IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 23, 2020, accepted October 6, 2020, date of publication October 9, 2020, date of current version October 22, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3029868

Research on Adaptive Job Shop Scheduling Problems Based on Dueling Double DQN

BAO-AN HAN AND JIAN-JUN YANG
Department of Industrial and Manufacturing Systems Engineering, Beihang University, Beijing 100191, China

Corresponding author: Bao-An Han (hanbaoan@buaa.edu.cn)

This work was supported in part by the Beijing Key Laboratory of Digital Design and Manufacturing; in part by the National High Technology Research and Development Program(863) of China under Grant 2012AA040907; and in part by the postgraduate innovation practice base of the modern design and advanced manufacturing technology for complex products in Beihang University.

ABSTRACT Traditional approaches for job shop scheduling problems are ill-suited to deal with complex and changeable production environments due to their limited real-time responsiveness. Based on disjunctive graph dispatching, this work proposes a deep reinforcement learning (DRL) framework, that combines the advantages of real-time response and flexibility of a deep convolutional neural network (CNN) and reinforcement learning (RL), and learns behavior strategies directly according to the input manufacturing states, thus is more appropriate for practical order-oriented manufacturing problems. In this framework, a scheduling process using a disjunction graph is viewed as a multi-stage sequential decision-making problem and a deep CNN is used to approximate the state-action value. The manufacturing states are expressed as multi-channel images and input into the network. Various heuristic rules are used as available actions. By adopting the dueling double Deep Q-network with prioritized replay (DDDQNPR), the RL agent continually interacts with the scheduling environment through trial and error to obtain the best policy of combined actions for each decision step. Static computational experiments are performed on 85 JSSP instances from the well-known OR-Library. The results indicate that the proposed algorithm can obtain optimal solutions for small scale problems, and performs better than any single heuristic rule for large scale problems, with performances comparable to genetic algorithms. To prove the generalization and robustness of our algorithm, the instances with random initial states are used as validation sets during training to select the model with the best generalization ability, and then the performance of the trained policy on scheduling instances with different initial states is tested. The results show that the agent is able to get better solutions adaptively. Meanwhile, some studies on dynamic instances with random processing time are performed and experiment results indicate that out method can achieve comparable performances in dynamic environment in the short run.

INDEX TERMS Adaptive scheduling, convolutional neural network, deep reinforcement learning, dueling double DQN, job shop scheduling problem (JSSP).

I. INTRODUCTION

With economic globalization, manufacturing enterprises are facing fierce market competition and unpredictable production environment changes. At the same time, information technologies such as cloud computing [1], Internet of things [2], big data technology [3], [4] and other technologies, have become key to advanced manufacturing. Cyber-Physical Systems (CPS) [5], Industry 4.0 [6] and Made in China 2025 [7] strongly promote the transformation and upgrading of China's manufacturing industries. Production scheduling, which plays a core role in the manufacturing process, is a particularly important aspect this advancement. The ability to locate potential knowledge, improve response speed, and promote self-learning and sustainable development of scheduling system are important issues that need to be solved urgently.

In the past few decades, there has been a lot of research on job shop scheduling, but most of them are for standard problems in static environments. Along with the development of data acquisition hardware, and the advancement of data processing technology, dynamic events in the workshop

The associate editor coordinating the review of this manuscript and approving it for publication was Nagarajan Raghavan.

can be captured in real time, which puts forward a higher requirement for the real-time response of job shop scheduling. Accordingly, the scheduling system should deal with all kinds of variabilities in production requirements, processing times and available machines, making it challenging to obtain high-quality schedules within the time limit for large-scale job shop scheduling.

At present, the most commonly used approximate algorithms in the scheduling field are heuristic [8], [9], meta-heuristic [10], [11] and other approximation algorithms [12]. Meta-heuristic algorithms provide higher quality solutions, but are infeasible for real-time scheduling, especially when solving large-scale problems, because the solution time increases exponentially and is strongly related to the problem structure. If the problem structure changes, the meta-heuristic algorithms need to be redesigned with poor generalization. In this case, heuristic algorithms are more practical, and are widely used because of their simplicity and rapidity. However, their solution quality is not ideal and their performance varies greatly depending on the problems and the objectives [8].

Various studies that addressed scheduling problems based on machine learning have been carried out to overcome the drawback of rule-based methods [13]. Foo and Takefuji [14] first proposed to use Hopfield model to solve job shop scheduling problem, and then a large number of researchers conducted improvement studies. Remus and Hill [15] was the first to use backpropagation (BP) model for scheduling. Akyol and Bayhan [16] gave a comprehensive overview on artificial neural network (ANN) approaches for solution of production scheduling problems, discussed both theoretical developments and practical experiences, and identified research trends. Weckman *et al.* [17] used a NN to capture the predictive knowledge regarding the assignment of operation's position in a sequence generated by genetic algorithm (GA). Lim *et al.* [18] proposed a case-based reasoning approach to find the best decision for the state expressed in a Petri net. At present, this approach can only solve small-scale scheduling problems, and it cannot effectively obtain high-quality scheduling that are required as training data sets, so it may not be satisfactory in real-world scheduling systems.

Reinforcement learning [19] is a learning and decision-making algorithm directly oriented to long-term goals based on state or state-action pair. It does not require a complete mathematical model of the learning environment, but can imitate human experience, learn and accumulate experience from previous examples or simulation experiments, and adjust the learning strategy through the immediate reward obtained through the interaction with environment, to optimally respond to different system states. At the same time, learned knowledge can be stored for reuse to achieve ''offline learning and online application''. As mentioned above, heuristic rules have variable scheduling performances for different problems. Are there methods that can lead to adaptive selections? Naturally, heuristic rules can be regarded as optional actions of RL. Each time an operation is scheduled according to the action, the scheduling system enters a new state, and an immediate reward is given as the evaluation. RL-based scheduling rule selection enables the agent to find an optimal strategy π^*, and to maximize the expected cumulative return when starting in any state and following the scheduling strategy. This optimal strategy is inherently a set of rules executed sequentially, and its scheduling result is not worse than that of any single rule.

The contributions of this work are as follows. (1) We proposed a double Deep Q-network (DDQN) model to construct a deep reinforcement learning framework for scheduling problems, which includes a target network and an online network to deal with the overestimation problem that exists in general DQN. In this framework, both static and dynamic instances can be optimized. (2) An RL environment based on disjunctive graph was established for the first time, and the process of a scheduling solution based on disjunctive graph was transformed into a sequential decision process which had not been tried in previous studies. In this environment, scheduling may start from a non-zero state, that is, some operations could be interactively dispatched first and then the remaining operations were scheduled by an algorithm. (3) At each discrete time step, the scheduling state was creatively represented as a multi-channel image to avoid handcrafted features as used in traditional RL. According to the input state, a CNN selected one heuristic rule to determine the job with the highest priority from the current set of schedulable tasks. (4) A novel reward function equivalent to the $C_{\max}$ was designed to evaluate the impact of each dispatch on scheduling objective. (5) An improved epsilon-decreasing strategy considering an elitist mechanism was proposed, which would select the optimal rule of the current best solution with a certain probability at the later stage of training. Experimental results showed that the scheduling performance was improved by 5.92% on average in all instances. (6) A large number of experiments have been carried out to analyze the sensitivity of different hyperparameters and verify the effectiveness on static problems and its generalization on dynamic problems with reactive scheduling and uncertain processing time.

To evaluate the proposed algorithm, the DRL agent was trained on several job shop scheduling problems (JSSPs) benchmarks from the OR-Library, where each benchmark was formulated as a decision-making problem and its corresponding RL environment was built based on disjunctive graph. In order to determine the optimal hyperparameters, the sensitivity of different parameters to the verification results was analyzed. The advantage of the proposed algorithm lies in that, once the off-line training process is over, the optimal strategy can be directly used online to obtain scheduling results with a running time approximately equal to that of simple rules, which can be equivalent to or even better than the performance of the GA, so as to ensure the scheduling accuracy and improve the solving efficiency. Different from the meta-heuristic algorithms, high quality solutions can be obtained without time-consuming iterative optimization

when new problems are encountered or the original problem structure changes. The static experimental results show that the scheduling results of this algorithm are not worse than any heuristic rule in all instances, with an average performance improvement of 11.56%, and the scheduling performance in only two instances is slightly worse than that of the two GAs, but the overall average performance is improved by 5.6% and 6.17% respectively. In addition to the strong optimization ability, the generalization and robustness of this method are more significant. The instances with random initial states were used as validation sets during training to select the model with the best generalization ability, and then the performance of the trained policy on scheduling instances with different initial states was tested. The results show that the agent has good generalization ability on unknown situations. At the same time, random instances with disturbance information were tested in dynamic environment, and the results show that this method can still obtain robust solutions adaptively in dynamic environment.

The rest of this paper is organized as follows. Section II presents related surveys and techniques. In section III, an adaptive scheduling framework that combines values-based DRL and disjunctive graph is proposed. Section IV describes the job shop scheduling model based on disjunctive graph to establish RL environment. Section V introduces the proposed DRL scheduling algorithm and describes how to transform scheduling problems into Markov or semi-Markov decision problems. Section VI shows the experimental results both in static and dynamic environments. The conclusions and future work are presented in Section VII.

II. LITERATURE REVIEW

This work mainly focuses on adaptive JSSP, and proposes to use DRL to cope with this problem. Thus, this section briefly introduces adaptive scheduling and the application of RL and DRL in production scheduling. A summary of studies involved is provided in Table 1.

A. ADAPTIVE SCHEDULING

A scheduling strategy that can make decisions dynamically with different system states is called adaptive scheduling strategy, and the scheduling method based on this strategy is called adaptive scheduling [9]. The essence of adaptive scheduling is to select the most suitable scheduling rules according to the optimization objective and system state in the production environment to ensure optimal or sub-optimal scheduling performance, in order to deal with the unexpected events that may happen in the job shop, such as variations in process times and machine breaking down. Rescheduling [20], [21], real-time scheduling [22], [23], online scheduling [24], [25] and random scheduling [26] are all types of adaptive scheduling.

In recent years, the adaptive production scheduling problem with an uncertain environment has been an active research area [27]–[32]. Yang and Wang [33] presented a new adaptive neural network and heuristics hybrid approach for job-shop scheduling. Aiming at the job-shop scheduling problem with bottlenecks, Varela *et al.* [34] designed a probabilistic-based heuristic method to guide backtracking search. To balance efficiency and stability, Rangsaritratsamee *et al.* [35] proposed a rescheduling methodology to generate schedules at each rescheduling point using a genetic local search algorithm. Lee [36] proposed an adaptive scheduling system based on fuzzy rules. Kundakci and Kulak [37] summarized the dynamic events in dynamic job shop scheduling and introduced hybrid GA to minimize makespan. Ning *et al.* [38] proposed an improved hybrid multi-phase quantum particle swarm algorithm to solve the dynamic scheduling of flexible job-shop problems. To deal with the dynamic flexible job shop scheduling problem considering machine failure, urgent job arrival, and job damage as disruptions, Wang *et al.* [39] adopted a modified GA to construct rescheduling strategy. Xiong *et al.* [40] simulated and analyzed dispatching rules for dynamic job shop scheduling. To achieve the real-time data-driven optimization decision, Zhang *et al.* [41] proposed a dynamic optimization model for flexible job shop scheduling based on game theory. Each machine actively requests processing tasks and the processing tasks will choose the optimal machines according to their real-time state. Piroozfard *et al.* [42] proposed an improved multi-objective GA to minimize the total carbon footprint and total late work. For distributed JSSP, Chaouch *et al.* [43] used a hybrid ant colony algorithm combined with a local search to minimize the global makespan over all the factories.

B. REINFORCEMENT LEARNING FOR SCHEDULING

RL does not search directly in the solution space, but fully reflects the characteristics of sequential decision problem through state or action value. In this way, the structure of a decision problem is fully utilized to search a strategy while the full exploration of neighborhood search algorithm using random mechanisms is reserved. This feature is expected to improve search effectiveness in solving large-scale problems. RL has been widely used in the field of scheduling. Aydin and Öztemel [44] used the Q-III learning algorithm for training an agent to dynamically select job shop dynamic scheduling rules, to minimize average tardiness. Experiments showed that in most cases the learned scheduling results were better than any rule in SPT, EDD and COVERT. Wang and Usher [45], [46] applied Q-learning to select dispatching rules for single machine scheduling problem with different objectives. Paternina-Arboleda and Das [47] obtained a dynamic control policy for stochastic lot-scheduling problem, which optimized the WIP inventory, the backorder penalty costs and the setup costs, while meeting the productivity constraints for the products. All these results showed the potential of RL in production scheduling, but they only focused on single machine scheduling.

