2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD) | 978-1-6654-0527-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/CSCWD54268.2022.9776116

An End-to-End Deep Reinforcement Learning Approach for Job Shop Scheduling

Linlin Zhao
School of mechanical Science and Engeering
Huazhong University of Science and Technology
Wuhan, China
zhaolinlin@hust.edu.cn

Weiming Shen
School of mechanical Science and Engeering
Huazhong University of Science and Technology
Wuhan, China
shenwm@hust.edu.cn

Chunjiang Zhang
School of mechanical Science and Engeering
Huazhong University of Science and Technology
Wuhan, China
zhangcj@hust.edu.cn

Kunkun Peng
School of Management
Wuhan University of Science and Technology
Wuhan ,China
pengkunkun@126.com

Abstract—**Job shop scheduling problem (JSSP) is a typical scheduling problem in manufacturing. Traditional scheduling methods fail to guarantee both efficiency and quality in complex and changeable production environments. This paper proposes an end-to-end deep reinforcement learning (DRL) method to address the JSSP. In order to improve the quality of solutions, a network model based on transformer and attention mechanism is constructed as the actor to enable a DRL agent to search in its solution space. The Proximal policy optimization (PPO) algorithm is utilized to train the network model to learn optimal scheduling policies. The trained model generates sequential decision actions as the scheduling solution. Numerical experiment results demonstrate the superiority and generality of the proposed method compared with other three classic heuristic rules.**

Keywords—Job shop scheduling problem, deep reinforcement learning, proximal policy optimization, end-to-end method, transformer, attention mechanism.

I. INTRODUCTION

Job shop scheduling (JSS) is a typical scheduling type in smart manufacturing. Job shop scheduling problem (JSSP) is essentially a combinatorial optimization problem and has been proven to be a Nondeterministic Polynomial-time (NP) hard problem [1]. At present, JSSPs are solved by assuming a static manufacturing environment that all processing information has been known in advance, and then computing a deterministic scheduling plan without any modification during processing. However, in real production, dynamic events, such as machine breakdowns, order insertions and rework, occur unexpectedly, will lead to a deviation far from original scheduling plan, thus seriously affecting the production efficiency.

JSSPs have been studied extensively in the past decades. The studies mainly have focused on meta-heuristic algorithms, such as tabu search (TS) [2] and genetic algorithm (GA) [3]. Meta-heuristic algorithms, benefits from their robust global searching abilities, can obtain the optimal scheduling solutions. However, meta-heuristic algorithms can’t respond to dynamic events instantly due to their time-consuming characteristics. Heuristic rules, also known as priority dispatching rules (PDRs), can respond to dynamic events in real time, but it is difficult to guarantee optimal solutions especially in a long run due to their short-sightedness. Moreover, PDRs are narrow-minded, which means it is impossible to find a PDR applicable to any scheduling environment. Therefore, raising an algorithm to guarantee both efficiency and quality is the main motivation of this study.

JSSP can be regarded as a sequential decision-making problem, which can be modeled as a Markov decision process (MDP) [4]. Reinforcement learning (RL) [5] is an effective way to address MDPs, which does not require a complete mathematical model of the learning environment and adjusts the learning policy through the rewards obtained by the interaction with the environment. However, RL fails to address large-scale problems and the problems with continuous state space. With the rapid development of deep learning (DL), deep reinforcement learning (DRL), combining the advantages of both DL and RL, has been proposed to address the above problems. One of the earliest work was from Zhang and Dietterich [6], they used Q-learning to train an agent to determine a most appropriate heuristic rule from a predefined rule set to minimize total tardiness. Their method obtained the better scheduling results than common heuristic rules. Later, more and more studies of (D)RL for scheduling problems has been carried out [7-12]. Most of these studies have achieved a certain adaptively of dispatching rules in some degree by selecting the most appropriate dispatching rules according to current production status. However, this type of (D)RL methods still searches in rule space not solution space, which limits the quality of scheduling solutions in some degree. In recently years, the combination of sequence-to-sequence model and DRL provides a new way to address combinatorial optimization problems [13, 14], which is also referred as the end-to-end method. End-to-end methods can output scheduling results directly rather than by executing rules indirectly.

