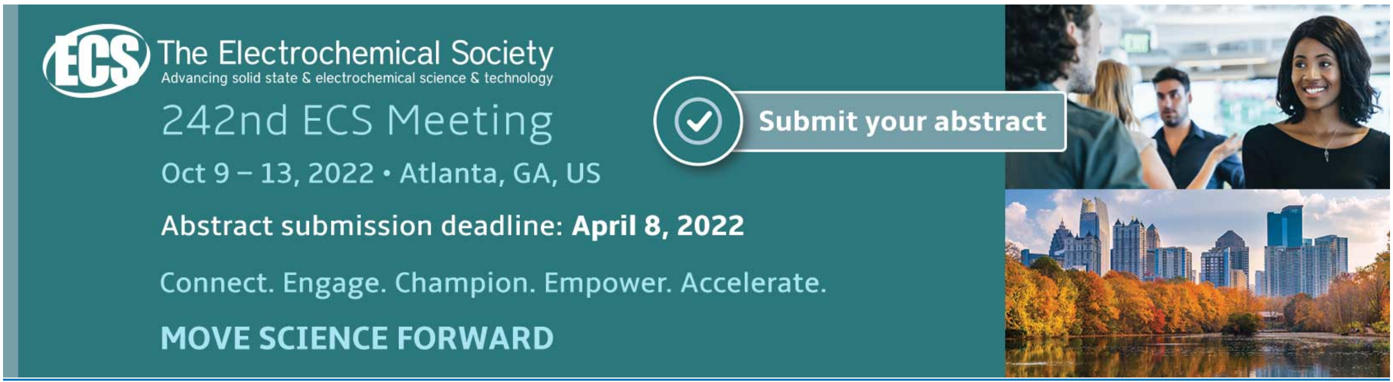
Journal of Physics: Conference Series

PAPER • OPEN ACCESS

An improved deep reinforcement learning approach for the dynamic job shop scheduling problem with random job arrivals



An improved deep reinforcement learning approach for the dynamic job shop scheduling problem with random job arrivals

Bin Luo[1,2], Sibao Wang[1,2,*], Bo Yang[1,2], Lili Yi[1,2]

[1]College of Mechanical Engineering, Chongqing University, Chongqing, 400044,

[2]China.b State Key Laboratory of Mechanical Transmission, Chongqing University, Chongqing, 400044, China.

[*]Corresponding author's E-mail: wangsibaocqu@cqu.edu.cn

Abstract. Deep reinforcement learning (DRL) method is a powerful way to solve the dynamic job shop scheduling problems (DJSSP). However, these DRL approaches are dispatching rules-based, meaning they are problem-specific, dependent on experience, and code effort. We propose a double loop deep Q-network (DLDQN) method with exploration loop and exploitation loop to solve DJSSP under random job arrivals, aiming to minimize the makespans of DJSSP. Simultaneously, by integrating into a single agent scheduling system, the proposed method could avoid complicated dispatching rules, enhancing the proposed method's versatility. The experiment results have confirmed the superiority of our method compared to other algorithms.

1. Introduction

Job shop scheduling problem (JSSP) is a typical NP-Hard combinatorial optimization problem[1] JSSP aims at assigning a number of jobs and each job predefined set of operations (tasks) matched specific machines in available time slots to optimize given objectives. Over the past decades, JSSP has been intensively studied and extensive techniques have been developed to solve static JSSP. However, in the real manufacturing systems, the environment is mostly dynamic in nature, like the new job insertions, machine failures, absenteeism of workers. These disturbances make the predetermined schedule inefficient or even invalid. In those real-time events, new job insertions may occur more frequently[2].

