Wuhan University
Journal of Natural Sciences

2022, Vol.27 No.6, 531-538

Article ID 1007-1202(2022)06-0531-08
DOI https://doi.org/10.1051/wujns/2022276531

Manufacturing Resource Scheduling Based on Deep Q-Network

□ ZHANG Yufei, ZOU Yuanhao, ZHAO Xiaodong†

School of Electronic and Information Engineering, Tongji University, Shanghai 201804, China

Abstract: To optimize machine allocation and task dispatching in smart manufacturing factories, this paper proposes a manufacturing resource scheduling framework based on reinforcement learning (RL). The framework formulates the entire scheduling process as a multi-stage sequential decision problem, and further obtains the scheduling order by the combination of deep convolutional neural network (CNN) and improved deep Q-network (DQN). Specifically, with respect to the representation of the Markov decision process (MDP), the feature matrix is considered as the state space and a set of heuristic dispatching rules are denoted as the action space. In addition, the deep CNN is employed to approximate the state-action values, and the double dueling deep Q-network with prioritized experience replay and noisy network (D3QPN2) is adopted to determine the appropriate action according to the current state. In the experiments, compared with the traditional heuristic method, the proposed method is able to learn high-quality scheduling policy and achieve shorter makespan on the standard public datasets.

Key words: smart manufacturing; job shop scheduling; convolutional neural network; deep Q-network

CLC number: TP 391

Received date: 2022-09-29

Foundation item: Supported by the National Key Research and Development Plan (2019YFB1706401)

Biography: ZHANG Yufei, female, Master candidate, research direction: resource optimization allocation, reinforcement learning. E-mail: zhang_yufei@tongji.edu.cn

† To whom correspondence should be addressed. E-mail: tjcad-zhaoxd@tongji.edu.cn

0 Introduction

With the rapid development of the Internet of Things (IoT), a new networked smart manufacturing mode named cloud manufacturing has made great progress. The scheduling of the manufacturing resource in a cloud manufacturing environment can be modeled as the Job Shop Scheduling Problem (JSSP)[1]. JSSP aims to determine the assignment of machines and the order of jobs within specified constraints. It belongs to the field of combinatorial optimization and has been proven to be a typical non-deterministic polynomial (NP)-hard problem.

Existing researches indicate that there are mainly two types of approaches to solve JSSP, heuristic-based algorithm and learning-based algorithm. The heuristic-based algorithm obtains feasible solutions through search and iteration, including the Tabu search algorithm[2], genetic algorithm[3], and particle swarm optimization algorithm[4]. However, this method suffers from certain drawbacks. For diverse distributed instances, the search pattern is rigid and relies on expert and domain knowledge[5]. With the advancement and application of deep learning, the learning-based algorithm has been proposed to deal with the mentioned challenges. Therefore, reinforcement learning (RL) plays an essential role in solving JSSP. The RL agents are divided into value-based agents and policy-based agents, represented by deep Q-network (DQN)[6] and proximal policy optimization (PPO)[7], respectively. Learning-based method updates the parameters of the network through the interaction between the agent and the environment, which has shown great potential for solving JSSP.

This paper proposes a manufacturing resource scheduling framework based on DQN. In this framework, the whole scheduling process is abstractly converted into a sequential decision problem addressed by the learning-based method. The learned agent can obtain high-quality solutions and the policy is suitable for scheduling instances of different scales. The main contributions in this paper are summarized as follows:

1) We convert the scheduling process into an Markov decision process (MDP), taking the feature matrix as the state space and a series of heuristic dispatching rules as the action space under the premise of determining the scheduling features.

2) We propose a scheduling framework to solve JSSP by adopting deep neural network named the double dueling DQN with prioritized experience replay and noisy network (D3QPN2). The learned dispatch strategy demonstrates the feasibility and effectiveness of the training algorithm.