To deal with multi-machine scheduling problems, Csaji and Monostori [48] studied a stochastic production scheduling problem with non-identical parallel machines. They

TABLE 1. Summary of relevant studies.

References	Type of problem	Number of objectives	Number of machines	Approach
[33]	Job shop scheduling	S	M	Adaptive neural network and heuristics hybrid approach
[34]	Job shop scheduling	S	M	Genetic algorithm
[35]	Dynamic job shop scheduling	M	M	Genetic local search algorithm
[36]	Flexible manufacturing system	S	M	Fuzzy rule-based system for an adaptive scheduling
[37]	Job shop scheduling	S	M	Hybrid genetic algorithm
[38]	Dynamic flexible job shop scheduling	M	M	Improved hybrid multi-phase quantum particle swarm algorithm
[39]	Dynamic flexible job shop scheduling	S	M	Improved genetic algorithm
[40]	Dynamic job shop scheduling	S	M	Four new proposed dispatching rules
[41]	Flexible job shop scheduling	S	M	Game theory
[42]	Flexible job shop scheduling	M	M	Improved multi-objective genetic algorithm
[43]	Distributed job shop scheduling	S	M	hybrid ant colony algorithm combined with local search
[44]	Dynamic job shop scheduling	S	S	Q-III
[45, 46]	Job shop scheduling	S	S	Q-learning to the single machine dispatching rule selection
[47]	Stochastic lot-scheduling	S	S	Multi-agent RL approach
[48]	Stochastic production scheduling	S	M	Homogeneous multi-agent system
[49]	Dynamic unrelated parallel machine scheduling	S	M	Q-learning
[50]	Unrelated parallel machine scheduling	S	M	R-learning
[51]	Dynamic job shop scheduling	S	M	Scheduling control policy based on B-Q learning algorithm
[52]	Dynamic job-shop scheduling	S	M	Q-learning
[53]	Dynamic job-shop scheduling	S	M	Centralized RL approach
[54]	Dynamic job shop scheduling	S	M	RL
[55]	Knowledgeable manufacturing system	S	M	Weighted Q-learning
[56]	Dynamic job shop scheduling	S	M	Q-factor algorithm
[57]	Dynamic job shop scheduling	S	M	Weighted Q-learning algorithm
[64]	Socio-technical Manufacturing System	S	M	Deep Q-learning
[66]	Semiconductor production scheduling	S	M	Deep Q-network
[67]	Job shop scheduling	S	M	Deep Q-network
[65]	Socio-technical manufacturing system	S	M	Deep Q-Network
[68]	Job shop scheduling	S	M	Actor-Critic DRL

M denotes multi, and S denotes single

expressed the scheduling problem as a Markov decision-making process, and calculated a near-optimal control policy through multi-agents. Zhang *et al.* [49], [50] investigated unrelated parallel machine scheduling with Q and R learning in order to minimize the mean weighted tardiness. Yang and Yan [51] proposed an improved Q-learning algorithm to obtain an adaptive scheduling control strategy. Wei *et al.* [52] used RL to select composite rules for dynamic job shop scheduling to minimize mean system tardiness. Qu *et al.* [53] developed a centralized RL approach for scheduling a manufacturing system of multi-state processes and multiple machines from a data-driven perspective. They [54] also proposed a scheduling method based on RL, which can adaptively update production plans by using real-time product and process events information during executions. Wang and Yan [55] designed a knowledgeable dynamic schedule based on multi-agents, and put forward an adaptive scheduling strategy based on weighted Q-learning. Shahrabi *et al.* [56] adopted a Q-factor algorithm for parameter estimation to improve the scheduling performance of dynamic JSSPs which considered random job arrivals and machine breakdowns. In order to adapt to changes of various production factors in dynamic and uncertain job shops, Wang [57] constructed a dynamic scheduling system model based on multi-agents, and adopted a weighted Q-learning based on clustering and dynamic search to optimize production.

C. DRL FOR SCHEDULING

Traditional RL is often used in small-scale problems with a discrete state space. Nevertheless, the state space faced by real RL tasks is often continuous, with an infinite number of states, such as operation slack and machine utilization in scheduling problem, which are all continuous values. In this case, it is necessary to consider approximating the value function. DRL [58], [59] is a new algorithm which combines deep learning with RL and realizes the end-to-end learning from perception to action. In brief, just like human beings, it can directly output actions through the deep neural network by inputted sensory information, such as vision without artificial feature extraction. DRL has the potential to enable agents to learn one or more skills completely and autonomously. Its purpose is to use a deep neural network to automatically learn the characteristics of dynamic scenes and then to use RL to learn the decision action sequence corresponding to the scene features. With the extensive application of DRL in

games [60], image processing [61], video [62], natural language [63], etc., it has gradually attracted prominence in the field of scheduling. Palombarini and Martinez [64] adopted a DRL approach which stored the rescheduling knowledge in a deep Q-network to learn schedule repairing policies directly from high-dimensional sensory inputs. After that, they [65] modelled the real-time rescheduling task as a closed-loop control problem and trained a deep Q-network to select repair actions in response to unexpected events and disturbances. Waschneck *et al.* [66] applied a deep Q-network to semiconductor manufacturing scheduling and trained a deep neural network with flexible user-defined objectives. Lin *et al.* [67] proposed an edge computing-based smart manufacturing factory framework, based on which the DQN was adjusted to solve the JSP. Liu *et al.* [68] viewed the JSSP as a sequential decision making problem and proposed to adopt DRL to deal with it. To speed up model training, they also proposed a parallel training method which combined asynchronous updates with deep deterministic policy gradient.

III. DRL SCHEDULING FRAMEWORK

Our contribution is to propose an adaptive scheduling framework that combines values-based DRL and disjunctive graph. The scheduling problem is transformed into a sequential decision-making problem by defining the environment, state, action, reward, and strategy. The DDDQNPR is then used to train the scheduling policy offline whenever an operation is assigned to a machine. Finally, the optimal scheduling policy can be used for online solving of different problems to obtain the optimal solutions. The proposed framework includes three parts: scheduling environment, offline learning, and online application (Fig. 1).

The scheduling environment is modelled with disjunctive graph. Compared with the simulation model used in most literatures, the principle of disjunctive graph model is not in the way of simulation clock advance, but divides discreate time step from the perspective of operation assignment. Thus, it can be convenient to add interactive constraints, so as to make the scheduling process closer to the actual production environment. Disjunctive graph-based scheduling solution is to first initialize the ready task set, from which the job with the highest priority is dispatched to machine. Then the job is removed from the constraint network and its subsequent task is added in the ready task set. This process is repeated until the ready task set is empty and the scheduling result is obtained. Such a complete process is called an episode, so the length of an episode is the total number of operations. At the same time, in order to verify that the process is generalizable and adaptive, dynamic events are also considered in the scheduling environment. When the scheduling events occur, the scheduling problem changes and rescheduling is required. The policy obtained from the training with the DRL algorithm can intelligently deal with these accidents.

In the offline learning phase, the DDDQNPR is adopted to train the samples generated during each allocation in the scheduling environment. The samples are first stored in the replay memory and importance sampling is performed based on TD errors with a minibatch. On one hand the sampled transitions are input in the online network Q to estimate its Q value, on the other hand, they are also input in the target network $\hat{Q}$ to calculate the target value. Stochastic gradient descent is applied to the loss function between estimated Q value and target Q value to update the parameters of the online network Q. The parameters of the target network are copied every C steps from the online network, and kept fixed on all other steps. After obtaining the estimated Q value, based on the exploration and exploitation strategy such as the ε-greedy strategy, the scheduling action, i.e. heuristic rule is selected to dispatch task.

Although it takes a long time to train during the learning phase, once the optimal strategy is learned, it can be applied to new scheduling problems, and optimal results can be obtained in a short time. In the online application phase, the first step is to synchronize dynamic scheduling events and update job and machine states, which are mapped into the optimal rule with the optimal policy. At the next time step a new job will be scheduled based on new states until the ready task set is empty.

IV. SCHEDULING ENVIRONMENT SETUP

The formulation of a JSSP is given first, where lowercase letters represent indexes, and bold uppercase letters represent sets. Suppose there are NJ jobs and NM machines in the job shop scheduling problem Each job j_i contains an operation set $\boldsymbol{Oi}$ consisting of multiple operations Oih. The job has a predetermined processing order, and the uninterrupted processing time of each operation is Pih. Each machine can process only one job at a time, and each job can be processed by only one machine at the same time. Precedence constraints between different jobs do not exist. A schedule is an allocation of all the operations to time intervals on the machines. The optimization problem is to find a feasible schedule of minimum length.

In the classical job shop scheduling problem, the solution is usually expressed by a disjunctive graph model [69], which is a directed graph $G = (V, C \cup D)$. V denotes a set of vertices corresponding to all operations of jobs. This set contains two virtual vertices: a source and a sink, which represent the start and the end of a schedule. Both dummy tasks have zero processing time. C is a set of conjunctive arcs which represent the precedence constraints between every two consecutive operations of the same job determined by process. D consists of undirected disjunctive edges connecting mutually unordered tasks which can be executed on the same machine. A simple disjunctive graph is shown in Fig. 2, where each circle represents an operation, its content is the operation name, and the weight on each arc equals to processing time of the operation where the arc begins. For example, O_{11} is the first operation of job one. Operation vertices with the same color indicate that the operations are processed on the same machine. The solid and dotted arrows represent conjunctive and disjunctive arcs respectively.

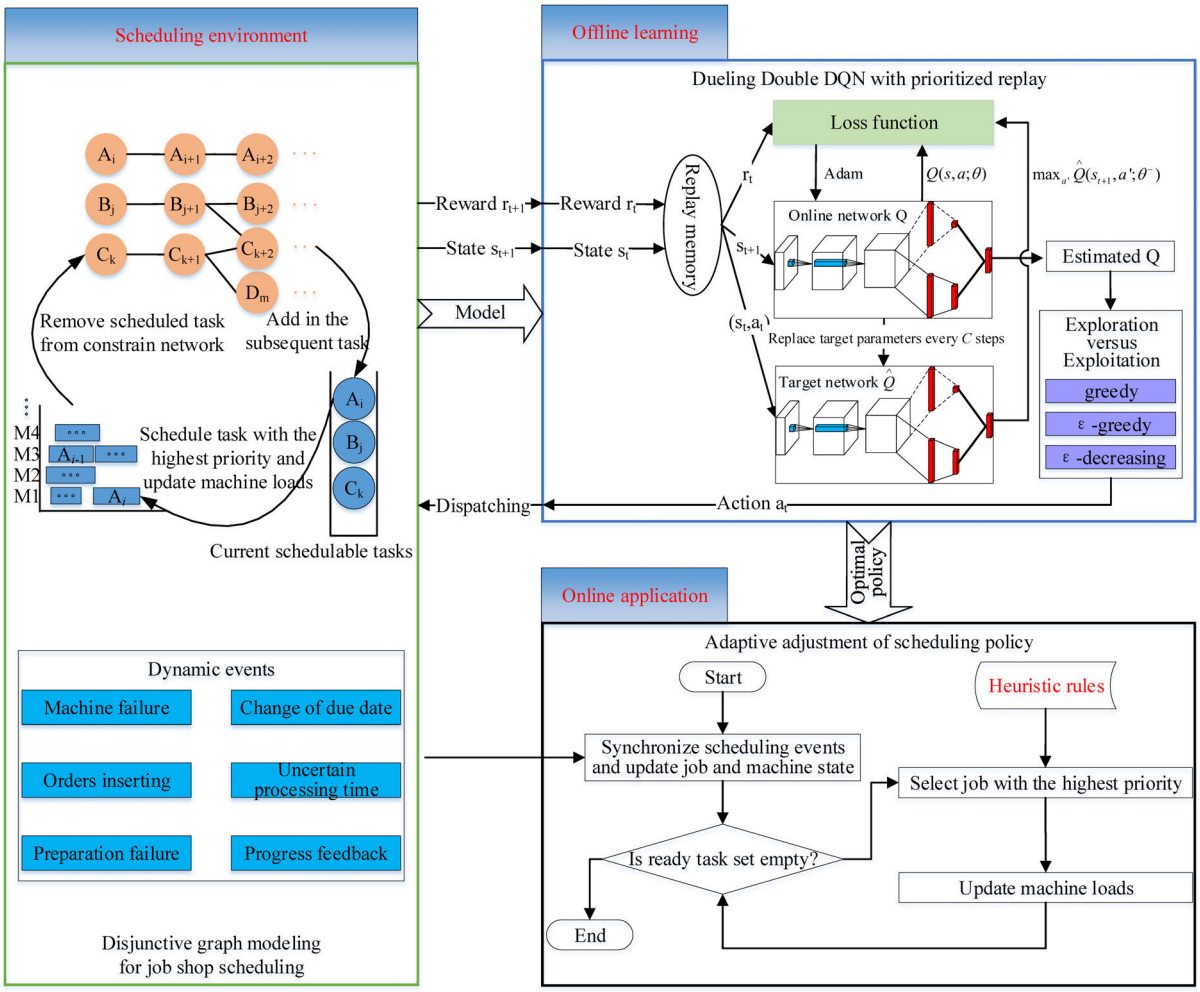


FIGURE 1. Scheduling framework with DRL including scheduling environment, offline learning and online application.