With the motivations above, we intend to develop an end-to-end DRL model to solve JSSPs to minimize makespan. The contributions of this paper can be listed as follows: (1) A network model is proposed to enable the DRL agent to search in solution space, in which we use the transformer network and attention mechanism. (2) A matrix describing the current production states in elements of job is proposed, and proximal policy optimization (PPO) algorithm is utilized to train the proposed network to learn scheduling policies to address JSSPs.

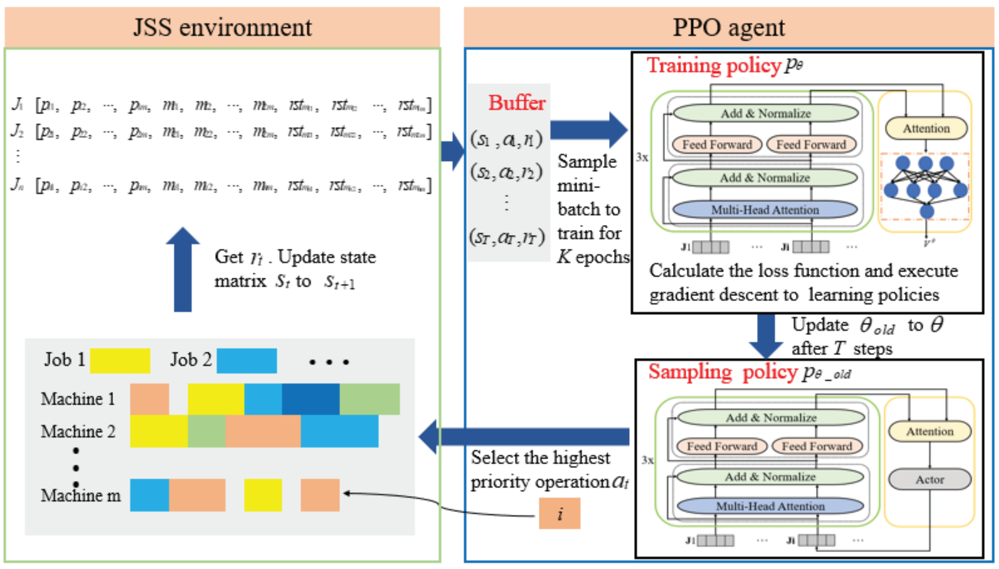
JSS environment
PPO agent
J_1 $[p_{11}, p_{12}, \cdots, p_{1m}, m_{11}, m_{12}, \cdots, m_{1m}, rst_{m_{11}}, rst_{m_{12}}, \cdots, rst_{m_{1m}}]$
J_2 $[p_{21}, p_{22}, \cdots, p_{2m}, m_{21}, m_{22}, \cdots, m_{2m}, rst_{m_{21}}, rst_{m_{22}}, \cdots, rst_{m_{2m}}]$
J_n $[p_{n1}, p_{n2}, \cdots, p_{nm}, m_{n1}, m_{n2}, \cdots, m_{nm}, rst_{m_{n1}}, rst_{m_{n2}}, \cdots, rst_{m_{nm}}]$
Get r_t . Update state matrix S_t to S_{t+1}
Job 1
Job 2
Machine 1
Machine 2
Machine m
Buffer
(s_1, a_1, r_1)
(s_2, a_2, r_2)
(s_T, a_T, r_T)
Sample mini-batch to train for K epochs
Training policy p_θ
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Multi-Head Attention
Attention
3x
v^π
Calculate the loss function and execute gradient descent to learning policies
Update θ_{old} to θ after T steps
Sampling policy p_{θ_old}
Actor
Select the highest priority operation a_t
i

Fig.1. Framework of the proposed end-to-end DRL method

II. Problem Description

JSSP in this paper can be described as follows. There are n jobs $J = \{J_1, J_2, \cdots, J_i, \cdots J_n\}$ and each job $J_i = \{O_{i1}, O_{i2}, \cdots, O_{ii}, \cdots O_{im}\}$ has m operations to be processed on m machines $M = \{M_1, M_2, \cdots, M_i, \cdots M_m\}$. Each operation $O_{i,j}$ has a fixed processing time $p_{i,j}$ and a unique designated machine $m_{i,j}$, which are given in advance. Moreover, there is a process constraint that the operations belong to the same job should be processed one after another in a fixed order. The machine constraint is that each machine can only process one operation at any time. What's more, the JSSP is addressed in the ideal production environment, so machine breakdowns, material shortage, handover time and buffer limitation are all neglected during processing. Therefore, based on the above description, mathematical formulations of JSSP can be established as follows.