1 Related Work

1.1 Representation Learning on Scheduling Resources

A crucial step in the interaction between the agent and environment is MDP designing, which includes the definition of the state space, action space, and reward function. The work by Lin *et al*[8] took the feature matrix as the state space and dispatching rules as the action space, the input features consist of customer order features and system features. Zhang *et al*[9] and Park *et al*[10] considered disjunctive graphs as the state space and adopted graph neural network (GNN) to learn the node embedding, while Han and Yang[11] used CNN to extract features. Regarding the reward function, existing studies have focused on short-term rewards and long-term rewards including global scheduling time[9] and machine utilization[12], the specific formulation is required to comprehensively take into account the relationship between the features of operations and the optimization objective.

1.2 Deep Reinforcement Learning for Combinatorial Optimization

There are many recent research efforts on applying learning-based algorithms, especially RL methods, to address combinatorial optimization problems. The existing RL methods are divided into two categories, PPO and DQN. PPO algorithm is a policy-based reinforcement learning algorithm using two neural networks: one for calculating the action distribution and the other for evaluating it. DQN is value-based, which combines value function approximation and neural network, and adopts the target network and experience replay to train the network. Refs. [5, 9, 10] adopted the PPO algorithm to train the agent and learn the scheduling policy, while Refs. [8, 11, 12] used an improved DQN algorithm. There are some extensions based on DQN to enhance performance and stability[13], such as double DQN[14], dueling DQN[15], prioritized experience replay[16], and noisy network[17]. The above variants of DQN are modified in terms of Q-value function, network architecture, experience replay buffer and the strategy of exploration and exploitation, respectively. These improvements bring new research space to solve the scheduling problem.

2 Problem Description

In the standard JSSP, there are m jobs $J = \{J_1, J_2, \cdots, J_m\}$ and n machines $M = \{M_1, M_2, \cdots, M_n\}$. Each job j_i consists of n operations $O = \{O_{i1}, O_{i2}, \cdots, O_{in}\}$, and each operation O_{ij} has two attributes, the processing time p_{ij} and the machine number m_{ij}. The goal of JSSP is to minimize the maximum completion time (makespan) under the specified constraints. There are three constraints: 1) For all operations of the same job, the subsequent operation cannot start until the previous operation is completed; 2) Each operation must be processed by each machine once; 3) A machine can only process one operation at a time and preemption is not allowed.

As illustrated in Fig. 1, JSSP instances can be represented by disjunctive graphs[9]. Let disjunctive graph $G = (O, C, D)$ be a hybrid graph with O as the vertex set. O is the set of all operations in the instance. There are two special nodes, the start node S and the terminal node T, whose processing time is zero. C is the set of conjunctions, the directed arcs, representing priority constraints between operations of the same job. D is the set of dis-

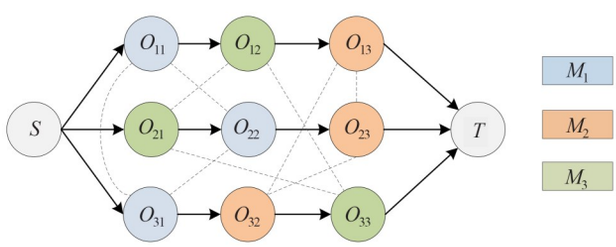


Fig. 1 The disjunctive graph representation of a JSSP instance

junctions, the undirected arcs, representing machine constraints. Each edge connects a pair of operations that require to be processed by the same machine. Therefore, acquiring a solution is equivalent to determining the direction of disjunctions, so that the scheduling result is a directed acyclic graph (DAG).

3 Method

3.1 Proposed Schedule Framework

The proposed framework is implemented to tackle the JSSP under the smart manufacturing environment. The overall architecture is displayed in Fig. 2. The left part is the scheduling environment JsspEnv, and the right part is the network structure named D3QPN2 which is a modified multi-layer neural network based on DQN.