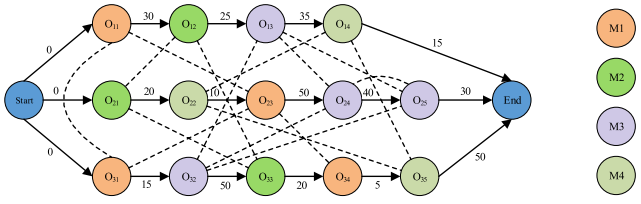


FIGURE 2. Disjunctive graph.

The disjunctive graph shown in Fig. 2 is in unsolved state. The scheduling process based on a disjunctive graph is to turn each undirected disjunctive edge into a directed conjunctive one. By picking directions for all disjunctive edges, the processed order of all competitive tasks requiring the same machine is determined and a feasible schedule is obtained if the resulting graph is acyclic. A directed acyclic graph is shown in Fig. 3.

After obtaining a directed acyclic graph that can describe a feasible solution, we need a more intuitive way to arrange all the vertices in G into a linear sequence through topological sorting, so that any pair of vertices in the directed acyclic graph has a clear sequence. Topological sorting is obviously not unique, but the final scheduling results corresponding to all topological sorts of a directed acyclic graph will be the same. Fig. 4 shows the topological sorting result of Fig. 3.

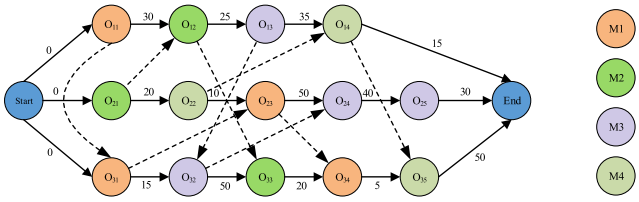


FIGURE 3. Disjunctive graph instantiation.

Therefore, to express the solution of scheduling problems through a disjunction graph is to determine the order of each operation subject to the precedence constraints and capability constraints, which is essentially a sequential decision problem and can be solved by RL, so combining disjunctive graph and DRL is an effective and direct way, and it has not been tried in previous studies. In the next section we will describe in detail how to express a scheduling problem as a RL problem and how to solve it.

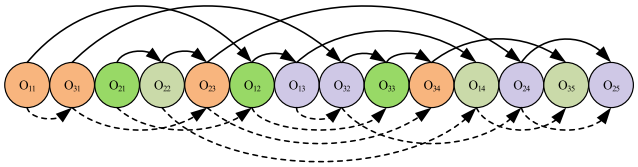


FIGURE 4. Topological sorting.

V. DRL FOR SCHEDULING

A. FUNDAMENTALS OF RL AND DRL

The Markov property refers to the fact that the next system state is only related to the current state and has no relation to the previous state. The multi-stage decision problem with state transition satisfying the Markov property is a Markov decision processes (MDPs). MDPs can be described as a 5-tuple as follows:

$$\mathcal{E} = \left\{Z^+, S, \mathcal{A}(s), P, R\right\} \tag{1}$$

where the agent is in environment $\mathcal{E}$, $Z^+ = \{0, 1, 2, \cdots\}$ denotes a collection of decision stages, each $s \in S$ in state space S describes the environment's state, $\mathcal{A}$ is the action space and denotes a set of all actions available in state s, P represents probability distributions of state transitions, $P(s, a, s')$ and $R(s, a, s')$ are the probability of transition to state s' and immediate reward respectively, from state s taking action a.

In standard RL, an agent interacts with environment $\mathcal{E}$ at multiple discrete time steps. At each time step t, the agent perceives the environment's state s_t and selects an action a_t from all possible actions $\mathcal{A}$ following its policy π. Afterthat, the agent perceives the next state s_{t+1} and receives a scalar reward r_t. The above process is repeated until the agent reaches a terminal state. The return $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ is the total accumulated reward from time step t with discount factor $\gamma \in (0, 1]$. The agent's goal is to maximize the expected return from each state s_t.

The action value $Q^\pi(s, a) = \mathbb{E}[R_t | s_t = s, a] = \mathbb{E}_{s',a'}\left[r + \gamma Q^\pi(s', a') | s, a\right]$ is the expected return for selecting action a in state s and following policy π. The second equation is the Bellman equation, describing the iterative form of the action value. The optimal value function $Q^*(s, a) = \max_\pi Q^\pi(s, a)$ gives the maximum action value for state s and action a under any policy. Once Q^* is learned the optimal policy $\pi^*(s) = \arg\max_a Q^*(s, a)$ is obtained. The optimal action value satisfies the following Bellman optimality equation:

$$Q^*(s, a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s', a') | s, a\right] \tag{2}$$

It has been proved that the action value function converges to Q^* with tabular solution methods. However, when solving problems with large state spaces, the difficulty is not just the memory needed for large tables, but also the time and data needed to fill them accurately if each state is stored in a table. In value-based model-free RL methods, the action value function is represented with a function approximator, such as a neural network. Let $Q(s, a; \theta)$ be an approximate action-value function with parameters θ, the updates to which can be derived from various RL algorithms. A common example of this type of algorithms is Q-learning, whose goal is to directly approximate the optimal action value function $Q^*(s, a) \approx Q(s, a; \theta)$. In one-step Q-learning, the parameters θ are updated by iteratively minimizing a sequence of loss functions, where the ith loss function is defined as:

$$L_i(\theta_i) = \mathbb{E}\left(r + \gamma \max_{a'} Q\left(s', a'; \theta_{i-1}\right) - Q(s, a; \theta_i)\right)^2 \tag{3}$$

where s' is the subsequent state of s.

When a neural network is used for an action-value (also known as Q) function approximation, RL is likely to be unstable or even to diverge. This instability can be caused by several factors: the correlations between sampled transitions, the fact that small updates to Q may significantly change the policy and therefore change the data distribution, the interdependence between the Q updates and the target values $r + \gamma \max_{a'} Q\left(s', a'\right)$ calculation. To address these instabilities, DQN introduced experience replay and a second network to randomize over the data and fix target values for some steps, thereby removing correlations in the observation sequence and stabilizing the training process. The target is updated by:

$$Y_t^{\text{DQN}} \equiv r_{t+1} + \gamma \max_a \hat{Q}\left(s_{t+1}, a; \theta_t^-\right) \tag{4}$$

Although the max operator used in the above formula can quickly bring the Q values closer to possible targets, it can be easily overestimated because the same values are used to select and to evaluate an action. To avoid this, DDQN decouples the selection from evaluation. Although not completely decoupled, the target network in the DQN architecture can be used for value function evaluation without introducing additional networks. Therefore, the online network is used to evaluate the greedy strategy, while the target network is used to evaluate its value. The update target for DDQN is as follows:

$$\begin{aligned} Y_t^{\text{DoubleDQN}} &\equiv r_{t+1} \\ &\quad + \gamma \hat{Q}\left(s_{t+1}, \arg\max_a Q\left(s_{t+1}, a; \theta_t\right), \theta_t^-\right) \end{aligned} \tag{5}$$

DQN and DDQN both use experience replay, which samples uniformly from replay memory. However, different samples have different TD errors. The greater the TD error, the greater the influence on the backpropagation, and vice versa. In Q-network, the TD error is the difference between the target Q value of the target network and the estimated Q of the online network. Prioritized Replay DQN calculates each sample's priority based on its TD error, and the loss function considering sample priority is defined as:

$$L_i(\theta_i) = \mathbb{E}\left(\omega_i\left(Y_t^{\text{DoubleDQN}} - Q(s, a; \theta_i)\right)\right)^2 \tag{6}$$

where ω_i is a scaled weight.

In Prioritized Replay DQN, the algorithm is optimized by calculating the samples' priority, whereas Dueling DQN

tries to optimize the algorithm by changing the structure of the neural network. Two subnetwork structures are added before the output layer, and used to learn value function and advantage function. The model's output combines the two to produce a state-action value Q (s; a) with:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A\left(s,a';\theta,\alpha\right)\right) \tag{7}$$

where θ denotes the parameters of the layers before the output layer, while α and β are the parameters of the two streams of fully-connected layers.

In this study, DDQN was used to estimate the action value function in scheduling in combination with prioritized experience replay and dueling network, and the mixed form is called dueling double DQN with prioritized replay (DDDQNPR). The following section describes in detail how this algorithm can be used to solve scheduling problems.

B. SCHEDULING PROBLEM TRANSFORMATION

Disjunctive graph-based scheduling dispatching is used to choose the preferred job from the ready tasks set for processing until all jobs are scheduled. It is essentially a sequential decision-making process. The key to the application of RL in solving scheduling problems is to transform scheduling problems into Markov or semi-Markov decision problems, that is, defining states, actions, and reward function.

1) STATE FEATURE EXPRESSION

State describes the scheduling environment characteristics including the global and local characteristics. The state space is a collection of all possible states. The state in which an agent needs to make a decision is called a decision state. For problems with small-scale state space, each state or state-action pair can be explicitly represented as arrays or tables. In this case, the RL is called tabular RL. However, practical problems often have a large or continuous state space, and RL algorithms cannot traverse all the states. Therefore, tabular RL algorithms will face the "dimension disaster" problem. One common way to do this is to generalize value functions, which can generalize from previous encounters with different states that are in some sense similar to the current one with a good approximation. The most important basis for approximation is to select suitable features. Here we describe the characteristic attributes that can be used to describe the scheduling system, and then construct the state features for RL.

In general, the selection of state features should follow the following principles:

a. The state features can describe the main features and changes of the scheduling environment, including global and local information.

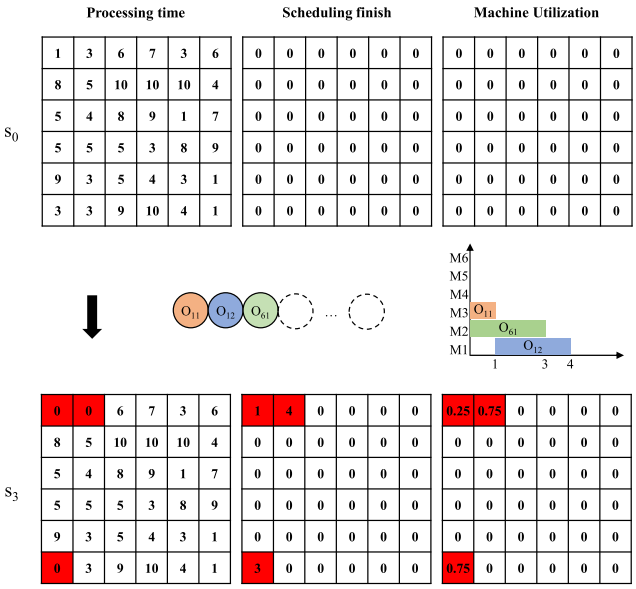


FIGURE 5. Scheduling state transition after scheduling the first three operations.

b. The selection of state features should be related to the scheduling objective, otherwise it will cause feature redundancy.
c. All states of different scheduling problems are represented by a common feature set.
d. The state features are numerical representation of the state attribute, which should be easy to calculate, and be normalized to ensure scale uniformity.

To make full use of deep learning to extract features from a raw input, a novel way of state expression is proposed, by which different scheduling features are used as different channels of an image with a height of job number and a width of operation number of each job. The scheduling features used here include processing time, scheduling result at current time step, and machine utilization.

Processing time channel is a matrix consisting of the processing time of each operation. Each element is normalized by the maximum processing time. The value will be zero if the operation has been assigned to a machine.

Scheduling result channel is a matrix composed of the finish time of each operation. Each element is normalized by the current makespan and initialized to zero.

Machine utilization channel represents the utilization of the machine required for each operation. Each element is initialized to zero and does not need to be normalized because its value is between 0 and 1.