To address this problem, researchers abstract the DJSSP as Markov decision process (MDP) where an intelligent agent interacts with dynamic environments in real time and obtain near-optimal or suboptimal policies in a data-driven way. As an essential branch of machine learning methods, reinforcement learning (RL) is a powerful method to deal with MDP[3]. Liu et al.[4] and Luo et al.[5] utilize deep reinforcement learning (DRL) to solve the DJSSP and DFJSP (dynamic flexible job shop scheduling problem),respectively. They adopt continuous production statuses as the input state features of DRL and determine the current optimal dispatching rule as output. However, naive applications of DRL methods to JSSP encounter some limitations. First, traditional reinforcement learning approaches such as Q-Learning or policy gradient based DRL are poorly suited to multi-agent environments. The non-stationary environment from any individual agent's view violates Markov assumptions required for Q-learning convergence, preventing modeling JSSP as a multi-agent system with each machine as an agent [6]. Second, the dispatching rules served as action space relies on designers' prior knowledge and parameter design experience, and it is challenging to design and select appropriate dispatching rules to achieve the optimization objectives with high quality. In contrast, it is easy for the DRL agent to participate in the scheduling directly, so the indirect way of scheduling through the dispatching rules does not release the DRL method's scheduling potential.

In this work, we utilize an improved DRL to solve the DJSSP considering New job insertions aiming to minimize the makespan. The primary contributions of this paper include:

•Two sets of the continuous state feature about job processing time adopted in DRL to describe production information overcome the inaccurate expression of continuous production features in traditional RL.

• A new application of DRL in DJSSP not based on artificial dispatching rules improves the versatility and the scheduling performance of the DRL method.

•Integrate the entire DJSSP into a single agent scheduling system to overcome multi-agent mutual interference causing the algorithm to fall into local optimality.

•An improved double loop deep Q network(DLDQN) algorithm is developed to achieve optimum through two synchronize processes: exploration loop and exploitation loop

2. Background

2.1. Dynamic job shop scheduling problem

The DJSSP addressed in this paper can be described as follows[7]: There are n successively arriving jobs $J=\{J_1,J_2,...,J_3\}$ to be scheduled on m machines $M=\{M_1,M_2,...,M_m\}$. Each job J_i consists of a predetermined sequence of operations where O_{ij} is the j -th operation of job J_i. Each operation O_{ij} must be processed on the designated machines. Due to the real-time disturbances (e.g., changes in processing time, urgent jobs arrival), the initial plan needs to be revised dynamically to adapt to the actual conditions. The notations used for the problem are given as Table 1:

Table 1. The notations for the problem formulation

Parameters	Description
t	the decision time t every time a new job arranged
T_t	total number of all scheduled operations in current t
T	max number of T_t
O_{ijk}	the j -th operation of job J_i is processed on the machine k
C_{ijk}	the Processing time of O_{ijk}
S_{ijk}	the Start time of O_{ijk}
E_{ijk}	the end time of O_{ijk}
X	X number of operations

Based on the above notations and assumptions, the considered DJSSP with new job arrivals is formulated as follows:

$$f=min_{1\leqslant k\leqslant m}\left\{max_{1\leqslant i\leqslant n}\left\{C_{ijk}\right\}\right\} \tag{1}$$

$$s.t\begin{cases}E_{ijk}\leqslant S_{i(j-1)k'} & (a)\\ E_{ijk}-S_{ijk}=C_{ijk} & (b)\\ E_{ijk}<E_{i'j'k'} & (c)\\ T=n\cdot m & (d)\\ T=max\left(T_t\right) & (e)\end{cases} \tag{2}$$

The objective of dynamic scheduling is to minimize the makespan as in Eq. (1). Eq. (2)(a) indicates that the job's subsequent process can not begin until the previous process has been completed. Eq. (2)(b) indicates that there has been no interruption in the operation of one job. Eq. (2)(c) indicates that each machine can only process one operation at a time. Eq. (2)(d) and Eq. (2)(e)suggested that each operation can be assigned on only one machine.