To begin with this process, the JsspEnv initializes the MDP according to the scheduling instances and feeds the current state into the agent. For a given state, the CNN is applied to extract features and approximate the state-action value (Q-value). Then the agent selects the action with the maximum Q-value to feedback to the environment. After receiving the action from the agent, the environment executes the action and calculates the corresponding reward, and then transfers to the next state. The state, action, next state, and reward during the interaction are stored in the replay buffer, and the agent updates the parameters by sampling the transition data from the buffer. After continuous interactions and updates as described above, the agent will be able to learn the converged scheduling policies.

3.2 Markov Decision Process Formulation

The state space, action space, and reward function constitute the majority of the MDP, which is constructed based on the interaction between the agent and the environment. We use Gymjsp[12] as the environment to convert the scheduling instances into MDP.

State In order to simplify the feature representation process and preserve more effective information, the feature matrix is considered as the state space of the manufacturing environment. The feature matrix can be viewed as a multi-channel image, and the subsequent feature processing is performed by CNN. We take the number of jobs m as the length of the image, and the number of operations n as the width. In addition, to describe the current state more comprehensively, each operation is assigned seven scheduling features. Each feature corresponds to a channel, and they are listed as follows:

1) Processing time: the required time of completing the operation.

2) Node Status: three kinds of status, 1 indicates finished, −1 indicates waiting, and 0 indicates processing.

3) Doable flag: True indicates the operation is doable.

4) Waiting time: the waiting time starts to calculate when the previous operation is completed and the current operation is ready to be processed until the specified machine is idle and starts to execute this operation.

5) Remaining time: the remaining time is calculated from the moment the current operation is being processed until the entire operation is completed, and the value is the processing time of the operation minus the time it has been executed.

6) Remaining operation: the number of subsequent operations in the same job. It is calculated from the total number of operations contained in this job minus the

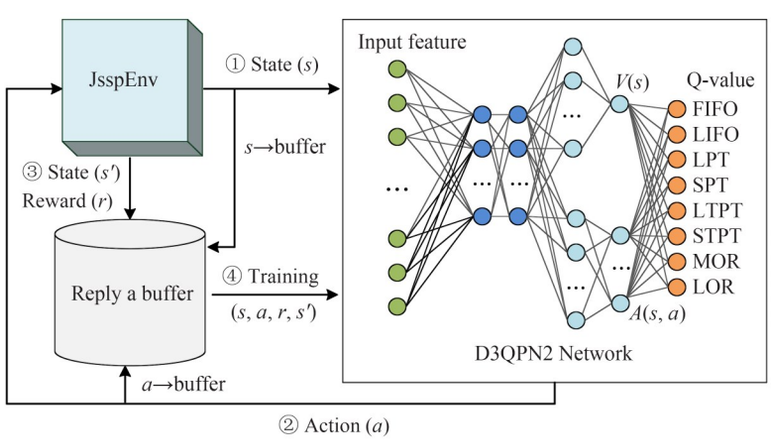


Fig. 2 The overall scheduling framework of D3QPN2

number of operations that have already been executed or are in progress.

7) Completion ratio: the degree of completion of the entire job, namely the number of completed operations as a proportion of all operations in the job.

Action The action space consists of a set of heuristic dispatching rules shown in Table 1. Each dispatching rule prioritizes the set of doable operations corresponding to each machine in accordance with different demands. The agent selects the appropriate rule to schedule operations for specified time steps. Therefore, the final scheduling result is the combination of a series of heuristic rules.