Fig. 5 illustrates the scheduling state transition process with the example of ft06. At the initial state s_0, each value in the processing time channel is the processing time defined in the scheduling instance, and the values in both scheduling result channel and machine utilization channel are all zeros. Assuming that topological sorting is obtained according to one rule, the scheduling environment enters a new state after each operation is scheduled. When the first three operations

are scheduled, the scheduling environment now is in state s_3, and the state values at the corresponding position (red area) of the three operations in each layer are updated. After that normalization will be performed.

2) ACTION DEFINITION

In a scheduling problem, the action is to select the preferred job to dispatch according to the scheduling rule, and the action space consists of a variety of different heuristic rules. To overcome the short-sighted nature of any single rule, the suitable scheduling rules at different states for different scheduling problems are selected through RL. The following principles should be followed when determining heuristic rules.

a. The sorting attributes used in heuristic rules should be diversified, which can be related to or irrelevant to the state. For example, there should be rules designed according to processing time and due date. However, in the state expression, features about processing time exist, but the ones about due date do not exist.
b. Different rules can be created by sorting an attribute in a different order. For example, two different rules, SPT and LPT, can be generated according to the processing time.

In this work, eighteen heuristic rules commonly used for minimizing scheduling makespan are listed in Table 2.

3) REWARD FUNCTION

The reward function definition is closely related to the scheduling objective. The immediate reward obtained by each state transition reflects the immediate effect of the action performed and represents the short-term influence of the action on the scheduling scheme. Cumulative reward reflects the long-term effects, which is RL's goal to maximize.

This study focuses on minimizing the makespan, the scale of which is different for different scheduling problems. In order to evaluate the immediate action uniformly, the following transformation is applied.

$$U = \frac{1}{NM}\sum_{m=1}^{NM}\frac{\sum_{i=1}^{NJ}\sum_{h=1}^{NO_i}P_{ihm}}{C_{\max}} = \frac{P}{NM\cdot C_{\max}} \tag{8}$$

where U is average machine utilization, P is total working time, NO_i is the operation number of job i. P and NM are both constant, so minimizing the makespan is equivalent to maximizing the average machine utilization. Let $r_k = U(k) - U(k-1)$, then the cumulative reward R can be calculated as follows:

$$\begin{aligned} R &= \sum_{k=1}^{K} r_k = \sum_{k=1}^{K} U(k) - U(k-1) \\ &= U(1) - U(0) + U(2) - U(1) \\ &\quad + \cdots + U(k) - U(k-1) \\ &= U(k) - U(0) = U(k) \\ &= \frac{P}{NM\cdot C_{\max}(k)} \end{aligned} \tag{9}$$

TABLE 2. Job actions.

Symbol	Rule	Description
$Ja^{(1)}$	SPT	Select the job with the shortest processing time.
$Ja^{(2)}$	LPT	Select the job with the longest processing time.
$Ja^{(3)}$	SPT+SSO	Select the job with the minimum sum of the processing time of the current and subsequent operation.
$Ja^{(4)}$	LPT+LSO	Select the job with the maximum sum of the processing time of the current and subsequent operation.
$Ja^{(5)}$	SPT*TWK	Select the job with the minimum product of current processing time and total working time.
$Ja^{(6)}$	LPT*TWK	Select the job with the maximum product of current processing time and total working time.
$Ja^{(7)}$	SPT/TWK	Select the job with the minimum ratio of current processing time to total working time.
$Ja^{(8)}$	LPT/TWK	Select the job with the maximum ratio of current processing time to total working time.
$Ja^{(9)}$	SPT*TWKR	Select the job with the minimum product of current processing time and total working time remaining.
$Ja^{(10)}$	LPT*TWKR	Select the job with the maximum product of current processing time and total working time remaining.
$Ja^{(11)}$	SPT/TWKR	Select the job with the minimum ratio of current processing time to total working time remaining.
$Ja^{(12)}$	LPT/TWKR	Select the job with the maximum ratio of current processing time to total working time remaining.
$Ja^{(13)}$	SRM	Select the job with the shortest remaining machining time not including current operation processing time.
$Ja^{(14)}$	LRM	Select the job with the longest remaining machining time not including current operation processing time.
$Ja^{(15)}$	SRPT	Select the job with the shortest remaining processing time.
$Ja^{(16)}$	LRPT	Select the job with the longest remaining processing time.
$Ja^{(17)}$	SSO	Select the job with the shortest processing time of subsequent operation.
$Ja^{(18)}$	LSO	Select the job with the longest processing time of subsequent operation.

where k is the counter for operations assigned, which can be viewed as a discrete time step in RL. $U(k)$ and $C_{\max}(k)$ are average machine utilization and makespan at time step k as shown in Fig. 6.

C. DRL SCHEDULING ALGORITHM

1) EXPLORATION AND EXPLOITATION

Exploration and exploitation are two important problems to be solved when applying a RL algorithm. Exploration is to select actions that are not currently optimal, and to explore new actions. This may increase the action value to find better actions and gain a larger reward in the long run. Exploitation consists in performing the optimal action at present to make full use of the learned empirical knowledge and get more rewards in the short term. Common methods to balance "exploration" and ''exploitation'' [19] include the greedy method, epsilon-greedy, epsilon-decreasing strategy, SoftMax, etc.

Algorithm 1 Dueling Double DQN With Proportional Prioritization for Job Shop Scheduling

1:Initialize minibatch k, step-size η, replay memory D, replay period K and capacity N, exponents α and β, budget T, $\Delta = 0, p_1 = 1$
2:Initialize action-value function Q with random weights θ
3:Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
4:**for** $e = 1$ **to** M **do**
5: Reset schedule scheme and clear schedule results. Observe state s_1
6: **for** $t = 1$ **to** T **do**
7: Select action a_t based on modified epsilon-decreasing strategy (Refer to (12))
8: Execute action a_t in scheduling environment and schedule the job with the highest priority. Remove it from ready task set and add in its subsequent task
9: Observe reward r_t and next state s_{t+1}
9: Store transition (s_t, a_t, r_t, s_{t+1}) in D with maximal priority $p_t = \max_{i<t} p_i$
10: **if** $t \equiv 0 \bmod K$ **then**
11: **for** $j = 1$ **to** k **do**
12: Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
13: Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
14: Set

$$y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \hat{Q}\left(s_{j+1}, \arg\max_a Q\left(s_{j+1}, a; \theta\right), \theta^-\right) & \text{otherwise} \end{cases}$$

15: Compute TD-error $\delta_j = \left(y_j - Q\left(s_j, a_j; \theta\right)\right)^2$
16: Update transition priority $p_j \leftarrow \left|\delta_j\right|$
17: Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q\left(s_j, a_j\right)$
18: **end for**
19: Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
20: Every C steps reset $\hat{Q} = Q$
21: **end if**
22: **end for**
23:**end for**

The epsilon-decreasing strategy is used in most RL algorithms, in which the probability epsilon decreases over time and exploration is transferred to exploitation gradually. When epsilon becomes 0, it becomes a greedy method and selects the optimal action. The action selection rule based on this strategy is as follows:

$$a = \begin{cases} \arg\max_{a'} Q(a') & \textit{with probability } 1-\varepsilon \\ \textit{random} & \textit{with probability } \varepsilon \end{cases} \tag{10}$$

where ε is the probability of selecting an action randomly and is updated by the following equation.

$$\varepsilon = \varepsilon_0 - \varepsilon_{\min} * \min(1, n\_iter / N_{explore}) \tag{11}$$

where ε_0 is the initial value of ε, $\varepsilon_{\min}$ is the minimum of ε, n_iter is the current step counter, and $N_{explore}$ is the total exploring steps.

In job shop scheduling problems, the elitist mechanism is introduced to ensure that the training process can converge to the optimal solution accurately and quickly. At the end of the training phase, the action is determined with the same probability $(1-\varepsilon)$*0.5 according to the trained strategy and the best-known policy respectively.

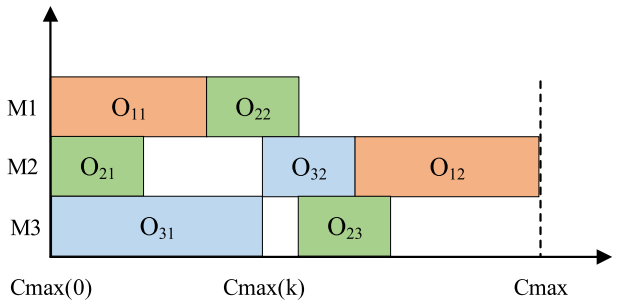


FIGURE 6. C_{max} at time step k.

$$a = \begin{cases} \begin{cases} \arg\max_{a'} Q(a') & \textit{with probability } (1-\varepsilon)*0.5 \\ \pi^{best}(s) & \textit{with probability } (1-\varepsilon)*0.5 \end{cases} \\ \textit{random} \quad \textit{with probability } \varepsilon \end{cases} \tag{12}$$

where π^{best} is the best-known policy so far.

2) TRAINING AND TESTING BASED ON DRL

In the training phase, a deep convolutional neural network architecture is employed for the Q-network. Specifically, the

input and output of the Q-network are the three-channel image which describe the processing time, scheduling results and machine utilization by each channel, and the predicted Q-values of the individual actions. Algorithm 1 describes the training phase of the proposed method. Given a scheduling instance for training the Q-network, lines 5-22 represent the complete process of scheduling all operations. The completion of one scheduling process is called as an episode, and e indicates the index of the episode currently being performed. Lines 5-22 continue until e reaches the maximum training episode M. At the starting time of an episode, the scheduling scheme is reset and the initial state s_1 is obtained (line 5).

At any time $t < T$ and T equals to the total operation number, the agent executes an action a_t according to the observed state. The action selection is based on the proposed ε-decreasing strategy with elitist (Equation 11). After the action a_t is executed, the job with the highest priority is scheduled and its current operation is removed from the ready task set. If the job has not been finished, its successive operation is added to the ready task set (line 8). Then the reward r_t and the next state s_{t+1} are observed and the transition $(s_t, a_t, r_t, s_{t+1})$is stored in the replay buffer (line 9 and 10). In line 10, it is also checked whether the number of stored transitions equals to N or not. If it is equal to N, new transitions replace the oldest ones.

Every period K, k transitions are sampled based on different probability and the importance-sampling weight for each transition is calculated (line 13 and 14). Given the sampled transitions, a loss is calculated from a Q-value and target value (line 15 and 16), and its absolute value is taken as the priority (line 17). In line 18, the learning algorithm performs a gradient descent step with respect to θ on the loss. The network parameter is updated by the accumulated weight-change on all the sampled transitions (line 20). To ensure the stable convergence of the training process, the weights of the target Q-network are periodically replaced to those for the Q-network. Finally, the trained Q-network is returned (line 25). Fig. 7 illustrates the above process more clearly in the form of a flow diagram.

The test phase is basically the same as the training process, but at this point only the optimal action is required to perform greedily (line 7), and the training process for the Q-network is no longer required (line 10-21).

VI. EXPERIMENTS

We evaluated the proposed algorithm on static benchmark problems and dynamic random problems.

A. ENVIRONMENT AND MODEL SETUP

In this study, some standard instances from the OR-Library were selected to initialize the scheduling environment. Table 3. lists the instances used in the experiment. All the benchmarks are static, namely it is assumed that all jobs are ready at time 0 and random factors are not considered, so each benchmark is both a training sample and a test sample. The goal of RL is to learn an optimal strategy that maximizes

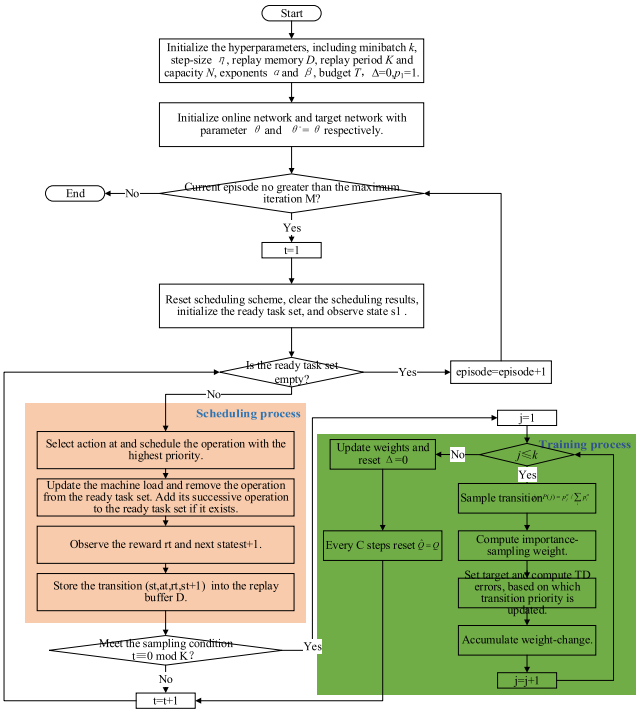


FIGURE 7. Scheduling process based on DRL

TABLE 3. Benchmarks.