$$opt.P = \min\left\{\max_{1\le i\le n} c_{i,m}\right\} \tag{1}$$

$$s.t.\begin{cases} c_{i,j} - p_{i,j} \ge c_{i,j-1} & \text{(a)} \\ c_{i,j} - (1-a_{i,h,k})p_{i,j} \ge c_{h,q} - (1-a_{i,h,k})p_{h,q}, \\ m_{i,j} = m_{h,q} = k & \text{(b)} \\ i,h = 1,2,\cdots n;\ \ j,q = 1,2,\cdots,m \end{cases} \tag{2}$$

Eq. (1) is the objective function in this paper that is to minimize the completion time of the last operation $O_{i,m}$ of all jobs, which is also widely known as minimizing makespan. Eq. (2)(a) is the process constraint for operations of the same job. Eq. (2)(b) guarantees the machine constraint that the next operation can not be processed until the previous operation is finished.

III. Solving JSSP Based on End-to-End DRL

In this section, we introduce an adaptive scheduling framework based on the end-to-end DRL to address JSSP, as shown in Fig.1. First, a network is proposed based on the transformer and attention mechanism to select the next operation to be processed. Then, Proximal policy optimization (PPO), a policy gradient algorithm, is utilized to optimize the proposed network to learn the scheduling policy to minimize makespan.

A. Production environment

DRL is an agent that continuously learns knowledge through interactions with the environment based on rewards or penalties it receives. $<S, A, R>$ is a triple that the agent needs to get by interacting with production environments. Agent makes an action according to the current state matrix S by observing environments, where the agent can also receive a reward R for evaluating the action A, which directly changes the policy of selecting future actions. The definition of the JSS production environment triple $<S, A, R>$ will be elaborated below.

- State

States are the only basis for an agent to make decisions, which should include all the information of decision-making affecting makespan. In the network model of this paper, the input is the sequence in elements of jobs, so each job information needs to contain its own processing information, so that the information of all jobs can be integrated to described the current complete production state. Processing information for a job J_i contains two parts: processing timing $P_i = \{p_{i1}, p_{i2}, \cdots, p_{ir}\}$ and designated machine indexes $M_i = \{m_{i1}, m_{i2}, \cdots, m_{ir}\}$ for its all operations. r is the unscheduled operations of J_i at current decision-making point. Moreover, the relative available starting time for each machine $RST_i = \{rst_{m_{i1}}, rst_{m_{i2}}, \cdots rst_{m_{ir}}\}$ is also provided to indicate the production information that has been scheduled. Relative time can avoid larger time value over time. $rst_k = MA_k - \min(\{MA_i\}), i = 1, \ldots, m$, and MA_k refers to the time allowed to process the next operation on M_k. The job will be removed from the input when its all operation are scheduled. An episode ends when all operations for all jobs are scheduled. Therefore, the state at t of J_i can be described as follows.

$$J_i = \{P_i, M_i, RST_i\} \tag{3}$$

- Action

As mentioned in Section 1, the action obtained by a DRL agent directly points at the job to be processed next, so the action can be provided in the format in Eq. (4).

$$a_t = \{J_1, J_2, \ldots, J_n\} \tag{4}$$

- Reward

The reward is used to modify the policy to achieve the objective function. As mentioned above, the objective of JSSP is to minimize the makespan. However, only when all operations are scheduled, that is the end of an episode, the makespan can be obtained. If only the last action is rewarded, that is known as sparse reward, which will cause the agent to learn slowly or even fail to learn effectively. Therefore, it is necessary to appropriately increase some rewards for some actions in an episode to assist the agent learning. Reward used in this paper is negative machine idle time between two adjacent operations on corresponding machines, that is the difference between the start time of the current operation and the completion time of the previous operation on this machine. The makespan is smaller if the total machine idle time is smaller. Therefore, it is reasonable to take current machine idle time as the reward.

$$r_t = \begin{cases} -(st_t^k - ft_{t-1}^k) \\ -\sum_{k=1}^{m}(C_{mak} - ft^k), \quad if\ t = T \end{cases} \tag{5}$$

where st_t^k is the start time of current operation on M_k, ft_{t-1}^k is the completion time of previous processing operation on M_k. C_{mak} is makespan obtained at the end of an episode at Tth step.