Table 1 Dispatching rules

Rule	Description
FIFO	Process the first job in available operations
LIFO	Process the last job in available operations
LPT	Process the job with the longest processing time
SPT	Process the job with the shortest processing time
LTPT	Process the job with the longest total processing time
STPT	Process the job with the shortest total processing time.
MOR	Process the job with the most operations remaining
LOR	Process the job with the least operations remaining

Reward Regarding reward function, the relationship between single step reward and optimization objective should be considered comprehensively. We adopt machine utilization as the reward function which is defined in Eq. (1). The smaller the number of idle machines, the higher the machine utilization and the greater the rewards obtained.

$$\text{reward} = -\frac{\text{the number of idle\_machine}}{\text{the number of total\_machine}} \tag{1}$$

State transition and terminal criterion Once the agent determines an action based on the current state and feeds it back to the environment, the environment performs the action for designated steps and switches to the next state. The model will update parameters based on the data of the state transition and terminate when the maximum number of episodes is reached.

3.3 Double Dueling DQN with Prioritized Experience Replay and Noisy Network

Based on the DQN algorithm, there are four improvements[13] in different directions, namely double DQN, dueling DQN, prioritized experience replay, and noisy DQN, which can boost the performance of the agent to a certain extent. In our work, the above variants are combined to form D3QPN2 to address scheduling problems. The specific process is shown in Algorithm 1.

Double DQN is adopted to address the problem of the overestimation of state-action value from DQN. As shown in Eq. (2), the action corresponding to the maximum Q value is obtained through the behavior network and then the expected value is calculated by substituting the action into the target network.

$$Y_t^{\text{Double}} = r_{t+1} + \gamma \hat{Q}\left(s_{t+1}, \text{argmax}_a Q\left(s_{t+1}, a; \theta\right); \theta^-\right) \tag{2}$$

Dueling DQN divides the state-action value into two parts which share the same convolutional encoder f_ξ, representing state value function V_η and action advantage function A_ψ, respectively, and then aggregates the two values. By modifying the network architecture, the performance is significantly improved. The formulation of the dueling network is as follows:

$$Q(s,a;\theta) = V_\eta\left(f_\xi(s)\right) + A_\psi\left(f_\xi(s), a\right) - \frac{\sum_{a'} A_\psi\left(f_\xi(s), a'\right)}{\text{the number of actions}} \tag{3}$$

The traditional DQN samples transitions uniformly and randomly from the replay buffer. While the prioritized experience replays samples data with the priority p related to the absolute value of temporal difference (TD) error. New transitions are stored with the highest priority to ensure that they are sampled at least once. The formulation of TD error is as follows:

$$\text{TD error} = r_{t+1} + \gamma \hat{Q}\left(s_{t+1}, \text{argmax}_a Q\left(s_{t+1}, a; \theta\right); \theta^-\right) - Q(s_t, a_t; \theta) \tag{4}$$

The noisy network is an exploration and exploitation strategy different from epsilon-decreasing strategy. For the same state, the latter adopts a completely random strategy and may select different exploration actions, while the noise network using the factorized Gaussian noise will select the same exploration actions.

$$w = \mu^w + \sigma^w \odot \varepsilon^w \tag{5}$$

$$b = \mu^b + \sigma^b \odot \varepsilon^b \tag{6}$$

$$y = b + wx \tag{7}$$

As shown in Eqs. (5) to (7), ε^w and ε^b are Gaussian noise vectors with mean value of 0, the other variables are network parameters, and $\odot$ is an operator used to multiply elements by elements.

Algorithm 1 D3QPN2

Input: Environment JsspEnv, random parameters of the network

Output: The learned scheduling agent

Initialize noisy behavior network Q with random weights θ

Initialize target network $\hat{Q}$ with random weights $\theta^{-}=\theta$

Initialize capacity N of replay memory D, target network update frequency F, schedule cycle C

For episode = 1 to Maximum **do**

reset the JsspEnv and get state s

While done == False

Select dispatching rule $a=\text{argmax}_a Q(s,\ a;\theta)$ from noisy behavior network Q

Execute a for C times, calculate the reward r and obtain the next state s'