Benchmarks	Source
ft06(6x6), ft10 (10x10), ft20(20x5)	Fisher and Thompson[70]
abz5(10x10), abz7(20x15)	Adams, Balas and Zawack[71]
la01-la40	Lawrence[72]
orb01(10x10), orb02(10x10)	Applegate and Cook[73]
yn01(20x20), yn02(20x20)	Yamada and Nakano[74]
dmu01.txt(20x15), dmu21.txt(40x15), dmu36.txt(50x20), mu41.txt(20x15), dmu61.txt(40x15), dmu76.txt(50x20)	Demirkol, Mehta and Uzsoy[75]
swv01.txt(20x10), swv06.txt(20x15), swv11.txt(50x10), swv16.txt(50x10)	Storer, Wu and Vaccari[76]
ta01.txt(15x15), ta11.txt(20x15), ta21.txt(20x20), ta31.txt(30x15), ta41.txt(30x20), ta51.txt(50x15), ta61.txt(50x20), ta71.txt(100x20)	Taillard[77]

the cumulative expected rewards from each state. RL agent should have certain generalization, and not just return an optimal solution. In order to achieve this, thirty instances with different initial states are randomly generated as a validation set, and in each instance $\delta = 30\%$ of the number of total operations are scheduled. δ is the scheduled ratio.

In order to verify the generalization of the proposed algorithm, random instances with different scheduled ratios, i.e., different initial states, were selected as the test sets. At the same time, to verify the robustness of the proposed algorithm, the random processing time disturbance was added into the environment. The processing time of each process follows a normal distribution $N(p_{ij}, \sigma^2)$, where p_{ij} is the predefined processing time, σ is the standard deviation.

B. TRAINING DETAILS

The deep architecture applied in the experiments was the dueling double DQN with prioritized replay, in which the

target network and the online network have the same configuration. The optimizer of the neural network is Adam. The dueling network splits into two streams of fully-connected layers. The value and advantage streams both have a fully-connected layer with 512 units. The final hidden layers of the value and advantage streams are both fully- connected with the value stream having one output and the advantage as many outputs as there are valid actions. The network has the rectifier linear unit [78] as an activation function and adopts batch normalization [79] in all convolutional layers and fully-connected layers except for the output layers of the two streams. The proposed algorithm is implemented by Tensorflow2.0 and run on a PC platform equipped with Windows 10 64 operating systems, 16G RAM, Intel i7 1.60GHz CPU.

In RL, determining the values of hyperparameters is crucial for the performance of the Q-network. Unfortunately, it is challenging to find the optimal values due to the large search space of hyperparameters. Therefore, by performing the random search [80], we found the values that yielded the best performance, and they are presented in Table 4. For the instance la16(10×10), the validation results under different network structure, learning rate, $\varepsilon_{\min}$, buffer size, target network updating frequency and batch size are analyzed to investigate their sensitivities. As shown in Fig. 8, the x and y coordinates of each point indicate the number of training and the average $C_{\max}$, respectively.

In Fig. 8(a), five different network structures are compared, in which each line represents a convolutional layer and comma symbols are used to separate the number of filters, the kernel size and the stride. It can be observed that the second network structure can achieve the optimal performance on the verification set. Note that no pooling layers are added between the convolutional layers. On the one hand, each pixel in the adopted state expression represents an operation, so pooling will remove some pixels, resulting in incomplete scheduling information. On the other hand, for the problems to be studied, the maximum size is 100×20, which is smaller than 84×84 in DQN, and the image size can be reduced only by the convolution kernels. In addition, the processing of the scheduling image is not the same as the general image processing, which first extracts rough features and then uses detailed features. This is because the scheduling image is not continuous, each pixel represents an operation, so local information should be retained as much as possible first, and then a wider range of information should be mined. This is proved by the comparison between the second and third network structures.

Fig. 8(b) indicates that too high or too low learning rate will cause a performance deterioration.

As shown in Fig. 8(c), the curve of $C_{\max}$ has the same trend under all $\varepsilon_{\min}$, but on the other hand, with the increase of $\varepsilon_{\min}$, the performance also decreases slightly.

Fig. 8(d) summaries the impact of buffer size on the $C_{\max}$, and it can be seen that the validation results are more stable when the buffer size is 100000 and 1000000.

FIGURE 8. Verification results of each hyperparameter in la16.

Fig. 8(e) shows that the target network updating frequency has little influence on the learning process, but the convergence speed is faster when the frequency is 200.

TABLE 4. Hyperparameters.

Hyperparameters	Values
Number of episodes	8000
Explore time steps	8000*total operation number*0.3
Epsilon	1-(1-ε_{min})*min(1,current_iter/explore time steps)
Buffer size	100000
Target Q update frequency	200
Batch size	256
Skip ratio	2%
Warm start	50
Prioritized replay alpha	0.6
Prioritized replay beta0	0.4
Reward gamma	1.0
Learning rate	10^{-5}

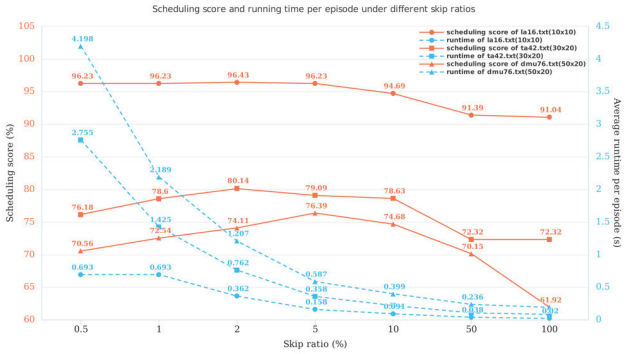


FIGURE 9. Scheduling scores and running time per episode under different skip ratios.

Fig. 8(f) means that the convergence speed is faster when the batch is 256.

In DQN, the agent sees and selects actions on every kth frame instead of every frame, and its last action is repeated on skipped frames. Inspired by this, the frame-skipping technique was also introduced into the scheduling problem, and several operations were scheduled based on the same rule after each selection. On the one hand, after an operation is scheduled, the state changing is too slight to affect the action selection. On the other hand, compared with training on every step, this technique allows the agent to train roughly k times more episodes without significantly increasing the runtime. Consequently, the number of skipped frame k is also studied as a hyperparameter to achieve a balance between scheduling accuracy and scheduling efficiency. Scheduling problems of different sizes have different episode length, so k is defined as in the form of the percentage of total number of operations, i.e. $k = NO{*}skip\ ratio$, where the *skip ratio* is 0.005, 0.01, 0.02, 0.05, 0.1, 0.5 or 1.

To evaluate the performance of different skip ratios, scheduling problems of different sizes were studied. The results are shown in Figure 9, in which the solid line represents the scheduling score and the dashed line represents the running time. It can be seen that with the increase of skip ratio, the running time per episode is gradually decrease, but the scheduling score tends to first rise and then fall. That is to say, training in each discrete time step cannot get the best solution, which is probably because the state transition in each scheduling step will lead to a huge state space, and it is difficult to find an optimal rule sequence from it, let alone make the agent converge to that sequence. However, it is obvious that the skip ratio cannot be too high. For example, if the skip ratio is 100%, then the scheduling ends when only one rule is executed, and this is the scheduling process same with simple rules. Meanwhile, the training process of RL is equivalent to supervised learning, in which only the initial state is input and the results of the 18 scheduling rules are obtained. Therefore, in order to balance the scheduling quality and training speed, generally, the *skip ratio* is valued within [0.02,0.05], and here let the *skip ratio* be equal to 0.02.

C. COMPARATION OF EXPERIMENTAL RESULTS

In the proposed DRL framework, eighteen heuristic dispatching rules were taken as the actions of the agent, and then the decision sequences of these rules are trained to obtain a scheduling result superior to any single rule. Therefore, the scheduling results of the proposed algorithm on all testing instances were first compared with those of heuristic rules. Simultaneously, to further illustrate the optimization performance, the proposed algorithm was also compared with GA. Two different encoding methods were used in GA, i.e. priority encoding, called GA_PRIORITY_CODE, and encoding using a rule number, called GA_RULE_CODE. The complete results are shown in Table 7 to Table 10 in the Appendix, where LB and UB in bold represent the lower bound and the upper bound respectively, and the numerical value in red is the solution corresponding to the optimal rule. Scheduling score $= C_{LB}/C_{alg}$, where C_{LB} is the C_{max} corresponding to the lower bound and C_{alg} is the C_{max} corresponding to the algorithm *alg*. The results demonstrate that the scheduling results of this algorithm are not worse than any heuristic rule in all instances, with an average performance improvement of 11.56%, and the scheduling performance in only two instances is slightly worse than that of the two genetic algorithms, but the overall average performance is improved by 5.6% and 6.17% respectively. The average scheduling score of DDDQNPR in all instances was 90.79%, higher than 81.82% of the best rule, 86.29% of GA_PRIORITY_CODE and 85.87% of GA_RULE_CODE.

The training process of instance ft06 is visualized in Fig. 10. The episode reward eventually converged to the maximum as the training progressed as shown in Fig. 10(a). The reason for the oscillation is that there was still a small probability of random action selection in the later stage of the training. Fig. 10(b) shows the training process of the makespan to be minimized, which has the opposite trend to the episode reward. The training curve of start Q value is shown in Fig. 10(c) and its significance is to predict the maximum reward from the initial state with the current neural network. After 1000 training episodes, the start Q value was basically stable and close to the true value. In Fig. 10(d) the variation of the errors in the training process is shown, and the final training error was close to 0.

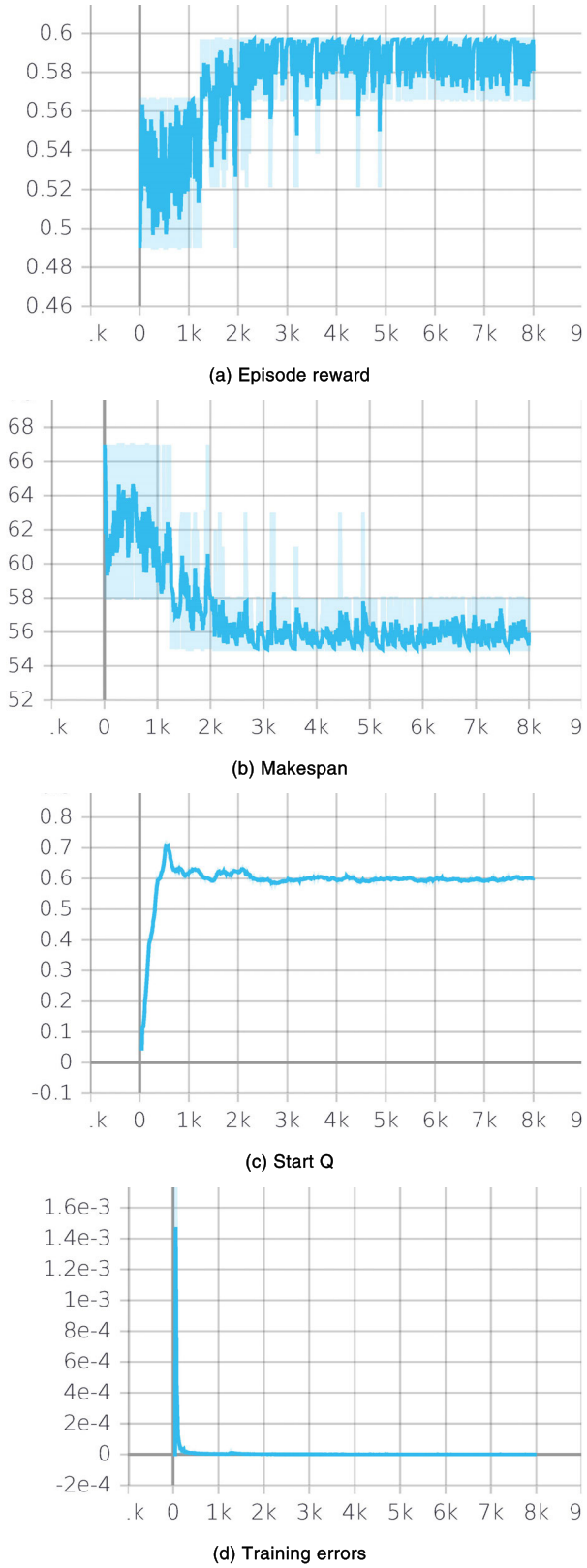


FIGURE 10. Training process of ft06.