B. Network model

The network model contains two parts, one is to extract features of job floor to provide decision-making support for the actor network, and the other is the actor network to select which job to be processed next. These two parts are connected in series.

To achieve the function searching in solution space, the network input is a sequence in the elements of unprocessed jobs, so that the actor network can make an action corresponding to the position in input job sequence. In order to deal with sequence inputs, transformer [15], with more simple internal networks and the characteristic of parallel training, is utilized to extract production features. Without the needs of mapping output sequences and positions among jobs, the transformer used in this paper removes the decoder and the position encoding part of encoder. The network contains three same attention layers, and one layer is shown in the green box of Fig.2. An attention layer is composed of two sub-layers, a multi-headed self-attentive (MHA) and a feed-forward neural network (FF) respectively. A residual connection is used to connect each layer and batch normalization is utilized to address the output of each sub-layer. In this paper, we use 16 heads and 128 output units in each layer.

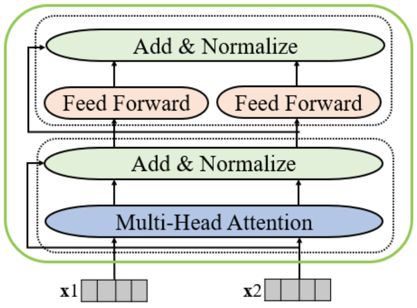

Fig.2. One attention layer in transformer

The actor network is designed to select the next processed job, that is to compute the probability distribution of outputs corresponding to positions of input job sequence. The attention mechanism (AM), originally used for machine translations, is utilized to assign attention to input jobs, then the job with the most attention will be processed next. The AM used in this paper is from Vinyals' work [13], which increases the information flowing to the attention model by adding an additional neural network. The output of the above transformer is defined as $(e_1, e_2, \cdots, e_n)$, and the attention vector is calculated by Eq. (6-8).

$$u_j = v^T \tanh(We_j + d), j \in (1, \cdots, n) \tag{6}$$

$$a_j = softmax(u_j), j \in (1, \cdots, n) \tag{7}$$

$$d' = \sum_{j=1}^{n} a_j^i e_j \tag{8}$$

where softmax function normalizes the vector u (length of job num n) to be the "attention" mask over the inputs, and v, W and d are learnable parameters of the model. Lastly, the attention vector d' will be sent to the following Eq. (9-10), which is similar to the attention model, to calculate the probability distribution P of the input job sequence J.

$$u_j = v^T \tanh(W_1 e_j + W_2 d'), j \in (1, \cdots, n) \tag{9}$$

$$P = softmax(u) \tag{10}$$

where softmax function normalizes the vector u to be the probability distribution P, and u, W_1 and W_2 are learnable parameters. Here, we still use e_j to propagate information to help attention vector d' to point at the job J_i by $softmax(P)$, which is the next job to be processed.

C. Optimization with PPO

Proximal policy optimization (PPO) [16] is a policy gradient algorithm, which is to learn the policy p_θ to maximize the total rewards of an episode $R(\tau) = \sum_{t=0}^{T} r_t$. PPO model is trained based on actor-critic (AC) structure, where the actor is to learn the policy of actions and critic is to estimate the expected reward. However, AC has a low data utilization rate and is sensitive to the length of steps of an episode. PPO adds importance sampling $r(\theta)$ to describe the difference between the old and new policies, thus reusing the data for a certain number of times. Moreover, clip function is used to limit the difference between the old and new policies. In PPO2, only one loss function for training if actor and critic share network parameters, and the loss function is defined in Eq. (13).