Store transition $(s,\ a, r,\ s',\ \text{done})$ in D with highest priority

Sample minibatch $(s_j,\ a_j, r_j,\ s'_j,\ \text{done}_j)$ of transitions according to priority p from D

set $y_j=\begin{cases} r_j & \text{if terminal} \\ r_j+\gamma\hat{Q}\left(s'_j, \text{argmax}_a Q\left(s'_j,\ a_j;\theta\right);\theta^{-}\right) & \text{otherwise} \end{cases}$

Calculate the loss $L=(y_j-Q(s_j,a_j;\theta))^2$ and update priority p

Perform a gradient step on loss with respect concerning parameters θ

End While

Every F times, update the network $\hat{Q}$: $\theta^{-}=\theta$

End For

4 Experimental Results

In this section, we evaluate the performance of the proposed method under the scheduling environment Gymjsp[12]. Compared with single fixed heuristic dispatching rule and genetic algorithm, our approach can achieve shorter makespan on public JSSP instances.

4.1 Experimental Setup

Dataset Our experiments are conducted on a set of public JSSP instances of different sizes. The scheduling dataset consists of Lawrence (la)[18], Applegate and Cook (orb)[19], Taillard (ta)[1], and Storer, Wu and Vaccari (swv)[20]. The above dataset contains different numbers of jobs and machines, in addition to setting priority constraints on operations and machine constraints. The performance of the scheduling method is evaluated by the scheduling time of the instances, also known as makespan.

Baselines There are hundreds of heuristic-based algorithms proposed for JSSP in previous studies with varying principles and performance. In order to save computational costs, we cannot compare them exhaustively, so the appropriate baselines are selected for comparison. As described in this method, the action space is represented by a collection of heuristic dispatching rules, therefore we choose eight single fixed rules as the baseline, including FIFO, LIFO, LPT, SPT, LTPT, STPT, MOR and LOR. Detailed information of single rule is shown in Table 1. In addition, to facilitate comparison between traditional heuristic algorithms and learning-based methods, the genetic algorithm (GA)[21] is considered as another baseline. The mentioned baselines are implemented in Python and set the same parameters.

Models and configurations The fixed hyperparameters for training are shown in Table 2. The CNN architecture is adopted to approximate the Q-value. Firstly, three convolutional layers are used for feature extraction, and then the fully connected layer is employed to calculate the state value and action advantage, which are combined into Q-value. For each instance size, we train the agent network for 8 000 epochs on Pytorch 1.7.1 with CUDA 10.1. The information on hardware is as follows: 1) CPU: Intel(R) Xeon(R) Gold 5220 CPU@ 2.20GHz; 2) GPU: GRID V100DX-16C.

Table 2 Hyperparameters for training

Hyperparameter	Value
Number of training episodes	8 000
Buffer size	10 000
Batch size	64
Target Q update frequency	100
Schedule cycle	8
Discount factor γ	0.99
Prioritized replay α	0.6
Prioritized replay β	0.4
Learning rate	0.000 1

4.2 Results and Discussion

To explicitly compare our approach with baselines, the scheduling results on the different methods are shown in Table 3. The curves of reward value and makespan during the training process are displayed in Fig. 3, where the instance la31(30×10) is taken as an example. Figure 4 presents the Gantt chart of the final scheduling result, taking an example of instance orb01 (10×10). It is added here that the instance swv11(50×10) is only trained for 3 000 epochs due to its large scale and high training cost, and the instance swv01(20×10) shows the convergence results for 5 000 epochs.

Table 3 presents the makespan on different sizes of instances obtained from our method and baselines. The makespan is considered as the evaluation index, and the smaller the makespan, the better the performance of the corresponding method. What stands out in the table is that our approach yields optimal results on each scale instance. Specifically, it can be observed that the genetic algorithm can achieve scheduling results close to D3QPN2 when the instance size is small, while as the size becomes larger, its performance turns worse compared to other heuristic dispatching rules, demonstrating that the genetic algorithm is hardly effective in solving large-scale scheduling problems. In addition, there was a significant difference among the eight single fixed dispatching rules. In terms of average scheduling time, SPT performs the best and LPT has the worst performance but is still a little in advance of the genetic algorithm. Calculated from the average time, the performance of D3QPN2 exceeds GA by over 22%, and as for the optimal single rule SPT, it still improves by 10%. Taken together, these results suggest that our method is able to achieve a better solution than other baselines with shorter makespan, that is to say, the learning-based approach does perform better than the traditional heuristic-based approach selected in our experiments.