In order to explain the disjunctive graph-based RL scheduling training process more clearly, the transition process of state, action and reward in each step was explained in detail

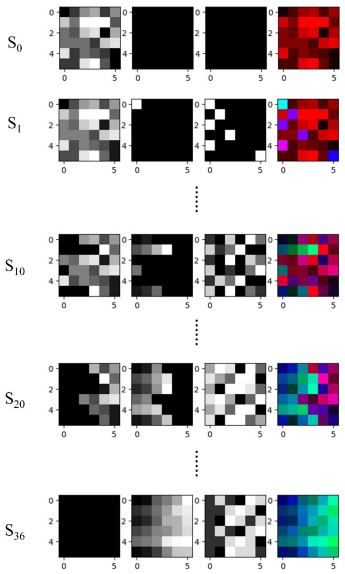


FIGURE 11. Scheduling state transition visualization for ft06 with the first three gray-scale images representing each channel, and the last one showing a color image that stacks three channels.

with an example of an episode during ft06 training, as shown in Fig. 11. Before the scheduling starts, we first initialized the ready task set $O_{\text{ready}} = \{O_{11}, O_{21}, O_{31}, O_{41}, O_{51}, O_{61}\}$, which consists of the first operation of each job. In the initial state s_0, the output of the neural network was [0.571401, 0.570248, 0.5685107, 0.5676666, 0.5711227, 0.57040733, 0.5690914, 0.56730187, 0.5695888, 0.570531, 0.5735151, 0.57267135, 0.57359517, **0.57441396**, 0.5734492, 0.57007927, 0.5694101, 0.57226104]. According to the exploration and exploitation strategy, supposing that the greedy behavior was selected, that is, the rule SPT∗TWKR (a_0) corresponding to the index 13 of the maximum Q value was executed to select the operation with the minimum product of current processing time and total working time remaining. It can be calculated that the sorting value of each operation in the ready task set is {1∗26 = 26, 3∗25 = 75, 6∗22 = 132, 7∗16 = 112, 3∗9 = 27, 6∗6 = 36}. Therefore, the operation O_{11} was first scheduled to the machine M_3 with the starting time of 0 and the ending time of 1. According to the reward function, the immediate reward $r_1 = U(1)\text{-}U(0) = 1/6\text{-}0 \approx 0.1667$. At the moment, the scheduling environment entered into a new state s_1. Finally, O_{11} was removed from the ready task set, and its successive operation O_{12} was added. This is the end of one-step scheduling. In the same way, we can get the transition of the following time step. Figure 11 also visualizes the state images after 10 and 20 operations are scheduled and at the end of the schedule, in which each state corresponds to 4 images. The first three are gray-scale images of each channel, and the last one is a color image stacking three channels. It can be seen that with the progress of scheduling, the system has undergone a significant state change. However, such images are not as easy to be recognized and understood as the images

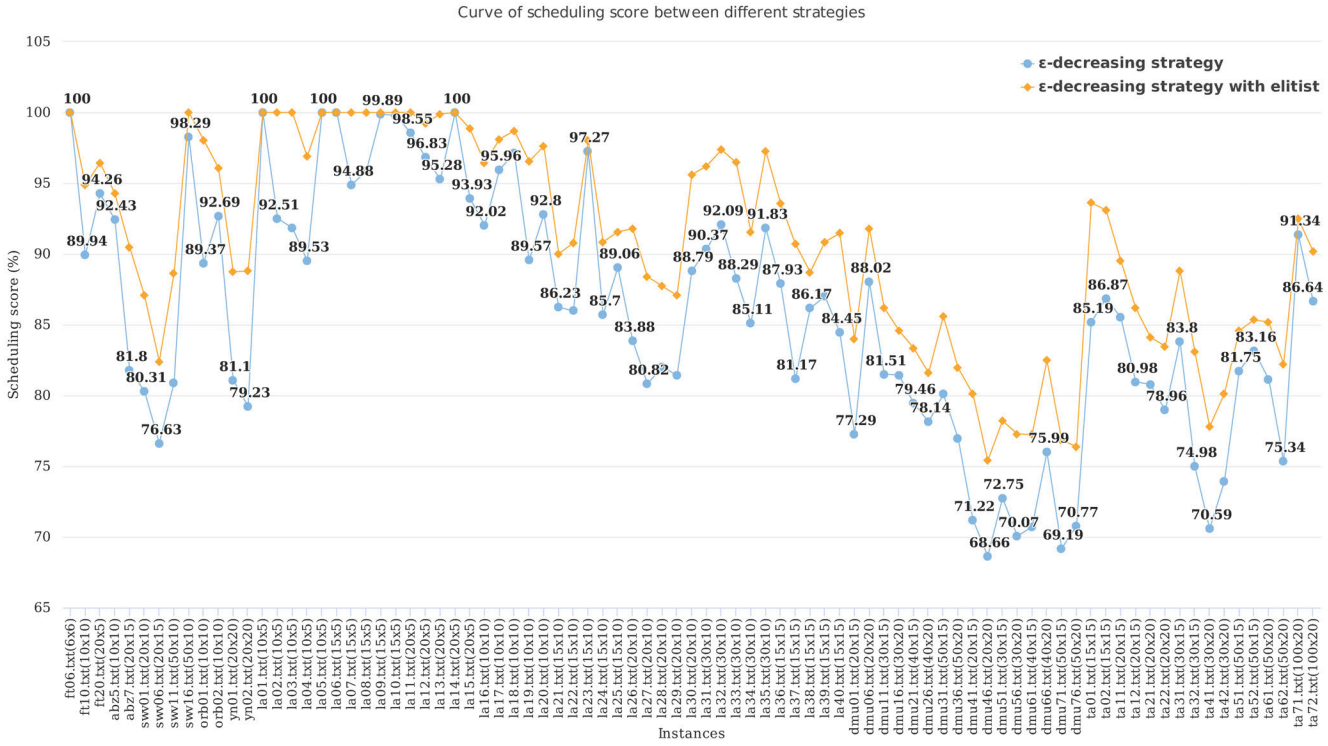


FIGURE 12. Comparison of different exploration vs. exploitation strategies.

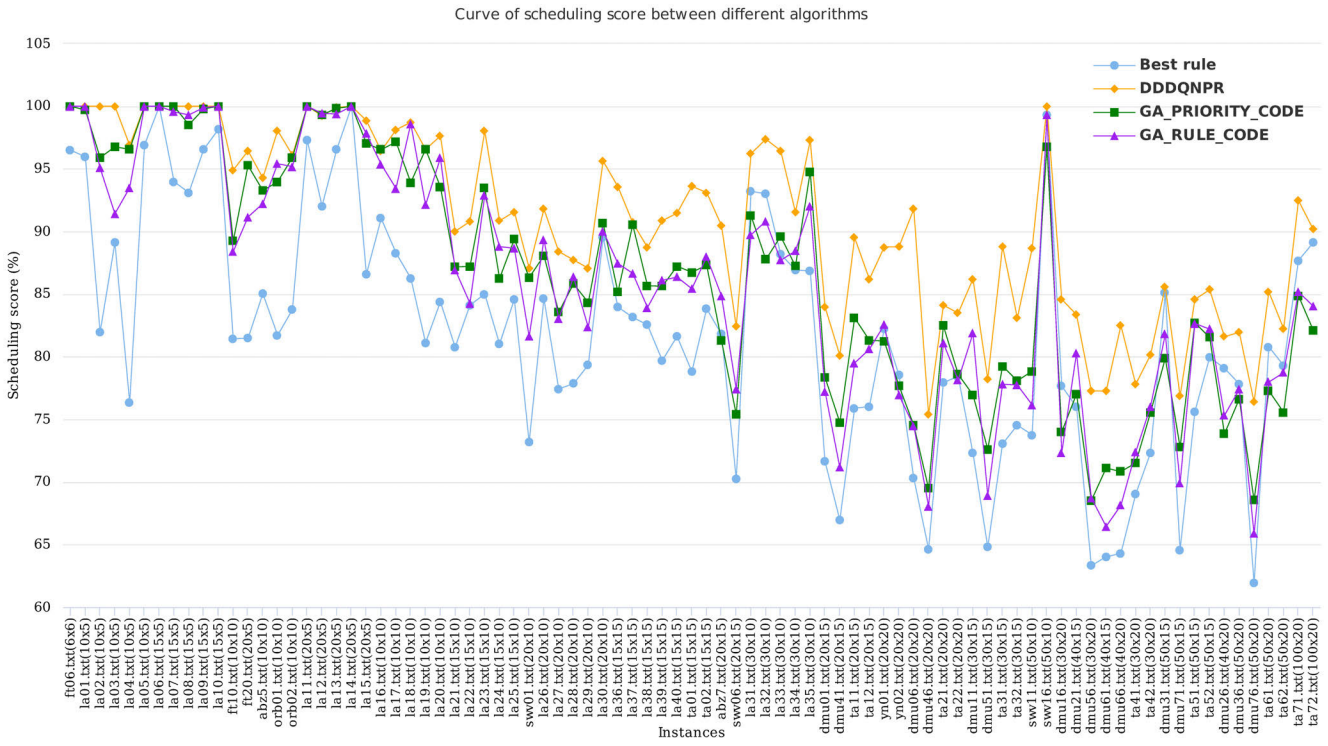


FIGURE 13. Curve of scheduling score between different algorithms.

taken at ordinary times. Therefore, we will try to interpret these images and their feature maps in future works.

The effect of the elitist strategy on the scheduling performance was also verified. The results of each scheduling instance under two different exploration and exploitation

strategies are summarized in Fig. 12, which shows that the exploration and exploitation strategy with elitist mechanism can further improve the solution quality. In the later stage of training, the ε decreasing strategy chooses greedy actions with a large probability, while the RL may fall into the local optimum. The exploration and exploitation strategy considering the elitist mechanism selects the optimal action of the global optimum solution with the same probability, the algorithm can jump out of local extreme points and approach to the global optimum.

Some interesting things can be found if arranging all the instances from Table 7 to Table 10 in order of problem size from small to large and drawing the scheduling score curve of each algorithm as shown in Fig. 13. First, the score curve of DDDQNPR has a similar trend with that of the optimal rule, which indicates that the performance of the proposed method is related to the quality of the rules. When the rule set contains better rules, RL is more likely to get high scheduling scores, and vice versa. Therefore, when selecting discrete rules using RL, an important task is to design better rules. Second, our method can obtain scheduling results that are not inferior to any single rule. At the same time, the performance of DDDQNPR is close to that of the two different GAs on the instances with the total number of operations dos not exceeding 150, but for the instances with a larger scale (total number of operations $\geq$200), DDDQNPR has significantly better performance. Finally, in general, the ability of this method to solve large-scale problems is somewhat lower than that of solving small-scale problems, because the scheduling state space of large-scale problems is huge, the learning error of using the same network structure is large, and more iteration times and more optimized network structure are needed to reduce the training error.