$$L_{actor}(s,a,\theta) = \max(A^{p\theta}(s,a)p_\theta(a \mid s)) \tag{11}$$

$$L_{critic}(s,a,\theta) = mse(A^{p\theta}(s,a)) \tag{12}$$

$$L_{PPO} = L_{actor} - c_1 L_{critic} + c_2 S[p_\theta](s) \tag{13}$$

where $A^{p\theta}(s,a)$ is the advantage function and is defined as $A_t^{p\theta}(s_t,a_t) = r_t + \gamma V^\theta(s_{t+1}) - V^\theta(s_t)$, and γ is a discount factor to discount future reward to the present. The advantage function shows the advantage of the reward obtained from the current action relative to the average reward. The rate of the old and new policies $r_t(\theta) = \dfrac{p_\theta(a_t \mid s_t)}{p_{\theta_k}(a_t \mid s_t)}$ is used to represent the importance sampling, and p_{θ_k} is the policy to sample from environment and p_θ is the policy to update network parameters. c_1, c_2 are coefficients, and S denotes an entropy bonus. The pseudo-code of PPO for JSSP is shown in Algorithm 1.

Algorithm 1 The PPO-based training method

```
1: Initialize buffer B with capacity C
2: Initialize PPO network with random weights $\theta$
3: Initialize old PPO network with weights $\theta_{old} = \theta$
4: for iteration= $1, 2, ..., L$ do
5:     Reset JSS environment with the initial jobs $n$ and machines $m$
6:     for $t = 0 : T$ do
7:         Update the JSS enviroment, and generate initial state $s_t$
8:         Run policy $p_{\theta_{old}}$ to take action $a_t$ based on current state $s_t$, and then get reward $r_t$ from enviroment for $a_t$
9:         Store $(s_t, a_t, r_t)$ to buffer
10:        if c>C(c is the number of items in buffer) then
11:            for epoch= $1, 2, ..., K$ do
12:                Sample mini-batch from the buffer
13:                Compute advantage estimates $A^p_\theta$
14:                Compute actor loss $L_actor$, critic loss $L_critic$ and entropy bonus $S[p_\theta](s)$
15:                Perform a gradient descent step on $(L_actor - c_1 L_critic + c_2 S[p_\theta](s))$ with respect to the parameter $\theta$ of PPO network
16:            end for
17:            Update the old PPO $\theta_{old}$ to $\theta$
18:            Clear the buffer
19:        end if
20:    end for
21: end for
```

IV. Numerical Experiments

A. Training process

In this study, the above network model is used as actor network. The critic network shares the transformer model with the actor to extract production features. The outputs of transformer for the critic flow into two fully connected layers with the hidden units (128,1). The parameter setting for training process is presented in TABLE I.

TABLE I. PARAMETER SETTING

Parameter	Value
Learning rate	10^{-4}
Discount factor γ	0.993
Clip range	**0.1**
Critic loss coefficient c_1	985
Entropy coefficient c_2	1228
Batch size	**2048**
Number of training epochs per update	80

The data used for training is generated randomly in a certain range, that the number of jobs n is among [5,20] and the number of machines m belongs to [5,10], and processing time of individual machines is a uniformly distributed integer between [0,100]. To reduce the magnitude of the input processing time, the input of time part is divided by 100. As described above for the rewards, PPO is to train the actor to minimize the total reward of an episode. However, the training instances for different production scales makes it unreasonable to use total reward directly to show the training process. The scaled total reward, divided by the its scale, is used to show the learning process of network model. The scale is defined as the number of operations nxm in this study. The average scaled total rewards of per 50 episodes in the first 1000 episodes are shown in Fig. 3. It can be seen that the curve of average scaled episode rewards increases smoothly with the increase of training episodes and stabilizes after 700 episodes, which indicating that the proposed end-to-end DRL method is convergent and has learnt the proper scheduling policy for JSSPs.

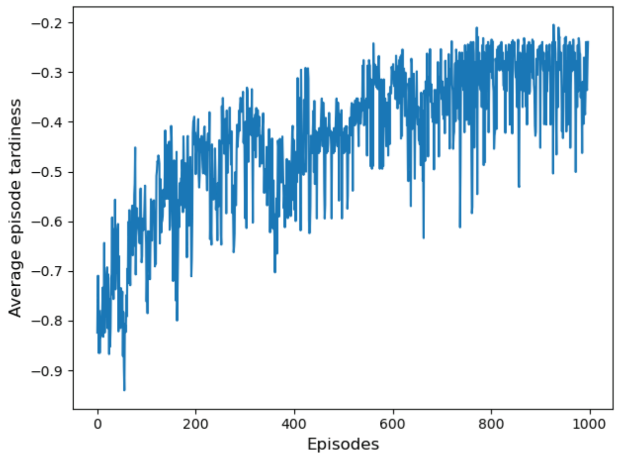

Fig.3. Training process

In order to further explore the global performance of the proposed DRL method during training, the model is tested on two predefined instances with different scales in OR-library LA06 (15X5) and LA21 (15X10) per 200 episodes, as shown in Fig.4 and Fig.5. The test of LA06 achieves the optimal solution and the test of LA21 obtains a good suboptimal solution, which demonstrates the global performance and generality of the proposed end-to-end DRL method.