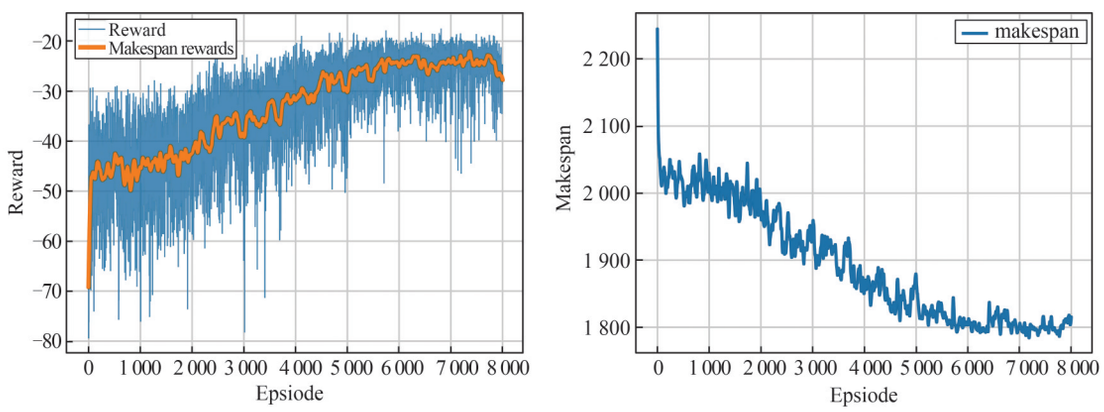
Figure 3 displays the reward and makespan curve. During the training process, the reward value continuously rises and gradually converges, while the scheduling time tends to be the opposite, becoming progres-

Table 3 Results on the standard public datasets

Instance	Size	Dispatching rule								GA	Ours
		FIFO	LIFO	LPT	SPT	LTPT	STPT	MOR	LOR		
la01	10×5	830	764	822	751	835	933	763	941	694	**675**
la06	15×5	1 078	1 031	1 125	1 200	1 098	1 012	926	1 095	952	**926**
la11	20×5	1 577	1 580	1 467	1 473	1 416	1 446	1 222	1 586	1 311	**1 222**
la21	15×10	1 417	1 479	1 451	1 324	1 278	1 541	1 251	1 547	1 394	**1 196**
la31	30×10	2 148	2 256	2 245	1 951	2 083	2 270	1 836	2 129	2 336	**1 834**
orb01	10×10	1 456	1 495	1 410	1 478	1 308	1 458	1 307	1 410	1 378	**1 134**
ta01	15×15	1 830	1 627	1 701	1 462	1 639	1 501	1 438	1 737	1 808	**1 401**
swv01	20×10	1 889	2 123	2 145	1 737	1 961	1 751	1 971	1 838	2 167	**1 642**
swv06	20×15	2 243	2 331	2 542	2 140	2 327	2 360	2 287	2 383	2 794	**1 971**
swv11	50×10	3 808	3 744	4 763	3 714	3 928	3 834	4 642	4 029	5 143	**3 505**
Average	—	1 827	1 843	1 967	1 723	1 787	1 810	1 764	1 869	1 997	**1 550**

sively shorter and fluctuating smoothly within a certain range. Due to the exploration and exploitation strategy of reinforcement learning, the curve oscillates continuously through training, but a convergence trend remains observable, and the oscillations become smaller at later stages. This oscillation is also variable for instances of different sizes, as the size gets smaller, the state space reduces. In addition, the speed of convergence of the reward value is also related to the instance size. The smaller the size of the instance is, the faster it converges.