2.2. The Necessities of single-agent DRL setting in DSJSP

Reinforcement learning (RL)[8] is a learning process in which an agent can periodically observe the state, make decisions, get the results from the environment, and adjust its strategy to achieve the optimal policy. Deep Q Network (DQN)[9] is an algorithm branch of the deep reinforcement learning (DRL) method, where DQN agent employs a neural network parameterized by the weight θ to approximate the objective $J(\theta) = \mathrm{E}_{s\sim p_{\pi}, a\sim\pi}(R_t)$ by updatingg, the RL agent executes action a from state s under a given policy π_θ taking the expected return gradient $\nabla_\theta J(\theta)$.

Although some papers[4, 5] have tried to apply DRL's latest achievements to DJJSP, DQN research in the multi-agent field is still immature. In a multi-agent setting, the Markov property assumpted as a requirement for the convergence of RL turns invalid with agents independently updating their policies, which turns the non-stationary of the algorithm, and the probability of taking a gradient step in the correct direction decreases exponentially with the number of agents, namely $P\left(\hat{\nabla}J, \nabla J > 0\right) \propto (0.5)^N$ [6].

However, single-agent systems will not encounter the instability problems caused by multi-agent setting. Therefore, in terms of convergence, a single-agent setting to DJSSP will be more reasonable.

3. Proposed methods

3.1. Definition of state features

The key to applying DRL to DJSSP is to select appropriate state features to express the current production status. Traditionally, scheduling experts rely on their knowledge and experience to extract state features that require a considerable amount of time, experience, and code effort. Simultaneously, the DRL scheduling system's performance based on the dispatching rules is sensitive to the quality of artificially state features, meaning these features are carefully designed for specific problems and thus low versatility. To enhance the algorithm's versatility, we hope to use simple but effective status features to reflect the current production status. Many methods use machine status as crucial information, such as the average utilization rate of machines. However, considering neural networks' ability to extract deep features from original information, we selected two sets of representative state features about the job's processing time as the algorithm's state input. Simultaneously, different state indexes often have different dimensions and dimensional units, i.e., the number of machines and the processing time of current operations can be significantly different. Therefore, the normalization technique is adopted to constrain the different state inputs of DLDQN to [0, 1] thus enable faster and more stable algorithm training. the state features at each rescheduling point *t* are listed as follows:

The ratio of each job's processed operation time to each job's total operations time $PO_i(t)$, as defined in Eq. (3):

$$PO_i(t) = \frac{\sum_{j=1}^{T_{it}}\left(E_{ijk} - S_{ijk}\right)}{\sum_{j=1}^{m} C_{ijk}}, for\, i = 1,2,\ldots,n \tag{3}$$

The ratio of each job's processed operation time to average processed time $POA_i(t)$, as defined in Eq. (4):

$$POA_i(t) = \frac{n \cdot \sum_{j=1}^{T_{it}}\left(E_{ijk} - S_{ijk}\right)}{\sum_{i=1}^{n}\sum_{j=1}^{T_{it}}\left(E_{ijk} - S_{ijk}\right)}, for\, i = 1,2,\ldots,n \tag{4}$$

Based on the definitions above, we can get two state vectors ϕ_t^1, ϕ_t^2 as the inputs of the system, which are defined in Eq. (5) and Eq. (6):

$$\phi_t^1 = \left\{PO_1(t), PO_2(t), \ldots, PO_n(t)\right\} \tag{5}$$

$$\phi_t^2 = \left\{ POA_1(t), POA_2(t), \ldots, POA_n(t) \right\} \tag{6}$$

3.2. *Action Space*

Unlike the fixed number of small state space, this work's action space corresponds to the total number of jobs n. At the time step t , the system will output only one job and therefore admit one corresponding operation into the scheduling queue and generate a history track of operation containing T_t processed operation. As illustrated in Figure 1, in the exploration loop, the scheduling sequence incorporating the new operation will be temporarily stored in the exploration set. If there are remaining unprocessed operations, the sequence will be handed over to the environment to calculate the new state features and then output a new job with a corresponding operation. The exploration loop will not stop until the scheduling is completed.