D. GENERALIZATION

Uncertain factors in real shop floor, such as insertion of a new job, machine breakdown, and delivery changes, may cause deviations from the predictive schedule over the course of its execution, and rescheduling is needed to adapt to the new environment. When rescheduling is performed, one job may have been finished or be being processed by a machine, so the scheme starts from a non-zero state at the rescheduling time. Moreover, in actual production environment, it is not to get an optimal solution, but to get a reasonable and feasible one quickly. Because the execution of a scheduling scheme is subject to many actual production factors, such as cutting tools, fixtures, measuring tools, special machines and even operators, all of which may lead the optimal solution to be an infeasible one. However, if all these limitations are modeled as constraints in the mathematical model, the mathematical model is bound to be huge, and the solution logic is very complex. Furthermore, the factors existing in the large-scale discrete manufacturing shop are far more than those listed above, and holographic modeling is also impossible. Therefore, in a complex production environment, the dispatcher will subjectively arrange some tasks that currently meet the processing conditions or are urgent, and then the remaining tasks will be sorted by an optimization algorithm. At this

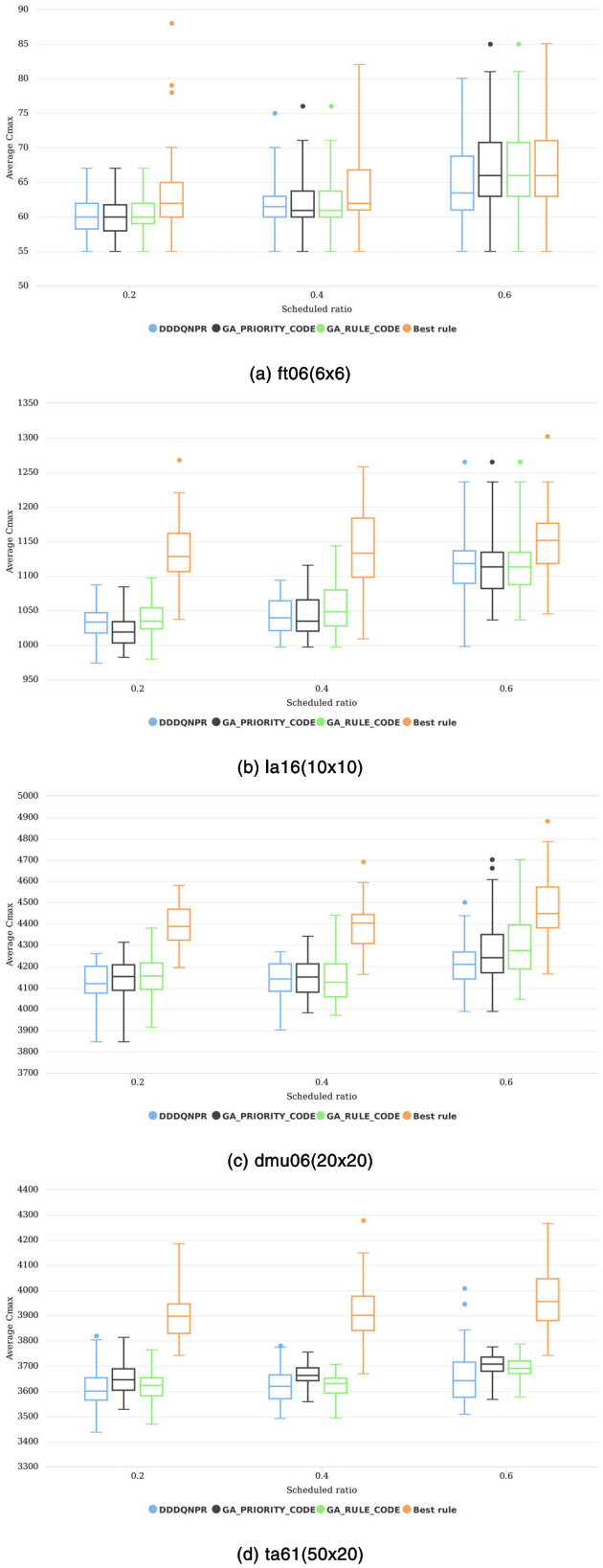
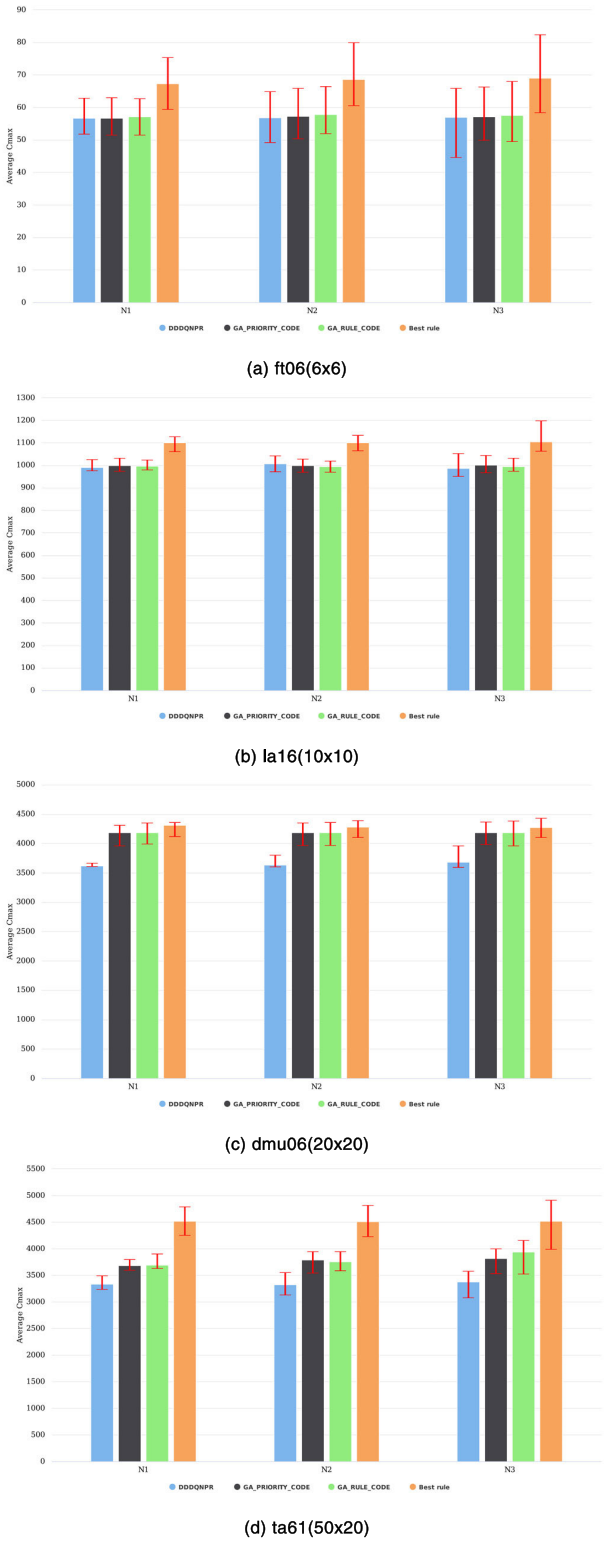


FIGURE 14. Scheduling results of each algorithm under different δ.



(a) ft06(6x6)

(b) la16(10x10)

(c) dmu06(20x20)

(d) ta61(50x20)

FIGURE 15. Scheduling results of each algorithm under different perturbation randomness.

time, the scheduling is also based on the scheme with non-zero state.

The scheduling scheme is updated in real-time when rescheduling is performed. The dispatchers are more

TABLE 5. Test time comparation in different initial states.

Instances	Scheduled ratio	DDDQNPR/s	Simple rule/s	GA/s
ft06(6x6)	0.2	0.04	1.42^{-4}	4.68
	0.4	0.03	1.31^{-4}	3.96
	0.6	0.02	1.14^{-4}	3.18
la16(10x10)	0.2	0.07	3.98^{-4}	22.86
	0.4	0.05	3.39^{-4}	14.31
	0.6	0.03	2.71^{-4}	10.90
dmu06(20x20)	0.2	0.13	2.92^{-3}	153.85
	0.4	0.10	2.26^{-3}	123.71
	0.6	0.06	1.63^{-3}	84.13
ta61(50x20)	0.2	0.33	1.45^{-2}	844.39
	0.4	0.25	1.13^{-2}	641.71
	0.6	0.17	8.02^{-3}	466.03

TABLE 6. Comparison of test time under stochastic processing time.

Instances	Average execution time		
	DDDQNPR	Best rule	GA
ft06(6x6)	0.05sec	1.55^{-4}sec	4.03sec
la16(10x10)	0.09sec	4.33^{-4}sec	18.97sec
dmu06(20x20)	0.16sec	3.48^{-3}sec	200.57sec
ta61(50x20)	0.44sec	1.81^{-2}sec	1082.43sec

concerned about the feasibility of the scheduling results, but not about the optimality. Therefore, the solution quality in the rescheduling case is not as important as the computation time. For non-zero state scheduling problems, the meta-heuristic algorithms require a large number of iterations to reschedule, and heuristic algorithms cannot be adapted to the changed environment. On the contrary, the DRL model proposed in this study can be trained in a random environment and quickly respond to new problems that have never been experienced before.

In order to verify the generalization performance of DDDQNPR, ft06(6×6), la16(10×10), dmu06(20×20) and ta61(50×20) were selected for the study. Some operations are scheduled randomly to establish non-zero state. The scheduled ratio δ was set to 20%, 40% and 60%, and 50 random instances were generated for each scheduled ratio as test sets. The results are displayed in Fig. 14, and the following conclusions can be summarized. (1) DDDQNPR is evidently superior to the simple rule. (2) The performance of DDDQNPR is close to that of the two GAs. Although the variance is larger on ta61, DDDQNPR has smaller minimum and median values. (3) With the increase of the scheduled ratio, each algorithm has a tendency to deteriorate, because the more operations are randomly scheduled, the less optimization space is left for algorithm. (4) The average test time of each algorithm is summarized in Table 5. It can be seen that the scheduling time increases with the increase of scheduling scale. Compared with GA (Since the running time of the GAs with different coding is basically the same, the GA here stands for either one), the test time of DDDQNPR is 2790 times and 114 times shorter respectively in the best and worst cases. At the same time, although it is 20-288 times longer than that of simple rules, the longest test time is also in seconds. DDDQNPR is perfectly acceptable to scheduler in terms of the calculation time taken to obtain a

TABLE 7. Static experiment results.

	SPT	LPT	LPT+LSO	LPT*TWK	LPT/TWK	LPT/TWKR	LPT*TWKR	LRM	LRPT	LSO	SPT+SSO	SPT*TWK	SPT*TWK	SPT*TWKR	SPT/TWKR	SRM	SRPT	SSO	LB	UB	GA (PRIORITY/RULE)	DDQN	Scheduling score(%)
ft06.txt (6x6)	87	73	71	71	78	83	73	57	67	61	86	89	71	90	61	94	94	88	**55**	**55**	55/55	55	100.00
ft10.txt (10x10)	1399	1534	1440	1554	1519	1472	1391	1142	1178	1424	1307	1445	1209	1516	1204	1415	1530	1471	**930**	**930**	1042/1052	980	94.90
ft20.txt (20x5)	1616	1613	1601	1629	1804	1650	1685	1614	1588	1430	1627	1656	1496	1689	1446	1595	1513	1609	**1165**	**1165**	1223/1278	1208	96.44
abz5.txt (10x10)	2006	1831	1606	2009	1693	1654	1512	1451	1451	1723	2042	1736	1834	1868	1572	2104	2093	1857	**1234**	**1234**	1323/1338	1309	94.27
abz7.txt (20x15)	1087	967	981	1038	1051	1038	871	845	802	969	1050	1107	1062	1176	866	1106	1082	1101	**656**	**656**	807/773	725	90.48
swv01.txt (20x10)	2043	2175	2112	2206	2455	2364	2149	2178	2123	1997	2135	2213	1964	2038	1923	2126	2093	2025	**1407**	**1407**	1630/1724	1616	87.07
swv06.txt (20x15)	2778	2442	2613	2524	2763	2729	2335	2321	2356	2483	2692	2706	2629	2655	2329	2833	2657	2452	**1630**	**1671**	2162/2105	1978	82.41
swv11.txt (50x10)	4378	4554	4614	4339	4499	4507	4309	4415	4382	4287	4095	4452	4444	4044	4492	4166	4193	4414	**2983**	**2983**	3783/3917	3365	88.65
swv16.txt (50x10)	3611	3478	3139	3581	3619	3607	3373	2945	2996	3206	3845	3899	3672	4023	2985	3785	4106	3752	**2924**	**2924**	3021/2945	2924	100.00
orb01.txt (10x10)	1434	1583	1564	1491	1479	1620	1424	1484	1495	1381	1384	1753	1447	1727	1537	1448	1489	1296	**1059**	**1059**	1127/1110	1080	98.06
orb02.txt (10x10)	1381	1376	1170	1422	1400	1215	1335	1182	1094	1261	1365	1437	1209	1350	1060	1314	1370	1346	**888**	**888**	926/933	924	96.10
yn01.txt (20x20)	1568	1314	1367	1389	1347	1469	1098	1100	1075	1316	1422	1500	1244	1573	1196	1645	1547	1360	**884**	**884**	1088/1071	996	88.76
yn02.txt (20x20)	1506	1364	1527	1497	1508	1475	1263	1108	1179	1360	1401	1539	1406	1589	1144	1627	1610	1675	**870**	**904**	1120/1131	980	88.78

schedule in seconds since rescheduling is usually conducted daily or hourly in many real-world production environments. (5) Since the scheduled ratio ($\delta = 30\%$) used to generate the validation set in the training scheduling policy is different from that used in the test, the generalization of the algorithm can be proved.