Figure 4 presents a Gantt chart of the scheduling order. It can be clearly seen that different colors are used to denote different categories of jobs, and the beginning and completion times of the operation on the specified machine are also visualized. By verifying the constraints such as instance data, the corresponding machine numbers and processing time, it can be demonstrated that the agent can efficiently and correctly accomplish the entire scheduling process. The results also illustrate the feasibility of the proposed method.



(a) Reward curve of D3QPN2 of JsspEnv-la31 (b) Makespan curve of D3QPN2 of JsspEnv-la31

Fig. 3 The training curves on la31 (30×10)

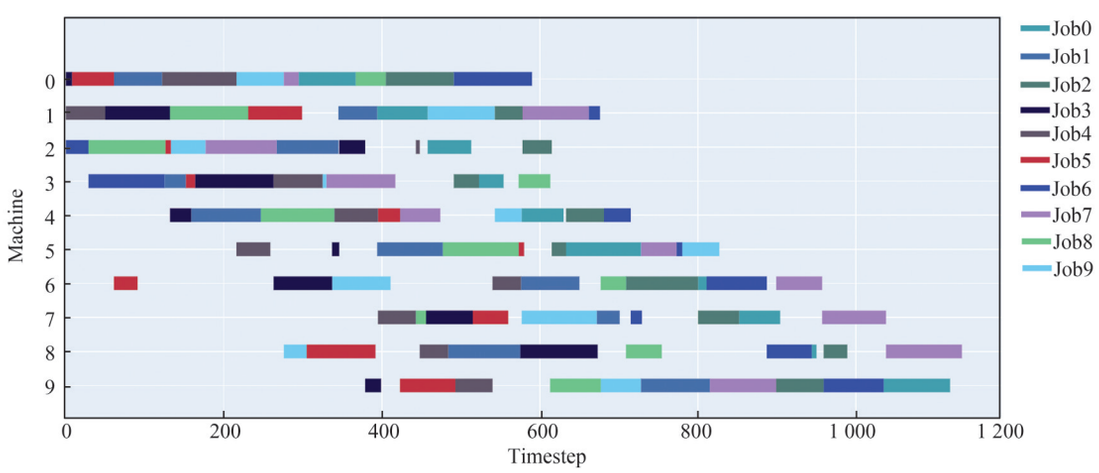


Fig. 4 The scheduling result on orb01 (10×10)

5 Conclusion

In this paper, we propose a manufacturing resource scheduling framework based on improved DQN to solve JSSP. Taking the feature matrix as the state space and the set of heuristic dispatching rules as the action space, we formulate the whole scheduling process as an MDP. In our framework, the dispatching features of operations are extracted by CNN, and it can learn a high-quality scheduling strategy using D3QPN2 from the transitions. The experimental results on the standard public dataset show that the agent can achieve better performance than traditional methods and single fixed dispatching rule under the premise of convergence.

In future research, we intend to explore more complex conditions, including multiple objectives and uncertainties. Additionally, since the current policy can schedule instances of the same size as the training ones, subsequent studies will concentrate on enhancing the generalization of the model.

Reference

[1] Taillard E. Benchmarks for basic scheduling problems [J]. *European Journal of Operational Research*, 1993, **64**(2): 278-285.

[2] Vela C R, Afsar S, Palacios J J, *et al*. Evolutionary tabu search for flexible due-date satisfaction in fuzzy job shop scheduling [J]. *Computers & Operations Research*, 2020, **119**: 104931.

[3] Liu S C, Chen Z G, Zhan Z H, *et al*. Many-objective job-shop scheduling: A multiple populations for multiple objectives-based genetic algorithm approach [J]. *IEEE Transactions on Cybernetics*, 2021: 1-15. DOI: 10.1109/TCYB.2021.3102642.