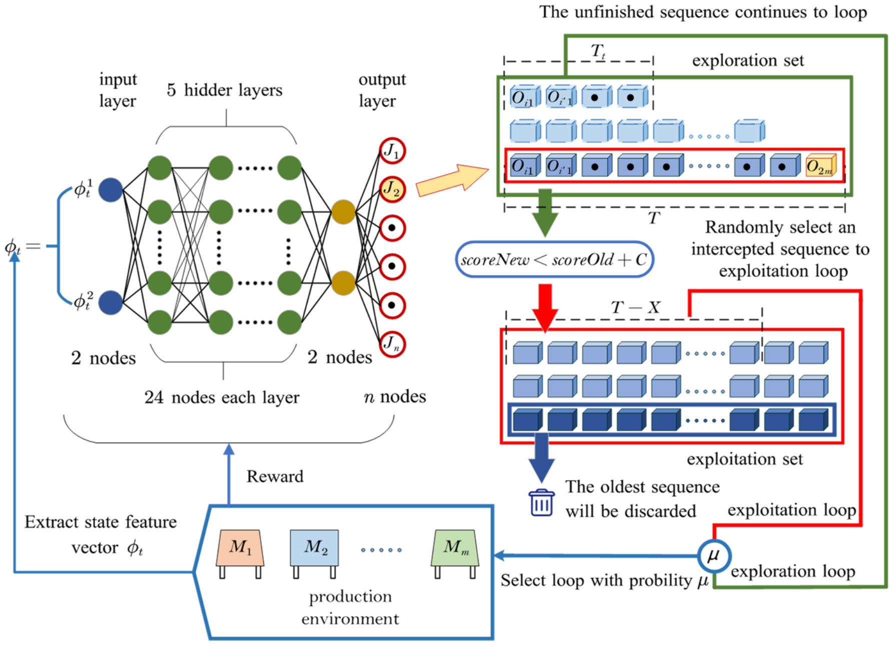


Figure 1. Structure of the DLDQN

3.3. *Overall framework of the training method*

The goal of the two-loop training is to minimize makespan by taking a sequence of actions. The different process needs to design different reward functions for action outputs. The procedure to calculate r_t is given in Table 2.

The training process is based on the exploration loop and the exploitation loop. Fig. 1 shows the complete DJSSP training flow of the scheduling system. The state vector and the weights of the network are initialized randomly. The deep architecture we use in the experiments includes one input layer with 2 neurons to receive two state feature vectors, five hidden layers with 24 neurons, and one hidden layer with 2 neurons. The numbers of neurons in the output layer are consistent with the total number of jobs n. At the beginning of a loop, the agent system will take an action according to the current state, namely, determining the job to be processed at the current step, and the exploration set will temporarily store the new operation sequence. Once a new operation is scheduled, the state transits to the next state, and the environment gives an immediate reward to the DQN agent and according to Algorithm 1.

In the exploitation process, the scheduling sequence that meets the exploration loop's filtering requirements will be stored in exploitation set. When the exploration set stores a certain number of scheduling sequences, a sequence will be randomly selected and cut to a particular length. This part of the sequence interacts with the environment to calculate the features and participates in the exploitation loop to build a new scheduling sequence. If a better makespan can be obtained, the newly developed sequence will also be stored in the exploitation set. The exploration loop and the development loop alternate with probability μ to avoid falling into the local optimum.

Table 2. Definition of the reward r_t at each decision point t

```
Algorithm 1 Definition of the reward $r_t$ at each decision point $t$
if $T_t < T$ then
    if Not Done (Not Done means the system has not selected the finished job at each decision point $t$ ) then
        $r_t \leftarrow T_t$ (The $r_t$ at the current decision point $t$ will increase proportionally with the number of all scheduled operations )
    else
        $r_t \leftarrow -C_{\max}$ ( $C_{\max}$ is the maximum completion time at the end of the current loop)
    end if
else
    $r_t \leftarrow makespan \cdot \frac{makespan}{C_{\max}}$ ( At this time $T_t < T$ , which means that the algorithm has processed all the Jobs, and $r_t$ will increase as $C_{\max}$ decreases)
end if
```