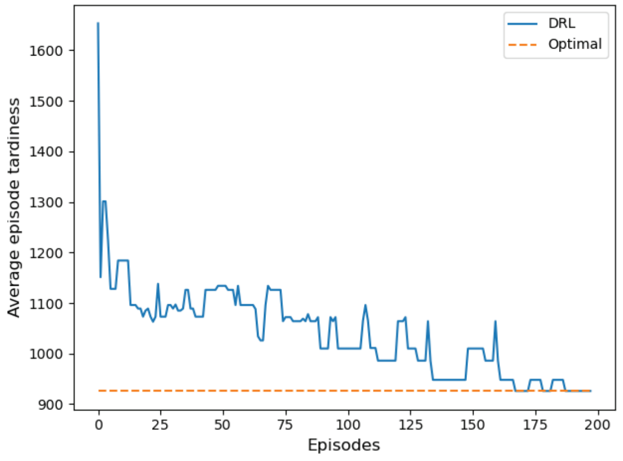

Fig. 4. Test instance LA06 (15x5)

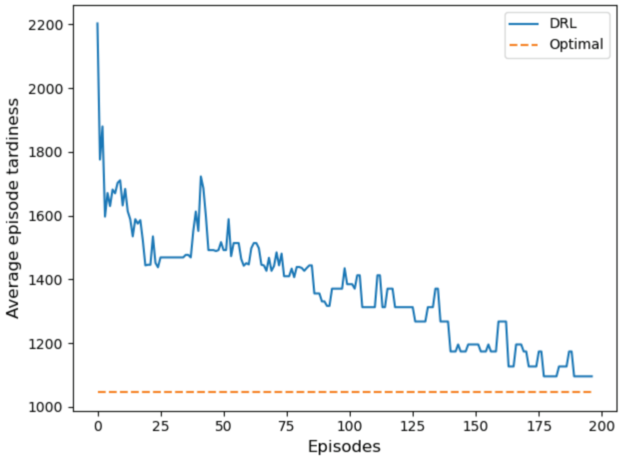

Fig. 5. Test instance LA21(15x10)

B. Experiments results

To verify its effectiveness, the proposed DRL method is compared with three classic heuristic rules (first in first out-FIFO, shortest processing time-SPT, most remaining processing time-MRT). Furthermore, genetic algorithm (GA) with no local search method is also compared. Benchmark instances of different scales in OR-library is utilized for validation. The scheduling results are shown in TABLE II. The optimal solutions are marked in bold and those with '*' indicate that the optimal solutions are obtained by corresponding methods.

TABLE II. EXPERIMENT RESULTS

Instances	**DRL**	**FIFO**	**SPT**	**MRT**	**GA**	**Optimal**
LA01(10x5)	**684**	772	708	755	706	666
LA02(10x5)	**715**	830	723	815	732	655
LA06(15x5)	**926***	**926***	1125	948	948	926
LA07(15x5)	**894**	1088	985	1001	985	890
LA11(20x5)	**1222***	1228	1259	**1222***	1304	1222
LA12(20x5)	**1039***	**1039***	1063	**1039***	1087	1039
LA16(10x10)	**990**	1180	1088	1076	1124	945
LA17(10x10)	**795**	943	925	915	918	784
LA21(15x10)	**1096**	1265	1271	1342	1279	1046
LA22(15x10)	**982**	1312	1151	1157	1298	927
LA26(20x10)	**1284**	1372	1399	1452	1493	1218
LA27(20x10)	**1308**	1644	1478	1519	1572	1235

From the results, the following conclusions can be drawn. For some instances where heuristic rules, like FIFO and MRT, can obtain optimal solutions, DRL also achieves the best, which indicating that it is easy for the DRL agent to learn the optimal scheduling policies under simple production environments. For other instances, the scheduling results of our method are significantly superior to each other rule, because DRL agent can make scheduling decisions by comprehensively observing the current production states. GA is a common method for solving JSP due to its global search. However, GA method combined with the local search strategy designed for specific problems can obtain optimal scheduling results. What's more, it needs to resolve when face different JSP problems, which takes much time on decision making. In general, the proposed end-to-end DRL method in this study guarantees both efficiency and quality of the solution, and has a good generalization.