[4] Ding H J, Gu X S. Improved particle swarm optimization algorithm based novel encoding and decoding schemes for flexible job shop scheduling problem [J]. *Computers & Operations Research*, 2020, **121**: 104951.

[5] Ni F, Hao J Y, Lu J W, *et al*. A multi-graph attributed reinforcement learning based optimization algorithm for large-scale hybrid flow shop scheduling problem [C]// *Proceedings of the* 27*th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. New York: ACM, 2021: 3441-3451.

[6] Mnih V, Kavukcuoglu K, Silver D, *et al*. Human-level control through deep reinforcement learning [J]. *Nature*, 2015, **518**(7540): 529-533.

[7] Schulman J, Wolski F, Dhariwal P, *et al*. Proximal policy optimization algorithms [EB/OL]. [2022-09-10]. *https://arxiv.org/abs/*1707.06347.

[8] Lin C C, Deng D J, Chih Y L, *et al*. Smart manufacturing scheduling with edge computing using multiclass deep Q network [J]. *IEEE Transactions on Industrial Informatics*, 2019, **15**(7): 4276-4284.

[9] Zhang C, Song W, Cao Z G, *et al*. Learning to dispatch for job shop scheduling via deep reinforcement learning [C]// *Proceedings of the* 34*th International Conference on Neural Information Processing Systems*. New York: ACM, 2020: 1621-1632.

[10] 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, **59**(11): 3360-3377.

[11] Han B A, Yang J J. Research on adaptive job shop scheduling problems based on dueling double DQN [J]. *IEEE Access*, 2020, **8**: 186474-186495.

[12] Zeng Y H, Liao Z J, Dai Y Z, *et al*. Hybrid intelligence for dynamic job-shop scheduling with deep reinforcement learning and attention mechanism [EB/OL]. [2022-09-10]. *https://arxiv.org/abs/*2201.00548.

[13] Hessel M, Modayil J, van Hasselt H, *et al*. Rainbow: Combining improvements in deep reinforcement learning [J]. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018, **32**(1): 3215-3222.

[14] van Hasselt H, Guez A, Silver D. Deep reinforcement learning with double Q-Learning [C]//*Proceedings of the* 30*th AAAI Conference on Artificial Intelligence*. New York: ACM, 2016: 2094-2100.

[15] Wang Z Y, Schaul T, Hessel M, *et al*. Dueling network architectures for deep reinforcement learning [C]// *Proceedings of the* 33*rd International Conference on International Conference on Machine Learning*. New York: ACM, 2016: 1995-2003.

[16] Schaul T, Quan J, Antonoglou I, *et al*. Prioritized experience replay [EB/OL]. [2022-09-10]. *https://arxiv. org/abs/*1511.05952.

[17] Fortunato M, Azar M G, Piot B, *et al*. Noisy networks for exploration [EB/OL]. [2022-09-10]. *https://arxiv. org/abs/*1706.10295.

[18] Lawrence S. *Supplement to Resource Constrained Project Scheduling*: *An Experimental Investigation of Heuristic Scheduling Techniques* [R]. Pittsburgh: Carnegie Mellon University, 1984.

[19] Applegate D, Cook W, Bonn U, *et al*. *A Computational Study of the Job-Shop Scheduling Problem* [M]. Bonn: Rheinische Friedrich-Wilhelms-Universität, 1990.

[20] Storer R H, Wu S D, Vaccari R. New search spaces for sequencing problems with application to job shop scheduling [J]. *Management Science*, 1992, **38**(10): 1495-1509.

[21] Gen M, Tsujimura Y, Kubota E. Solving job-shop scheduling problems by genetic algorithm [C]// *Proceedings of IEEE International Conference on Systems*, *Man and Cybernetics*. New York: IEEE, 1994: 1577-1582.

□