4. Numerical experiments

In this section, computational experiments are carried out to investigate the effectiveness of the proposed rescheduling algorithm. At first, a sensitivity study of parameter μ of selecting the two cycles is conducted. Then, we compare the performance of DLDQN with the ACDRL algorithm under different benchmark problem instances of the traditional JSSPs. To show the algorithm's rescheduling ability, we simulate a new job insertion in different size instances and verify the rescheduling scheme's superiority over the previous schemes. The proposed algorithm is coded in Python 3.6 and runs on a PC with Intel Core i5-8300 @ 2.3GHz processor and 4GB RAM.

4.1. Sensitivity study on the control parameter μ

The loop control parameter μ plays an essential role in affecting the proposed algorithm's practical implementation performance. To determine the appropriate value of μ , we increase it from 0 to 1 every 0.1 and independently train the agent system 20 times in the FT06 practical problem under each value. The box plots of makespan from 20 runs under different values of μ are provided in Figure 2.

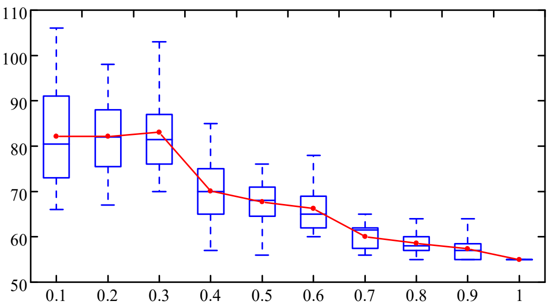


Figure 2. Box plots of makespan under different levels of μ. X axis μ; Y axis: makespan

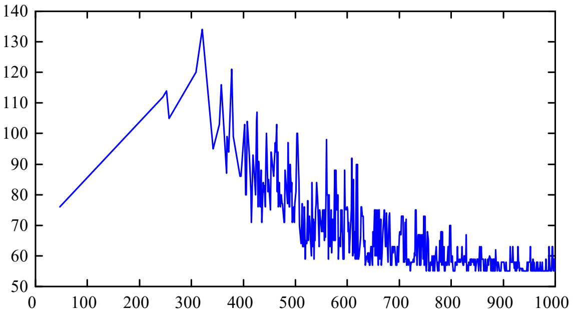


Figure 3. The makespan obtained by the DLDQN in 1000 training loop whenμ=1. X axis: number of training loop; Y axis: makespan.

The μ value determines the probability of choosing the exploration loop. It can be observed that when the μ value is less than 0.3, makespan has a broader distribution and a higher mean. The smaller μ value weakens the exploration loop so that the complete scheduling sequence obtained by the exploitation loop is limited and low quality, which makes the whole algorithm easy to fall into local optimization. The increase in μ value after 0.4 strengthens the exploration loop so that the exploitation obtains more high-quality scheduling sequences, and when μ =1, the algorithm achieves the lowest degree in terms of distribution range and the average value of makespan. Figure 3. demonstrates the convergence of our proposed approach on makespan in the first 1000 training episodes when $\mu = 1$. We can see that a larger value within 300 episodes leads to a higher probability of choosing an exploration loop. At this phase, the algorithm mainly learns $T_t \rightarrow T$ to complete the whole scheduling successfully and avoid invalid scheduling. Then, as the μ decreases, the exploitation loop decreases, the exploitation loop and exploration loop alternate to promote the algorithm to reach the optimum. A higher μ value will provide the algorithm with sufficient initial explosive power to avoid falling into the local optimum, but μ greater than 1 means that only the exploration cycle will be carried out, which will delay the starting point of the exploitation loop and make μ still too large at the training's end, resulting in the instability of the algorithm. We conclude that μ = 1 is recommended in this work.

