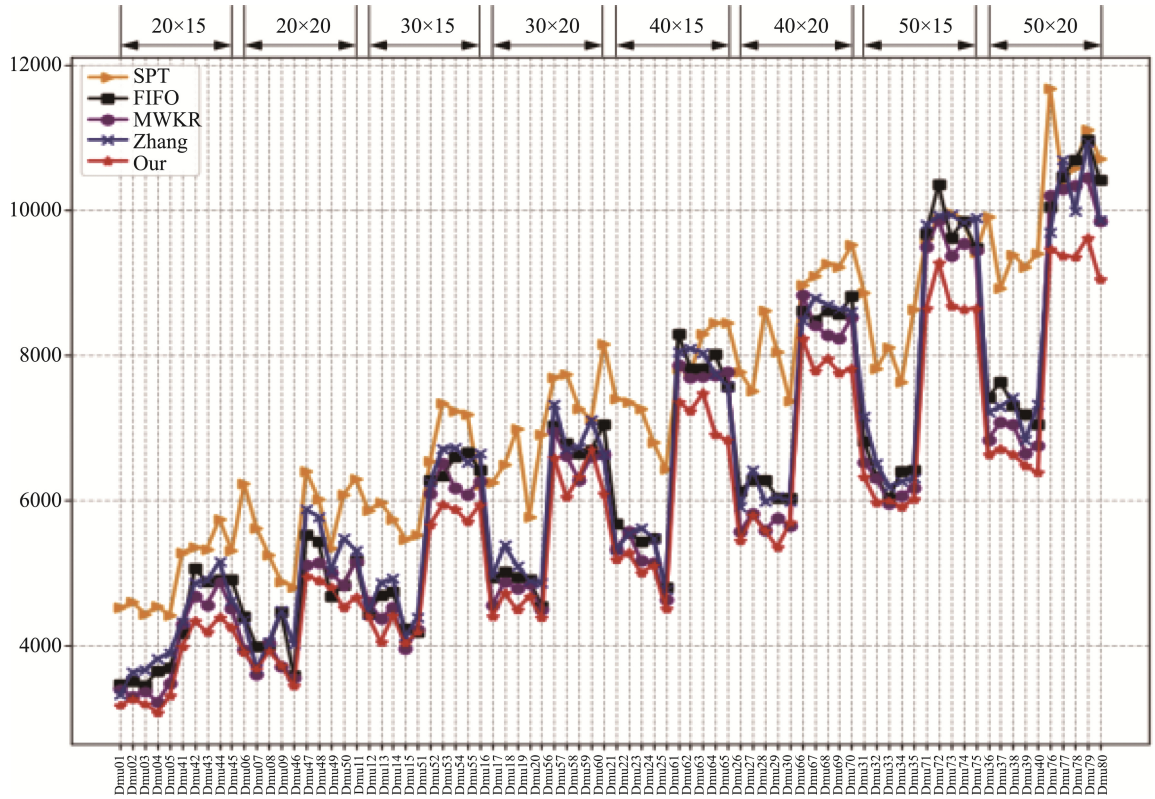


Fig.8 Comparison of DMU test data sets

Compared with the GNN node coding embeddings adopted in [5] and [14], this network can map the model to a specific space by introducing an adjacency matrix regularization mechanism, thus improving the performance and generalization ability of the model. Taking the quality difference between the predicted completion time and the theoretical completion time as the state representation and reward design, the state feature representation methods [5], [14] and [16] can only satisfy the distinction of different state features. On this basis, the state feature representation proposed in this paper can realize the connection with the learning goal and is conducive to the decision-making of the policy network. At the same time, it makes up for the agent's ability to explore the state space and get rewards, so as to avoid the agent falling into local optimal or unable to learn. In the face of complex combinatory optimization problems, Actor-Critic algorithm will be affected by local optimal solutions and gradient instability, etc. In each iteration of Actor-Critic algorithm improved by IG algorithm, IG can use greedy algorithm to construct new policy parameters, that is, the optimal action based on the current strategy. Doing so helps the Actor-Critic algorithm explore the search space more efficiently and find better strategies. Through continuous iterative improvement, the search ability of Actor-Critic algorithm is improved to obtain better strategy performance. But[16] it takes less time to solve large-scale problems.