V. CONCLUSIONS

In this research, an end-to-end DRL method is proposed to address JSSPs, which can be formulated as sequential decision-making problems. DRL can learn offline and make decisions online, thus achieving real-time response to dynamic disturbances and keeping improving the ability of decision-making. Moreover, RL can optimize predefined global objectives based on current production states, due to which it can avoid the short-sightedness of decision-making. Besides, an end-to-end method is utilized in this study to break through the limitations of rules and further improve the quality of solutions. In comparison with other heuristic rules, our method achieves better performance.

In future work, sensitivity experiments of individual hyperparameters will be conducted to further improve the quality of solutions. What's more, the reactive dynamic scheduling, a representative of the sequential decision - making problem, is a more suitable production environment for our method. Therefore, it is worthy to research reactive scheduling in dynamic production environment based on the proposed end-to-end DRL method.

REFERENCES

[1] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshopand jobshop scheduling," *Mathematics of operations research,* vol. 1, no. 2, pp. 117-129, 1976.

[2] M. Dell'Amico and M. Trubian, "Applying tabu search to the job-shop scheduling problem," *Annals of Operations research,* vol. 41, no. 3, pp. 231-252, 1993.

[3] J. F. Gonçalves, J. J. de Magalhães Mendes, and M. c. G. Resende, "A hybrid genetic algorithm for the job shop scheduling problem," *European journal of operational research,* vol. 167, no. 1, pp. 77-95, Nov 2005.

[4] O. Alagoz, H. Hsu, A. J. Schaefer, and M. S. Roberts, "Markov decision processes: a tool for sequential decision making under uncertainty," *Medical Decision Making,* vol. 30, no. 4, pp. 474-483, Dec 2010.

[5] M. Van Otterlo and M. Wiering, "Reinforcement learning and markov decision processes," in *Reinforcement learning*: Springer, 2012, pp. 3-42.

[6] W. Zhang and T. G. Dietterich, "A reinforcement learning approach to job-shop scheduling," in *IJCAI*, 1995, vol. 95, pp. 1114-1120: Citeseer.

[7] M. E. Aydin and E. Öztemel, "Dynamic job-shop scheduling using reinforcement learning agents," *Robotics and Autonomous Systems,* vol. 33, no. 2-3, pp. 169-178, Nov 2000.

[8] T. Gabel and M. Riedmiller, "Adaptive reactive job-shop scheduling with reinforcement learning agents," *International Journal of Information Technology and Intelligent Computing,* vol. 24, no. 4, pp. 14-18, 2008.

[9] J. Shahrabi, M. A. Adibi, and M. Mahootchi, "A reinforcement learning approach to parameter estimation in dynamic job shop scheduling," *Computers & Industrial Engineering,* vol. 110, pp. 75-82, Aug 2017.

[10] B. Cunha, A. M. Madureira, B. Fonseca, and D. Coelho, "Deep reinforcement learning as a job shop scheduling solver: A literature review," in *International Conference on Hybrid Intelligent Systems*, 2018, pp. 350-359: Springer.

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

[12] H. Wang, B. R. Sarker, J. Li, and J. Li, "Adaptive scheduling for assembly job shop with uncertain assembly times based on dual Q-learning," *International Journal of Production Research,* vol. 59, no. 19, pp. 5867-5883, Jul 2021.

[13] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," *arXiv preprint arXiv:1506.03134,* Jun 2015.

[14] R. Pan, X. Dong, and S. Han, "Solving permutation flowshop problem with deep reinforcement learning," in *2020 Prognostics and Health Management Conference (PHM-Besançon)*, May 2020, pp. 349-353: IEEE.

[15] A. Vaswani *et al.*, "Attention is all you need," in *Advances in neural information processing systems*, Dec 2017, pp. 5998-6008.

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