4.2. Comparison with the MAQL

To demonstrate the performance of the DLDQN for solving JSSPs, three algorithms of JSSPs are utilized for comparison. The comparison algorithms are listed as follows. To the best of our knowledge, ACDRL proposed by Liu et al.[4]are the first attempts to apply the DRL algorithm to DJSSP, explaining why we only compare with this DRL method. ACDRL is a branch of the DRL method.

Optimal solution (OPT)

Traditional RL: Multi-Agent Q-learning(MAQL)[10]

ACDRL: Actor-Critic Deep Reinforcement Learning[4]

Comparison results are listed in Table 3. The average makespan over 20 runs for each instance are evaluated. Compared with the ACDRL, the DLDQN can obtain a lower makespan in almost all instances, meaning that DLDQN does not rely on scheduling rules to obtain a better schedule. Moreover, note that the ACDRL method is challenging to achieve the optimum even on the small instance Ft06, but DLDQN has obtained the optimum in most instances, which shows that the DLDQN without scheduling rule constraints can better release the scheduling ability of DRL method. Compared with QL, the DLDQN demonstrates better performance in almost all instances, showing that the discrete state's features of the stand QL algorithm are rough in expressing the production statuses of DJJSP, which is not as reasonable and effective as DLDQN`s continuous state features.

Table 3. Comparison Results of makespan for each instance

Instance	OPT	MAQL (Cmax)	ACDRL (Cmax)	DLDQN (Cmax)
Ft06(6×6)	55	57	58	**55**
Orb02(10×10)	888	931	1002	**888**
Orb03(10×10)	1005	1095	1150	**1005**
Orb04(10×10)	1005	1068	1132	**1005**
Orb05(10×10)	887	976	1045	**887**
Orb06(10×10)	1010	1064	1106	**1010**
Orb07(10×10)	397	424	468	**397**
Orb08(10×10)	899	956	1022	**899**
Orb09(10×10)	934	996	1082	**934**
Average	936	978	1026	**936**

In conclusion, DLDQN is more reasonable and efficient in obtaining optimal makespan due to its accurate continuous state features and not constrained by dispatching rules.

4.3. The rescheduling ability of the proposed algorithm

To verify the algorithm's rescheduling ability, we choose three different size instances of FT06(6×6), LA02(10×6), and Orb06(10×10) to analyze the impact of new job arrivals on the agent system. As shown in Tables 4-6, "Multiples" means that the newly arrived job's processing time is 1.0, 1.5, 2.0, 2.5 times the average processing time of all operations in different instances. " T_t "means the time point at which the new job arrives, i.e., 20%, 40%,60%, and 80% of T . "Static " denotes the optimal scheduling scheme obtained by the algorithm under static conditions. We assume that after the new job arrives and is processed, the subsequent plan is still executed, thus obtaining this static scheme's makespan in the dynamic environment. The new scheme`s makespan rescheduled by agent system encountering new job arriving is marked as " new." Results indicate that the previous optimal scheme makespan is influenced by different arrival times and job`s operation time. The arrival of new jobs at 20% and 40% T_t have a more significant impact on the static scheme than new jobs arriving at 80% T_t . For the arrival of new jobs at 20% and 40%, the rescheduling plan is 100% better than the previous plan in FT06, and only one item equal in LA02 and Orb06, showing that the DLDQN agent has a sufficient rescheduling ability for the early new job insertion. When the arrival time has little effect on the production plan, i.e., T_t is 80%, the two schemes' makespan is close to the same, showing that the DLDQN agent realizes that sticking to the original static scheme is a better choice. When the T_t is 60%, the DLDQN agent will evaluate the impact of the new job insertion and determine to stick to the original scheme or to reschedule to get a better scheme.