Table 6 Comparison of LA test set results

$n\times m$	SPT	FIFO	MWKR	[14]	Ours
10×5	14.80%	17.95%	16.49%	16.06%	**11.26%**
15×5	14.86%	9.57%	5.79%	**1.08%**	1.97%
20×5	13.71%	7.96%	4.88%	**2.12%**	2.99%
10×10	15.67%	25.32%	14.82%	17.05%	**10.89%**
15×10	28.68%	29.40%	19.81%	21.96%	**13.21%**
20×10	33.43%	24.45%	20.88%	27.25%	**14.86%**
30×10	13.88%	11.16%	7.79%	6.27%	**3.39%**
15×15	24.58%	25.29%	15.83%	21.40%	**14.38%**
Average Gap	19.95%	18.88%	13.28%	14.14%	**9.12%**

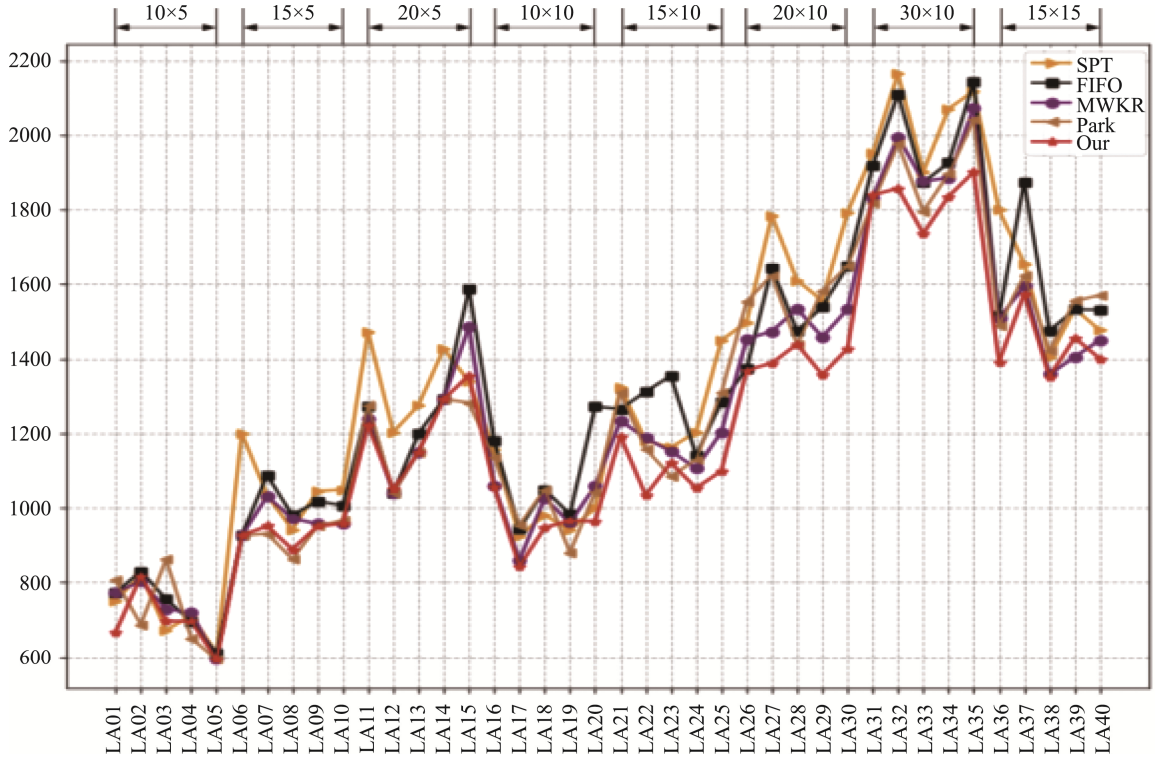


Fig.9 Comparison of LA test data sets

4 Conclusion

① Six state features are designed, including four state features that distinguish the optimal action in the candidate set, and two state features that are related to the learning goal to improve the sparse reward problem, making it easier for the agent to capture effective state information, better understand the environment, and improve the learning effect and stability of the policy network.

② By introducing IG algorithm to optimize Actor-Critic algorithm, IG algorithm is used to calculate information entropy, select actions with maximum information gain for expansion, and find the optimal policy parameters to optimize policy parameters, which effectively promotes the exploration ability of policy networks and improves the solution of sparse reward problem of policy networks.

③ The generalization ability of the policy model is verified on the TA, DMU and LA test sets, where the results of the optimal approximate solution account for 86.25%, 92.50% and 72.50%, respectively, which proves the effectiveness of the improved policy model to solve the minimum completion time of job-shop scheduling.

④ The ability of the strategy model to solve the approximate solution of the minimum completion time was verified on the TA, DMU and LA test sets. The results showed that the minimum average gap was increased by 3.49%, 5.31% and 4.16%, respectively, and the decibel was increased by 5.34%, 11.97% and 5.02%, respectively, compared with the minimum average gap based on the priority rule method. The accuracy of the approximate solution of the minimum completion time of job-shop scheduling is effectively improved.

Author Contributions:

Yangfan YU: Writing – original draft, Software, Conceptualization, Validation, Visualization.Anjiang CAI: Supervision, Funding acquisition, Writing – review & editing. Manman ZHAO: Writing – review & editing, Formal analysis.

Funding Information:

Shaanxi Provincial Key Research and Development Project (2023YBGY095) and Shaanxi Provincial Qin Chuangyuan "Scientist + Engineer" project (2023KXJ247) Fund support.

Data Availability:

The authors declare that the main data supporting the findings of this study are available within the paper and its Supplementary Information files.

Conflict of Interest:

The authors declare no competing interests.

Dates:

Received 17 July 2023; Accepted 17 December 2023; Published online 31 March 2024

References

[1] Chupeng Su, Cong Zhang, Dan Xia, et al.Evolution strategies-based optimized graph reinforcement learning for solving dynamic job shop scheduling problem[J]. Applied Soft Computing, 2023, 145. DOI:10.1016/j.asoc.2023. 110596.

[2] Kun Lei, Peng Guo, Wenchao Zhao, et al.A multi-action deep reinforcement learning framework for flexible Job-shop scheduling problem[J]. Expert Systems with Applications, 2022, 205.DOI:10.1016/j.eswa.2022.117796.

[3] Wang L, Hu X, Wang Y, et al. Dynamic Job-shop Scheduling in Smart Manufacturing using Deep Reinforcement Learning[J]. Computer Networks, 2021, 190(2):107969. DOI:10.1016/j.comnet.2021.107969.

[4] Schulman J, Wolski F, Dhariwal P,et al.Proximal Policy Optimization Algorithms[J]. 2017. DOI: 10.48550/arXiv. 1707.06347.

[5] Zhang C, Song W, Cao Z, et al. Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning. 2020.DOI:10.48550/arXiv.2010.12367.

[6] Hamilton W, Ying Z, Leskovec J. Inductive representation learning on large graphs[J]. Advances in neural information processing systems, 2017, 30.

[7] Minghai Yuan, Hanyu Huang, Zichen Li,et al.Amulti-agent double Deep-Q-network based on state machine and event stream for flexible job shop scheduling problem[J].Advanced Engineering Informatics. 2023, 58:102230. DOI: 10.1016/ j.aei.2023.102230.

[8] Lei K, Guo P, Zhao W, et al. A multi-action deep reinforcement learning framework for flexibleJob-shop scheduling problem[J]. Expert Syst. Appl. 2022, 205:117796. DOI:10.1016/j.eswa.2022.117796.

[9] Jacek Błażewicz, Pesch E, Sterna M. The disjunctive graph machine representation of the job shop scheduling problem[J]. European Journal of Operational Research, 2000, 127(2): 317-331. DOI:10.1016/S0377-2217(99)00486-5.

[10] Xu K, Hu W, Leskovec J, et al. How Powerful are Graph Neural Networks?[J]. 2018. DOI: 10.48550/arXiv. 1810.00826.

[11] Ying K C, Lin S W, Cheng C Y, et al. Iterated reference greedy algorithm for solving distributed no-idle permutation flowshop scheduling problems[J].Computers & Industrial Engineering, 2017, 110(aug.): 413-423. DOI:10.1016/ j.cie.2017.06.025.

[12] Y.-Z. L, Q.-K. P, He X, et al. The distributed flowshop scheduling problem with delivery dates and cumulative payoffs[J]. Computers & Industrial Engineering, 2022(165-):165.

[13] Zixiao Pan, Ling Wang, Jingjing Wang, JiawenLu, Deep

reinforcement learning based optimization algorithm for permutation flow-shop scheduling[J], IEEE Trans. Emerg. Top.Comput. Intell. 2021.

[14] Park J, Chun J, Kim S H, et al. Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning[J]. International Journal of Production Research, 2021(4):1-18. DOI:10.1080/00207543.2020.1870013.

[15] Hameed M S A, Schwung A. Graph neural networks-based scheduler for production planning problems using reinforcement learning[J]. Journal of Manufacturing Systems, 2023.

[16] Ruiqi Chen, Wenxin Li, Hongbing Yang, A deep reinforcement learning framework based on an attention mechanism and disjunctive graph embedding for the job-shop scheduling problem[J], IEEE Trans. Ind. Inform. 2022,19 (2) : 1322-1331.