Table 4. Comparison results of the static scheme with the new scheme in Ft06.

multiples	T_t: 20%		T_t: 40%		T_t: 60%		T_t: 80%	
	static	new	static	new	static	new	static	new
1.0	60	**58**	60	**58**	60	**58**	59	59
1.5	63	**60**	63	**58**	63	63	63	**62**
2.0	65	**62**	65	**58**	65	**63**	64	64
2.5	68	**61**	68	**61**	68	**66**	67	67

Table 5. Comparison results of the static scheme with the new scheme in La02.

multiples	T_t: 20%		T_t: 40%		T_t: 60%		T_t: 80%	
	static	new	static	new	static	new	static	new
1.0	699	**689**	707	**674**	**707**	713	670	670
1.5	726	**718**	734	**726**	734	734	697	697
2.0	752	**742**	760	**723**	760	760	723	723
2.5	779	**769**	787	787	787	787	750	750

Table 6. Comparison results of the static scheme with the new scheme in Orb06.

multiples	T_t: 20%		T_t: 40%		T_t: 60%		T_t: 80%	
	static	new	static	new	static	new	static	new
1.0	1042	**1040**	1064	**1050**	1066	**1066**	1035	1035
1.5	1070	**1068**	1092	**1046**	1094	**1080**	1063	**1036**
2.0	1098	**1089**	1120	**1055**	1122	1122	1091	**1036**
2.5	1126	**1090**	1148	**1079**	1150	1150	1119	**1036**

We conclude that the intelligent system has shown good rescheduling ability in the face of new job insertion, especially early insertion.

5. Conclusions

In this paper, A single-agent system without dispatching rules is developed to address the DJSSP with new job insertions. The action space we adopt is directly oriented to jobs rather than dispatching rules. We propose a double loop architecture to integrate the global search capability of the exploration loop and the exploitation loop's local optimal convergence capability to promote DQN to find the global optimum of the problem space. Two generic continuous features normalized in the range of [0, 1] are utilized to represent a state at each rescheduling point. The results demonstrate that the DLDQN performs better than standard Q-learning and dispatching rules-based DRL. Meanwhile, the comparisons between the static scheme and the rescheduled scheme have further verified that the DLDQN could handle unexpected new job insertion incidents.

In future work, we will extend the current work to multi-objective optimization, specifically, a centralized reward function to constrain multiple single-agent systems with different optimization objectives.

Funding

The presented work was supported by the National Science and Technology Innovation 2030 of China Next-Generation Artificial Intelligence Major Project, (no. 2018AAA0101804) and Natural Science Foundation Project of Chongqing Science and Technology Commission (no. cstc2019jcyj-msxmX0058).

References

[1] M. Nouiri, A. Bekrar, A. Jemai, S. Niar,A. C. Ammari. (2018) An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem. J Intell Manuf, 29: 603-615.

[2] J. Zhang, G. Ding, Y. Zou, S. Qin,J. Fu. (2019) Review of job shop scheduling research and its new perspectives under Industry 4.0. J Intell Manuf, 30: 1809-1830.

[3] W. Zhang. (1996) Reinforcement Learning for Job-Shop Scheduling, Ph. D. Dissertation, Oregon State University.

[4] C.-L. Liu, C.-C. Chang,C.-J. Tseng. (2020) Actor-Critic Deep Reinforcement Learning for Solving Job Shop Scheduling Problems. IEEE Access, 8: 71752-71762.

[5] S. Luo. (2020) Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. Appl. Soft. Comput, 91.

[6] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel,I. (2017) Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. Adv Neural Inf Process Syst, 30: 6379-6390.

[7] M. Zhang, F. Tao,A. Y. C. Nee. (2020) Digital Twin Enhanced Dynamic Job-Shop Scheduling. J Manuf Syst.

[8] R. S. Sutton,A. G. Barto.(2018) Reinforcement learning: An introduction, MIT press, 2018.

[9] K. Arulkumaran, M. P. Deisenroth, M. Brundage,A. A. Bharath. (2017) Deep Reinforcement Learning: A Brief Survey. IEEE Signal Process Mag, 34:26-38.

[10] Y. M. Jiménez. (2012) A generic multi-agent reinforcement learning approach for scheduling problems, PhD, Dissertation, Vrije Universiteit Brussel.