Finally, to evaluate the robustness of the proposed algorithm, further experiments were carried out on the problems with stochastic processing time, and a disturbance factor was added to the time of scheduling instances to simulate the scheduling environment with uncertain processing time. For each instance (ft06, la16, dmu06, ta61), the normal distribution $N\ (p_{ij}, \sigma^2)$ obeyed by the processing time was defined, where p_{ij} was theoretical value given in the original definition, and σ was the standard deviation which could be 1,2 or 3 to produce varying degrees of random disturbance. The RL agent interacted continuously with the dynamic environment and learned the optimal scheduling policy, which was stored locally in the form of parameters of each layer of the neural network, so that the backup could be restored when applied online.

For each instance size, test scheduling problems were solved 100 times with the learned model restored from the policy parameters. Besides, the rules and GA were evaluated with the same seeds. Fig. 15 depicts the average, minimum, and maximum C_{max} results of DDDQNPR, GA, and the best rule whose average C_{max} was the lowest among the eighteen simple rules, where N1, N2 and N3 in the x-coordinate respectively represent the normal distributions obeyed by processing time with variance of 1, 2 and 3. The following conclusions can be drawn. (1) On small-scale problems (ft06 and la16), DDDQNPR had the same performance as GA, while on larger-scale problems (dmu06 and ta61), DDDQNPR showed better performance than GA. In all instances, DDDQNPR was significantly better than the best rule, which means that the proposed algorithm learned the correct strategy to determine the proper heuristic rules in different scheduling states. (2) With the increase of variance, the difference between the maximum and the minimum C_{max}s calculated by each algorithm becomes larger, because the stronger the disturbance to the time, the greater the uncertainty of the problem. (3) The difference between the maximum and minimum C_{max}s becomes larger as instance scale increases. This observation can be attributed to the fact that the influence of uncertainty on the processing time increases as the sizes of state and action spaces grow. (4) The average test time of each algorithm on different scale instances was shown in Table 6. Since different disturbances and two kinds of GAs have little influence on the test time, there is no separate statistics here. It can be seen that DDDQNPR can still get better scheduling results in seconds in practical scale instances, but GA takes more than 1000 seconds. In a real production environment, the scale of the scheduling problem is often larger, and a rescheduling time that is too long will lead to an inability of catching up with the real-time production status.

VII. CONCLUSION

In this study we proposed an adaptive scheduling framework to deal with job shop scheduling problems based on DRL and disjunctive graph, which combines a dueling network, double DQN and prioritized experience replay. In the scheduling environment, based on the disjunctive graph model, the scheduling state at each time step is expressed as a multi-channel image, and the scheduling problem is transformed into a sequence decision problem through topological sorting. The action space of RL is a combination of heuristic rules that are easy to execute in a complex environment. The traditional ε decreasing strategy is improved by introducing the elitist mechanism to avoid algorithm falling into the local optimum.

Static environments are initialized by the benchmarks in the OR-Library, and the uncertain processing time is taken into account in dynamic environments. Experimental results show that the proposed algorithm can obtain better scheduling results than heuristic rules in all static instances, with an average scheduling score of 90.79%. Our algorithm outperforms traditional dispatching rules and executes almost as fast

TABLE 8. Static experiment results.

	SPT	LPT	LPT+LSO	LPT*TWK	LPT/TWK	LPT/TWKR	LPT*TWKR	LRM	LRPT	LSO	SPT+SSO	SPT*TWK	SPT*TWK	SPT*TWKR	SPT/TWKR	SRM	SRPT	SSO	LB	UB	GA (PRIORITY/RULE)	DDQN	Scheduling score(%)
la01.txt (10x5)	920	889	833	941	910	910	832	694	735	806	1013	1222	883	1153	820	1027	1210	844	**666**	**666**	668/666	666	100.00
la02.txt (10x5)	901	894	1020	833	979	854	871	905	812	815	1016	943	805	950	799	979	966	952	**655**	**655**	683/689	655	100.00
la03.txt (10x5)	770	748	707	813	747	782	788	787	704	793	906	899	794	889	670	834	897	904	**597**	**597**	617/653	597	100.00
la04.txt (10x5)	916	848	887	800	848	801	804	783	790	868	842	936	830	966	773	880	976	937	**590**	**590**	611/631	609	96.88
la05.txt (10x5)	827	787	730	767	748	785	707	612	612	633	940	914	667	841	621	661	905	894	**593**	**593**	593/593	593	100.00
la06.txt (15x5)	1369	1105	1047	1022	1083	1074	1020	968	926	1009	1555	1566	1101	1454	976	1439	1498	1201	**926**	**926**	926/926	926	100.00
la07.txt (15x5)	1128	1145	1142	1107	1099	1125	1070	1039	1031	1104	1245	1287	947	1293	1012	1289	1282	1187	**890**	**890**	890/894	890	100.00
la08.txt (15x5)	1168	1061	1150	1063	967	1089	1017	1023	1011	1131	1316	1231	1085	1375	927	1143	1348	1190	**863**	**863**	876/869	863	100.00
la09.txt (15x5)	1289	1105	1263	1215	1217	1129	1254	985	1091	1174	1276	1265	1308	1398	1042	1463	1384	1319	**951**	**951**	953/952	951	100.00
la10.txt (15x5)	1345	1136	1078	1140	996	1141	1065	1031	1052	976	1548	1602	1197	1558	1067	1279	1509	1604	**958**	**958**	958/958	958	100.00
la11.txt (20x5)	1654	1476	1421	1370	1507	1543	1355	1256	1316	1289	1709	1591	1387	1523	1401	1715	1754	1470	**1222**	**1222**	1222/1222	1222	100.00
la12.txt (20x5)	1352	1222	1241	1180	1271	1316	1140	1140	1167	1173	1311	1489	1201	1415	1129	1606	1622	1324	**1039**	**1039**	1046/1045	1047	99.24
la13.txt (20x5)	1747	1298	1271	1335	1450	1403	1269	1225	1191	1350	1466	1439	1402	1664	1254	1522	1517	1592	**1150**	**1150**	1152/1157	1151	99.91
la14.txt (20x5)	1757	1360	1319	1328	1360	1485	1309	1297	1292	1341	1899	1928	1408	1930	1295	1759	1669	1895	**1292**	**1292**	1292/1292	1292	100.00
la15.txt (20x5)	1476	1510	1551	1472	1546	1527	1476	1493	1415	1411	1685	1523	1394	1708	1547	1778	1900	1627	**1207**	**1207**	1244/1234	1221	98.85
la16.txt (10x10)	1588	1238	1230	1184	1171	1480	1184	1216	1118	1274	1289	1248	1420	1451	1038	1547	1371	1565	**945**	**945**	979/991	980	96.43
la17.txt (10x10)	1094	1157	976	1145	1291	1181	1012	888	1004	1060	1060	1490	1054	1249	909	1295	1416	1135	**784**	**784**	807/839	799	98.12
la18.txt (10x10)	1259	1264	1235	1290	1267	1413	1161	1027	983	1090	1131	1383	1097	1415	1008	1174	1353	1232	**848**	**848**	903/860	859	98.72
la19.txt (10x10)	1339	1140	1163	1351	1256	1177	1074	1051	1089	1134	1332	1299	1361	1279	1038	1375	1302	1225	**842**	**842**	872/914	872	96.56
la20.txt (10x10)	1331	1293	1230	1225	1308	1313	1103	1069	1076	1219	1375	1512	1284	1730	1145	1463	1447	1453	**902**	**902**	964/941	924	97.62
la21.txt (15x10)	1707	1545	1431	1667	1502	1672	1514	1295	1318	1416	1625	1814	1377	2018	1338	1760	1806	1962	**1046**	**1046**	1200/1203	1162	90.02
la22.txt (15x10)	1257	1409	1394	1397	1489	1759	1298	1102	1135	1421	1664	1781	1358	1761	1237	1686	1736	1428	**927**	**927**	1063/1100	1021	90.79
la23.txt (15x10)	1522	1330	1628	1417	1442	1516	1322	1214	1258	1348	1675	1737	1482	1761	1373	1587	1737	1604	**1032**	**1032**	1104/1111	1053	98.01
la24.txt (15x10)	1554	1472	1373	1464	1453	1500	1386	1181	1178	1281	1407	1695	1382	1714	1154	1618	1624	1701	**935**	**935**	1084/1053	1029	90.86
la25.txt (15x10)	1624	1382	1547	1491	1608	1603	1291	1155	1209	1429	1754	1780	1358	1638	1205	1869	1849	1642	**977**	**977**	1093/1102	1067	91.57
la26.txt (20x10)	2137	1616	1656	1696	1686	1790	1669	1498	1439	1797	1867	2150	1622	2113	1576	2183	1989	1988	**1218**	**1218**	1383/1363	1327	91.79
la27.txt (20x10)	2048	1776	1917	1755	1706	1931	1616	1619	1595	1693	2307	2082	1863	1975	1631	2009	2161	2037	**1235**	**1235**	1478/1487	1397	88.40
la28.txt (20x10)	2034	1668	1748	1719	1672	1703	1671	1562	1631	1580	1941	2030	1770	1965	1661	2283	2145	1765	**1216**	**1216**	1417/1408	1386	87.73
la29.txt (20x10)	2048	1649	1919	1620	1641	1765	1582	1521	1452	1602	1963	1872	1666	2048	1544	2161	2154	1713	**1152**	**1152**	1366/1399	1323	87.07
la30.txt (20x10)	2081	1783	1997	1813	1891	1848	1676	1514	1566	1689	2200	1951	1831	2141	1645	2110	2444	2017	**1355**	**1355**	1495/1505	1417	95.62
la31.txt (30x10)	2379	2394	2288	2245	2538	2381	2135	1914	2057	2148	2518	2579	2415	2947	2102	2912	2783	2296	**1784**	**1784**	1955/1988	1854	96.22
la32.txt (30x10)	2823	2571	2421	2694	2552	2509	2313	1989	1993	2238	2754	3008	2574	2710	2170	2807	3167	2674	**1850**	**1850**	2108/2037	1900	97.37
la33.txt (30x10)	2487	2372	2150	2195	2445	2424	2022	2003	1977	2099	2463	2504	2422	2628	1949	2514	2514	2209	**1719**	**1719**	1918/1960	1782	96.46
la34.txt (30x10)	2500	2425	2340	2252	2232	2515	2113	2041	1990	2027	2754	2610	2382	2499	1980	2607	2737	2337	**1721**	**1721**	1973/1946	1880	91.54
la35.txt (30x10)	2440	2514	2443	2321	2571	2618	2324	2174	2201	2316	2518	2654	2481	2756	2332	2682	2842	2425	**1888**	**1888**	1992/2052	1941	97.27
la36.txt (15x15)	2070	1884	1821	1720	2003	1792	1759	1510	1513	1848	2203	2363	1876	2354	1626	2462	2218	1955	**1268**	**1268**	1489/1450	1355	93.58
la37.txt (15x15)	2075	1940	2152	1851	1983	1983	1910	1691	1680	1915	2031	1971	1776	2320	1731	2151	2176	2251	**1397**	**1397**	1543/1612	1540	90.71
la38.txt (15x15)	1944	1841	1841	1834	1886	1826	1679	1449	1456	1528	1850	1784	1719	1807	1636	2005	1928	1818	**1196**	**1196**	1396/1425	1348	88.72
la39.txt (15x15)	1790	2064	1908	1865	1920	1902	2012	1547	1552	1855	1931	1952	1752	1859	1601	2289	2102	2181	**1233**	**1233**	1440/1432	1357	90.86
la40.txt (15x15)	2003	1829	1747	1707	1933	1922	1800	1497	1604	1780	1728	2147	1658	2050	1586	2006	2231	1850	**1222**	**1222**	1401/1415	1336	91.47

as heuristic dispatching rules in dynamic environments with uncertain processing time.

Future research will mainly focus on the following aspects. First, the reward function in this study adopted the average machine utilization difference between two successive time steps. We will evaluate whether there is a more efficient setup method. Second, this study only focused on the JSSP. In real life workshops, flexible JSSPs are more common due to the existence of multiple replaceable machines. Therefore, in future work we will apply the proposed DRL framework to solve flexible JSSPs. Third, our neural networks have different input sizes for scheduling problems of different sizes, and can only be used online to calculate scheduling problems of the same size. In the scheduling field, it is challenging to integrate domain and expert knowledge. Last, the DDDQNPR used in this study is inherently a value-based method, which cannot directly optimize over the scheduling policy. On this account, we will investigate other advanced policy-based methods including PG, AC, A3C, TRPO and compare their performances with DDDQNPR.

APPENDDIX

See Tables 7–10.

TABLE 9. Static experiment results.

	SPT	LPT	LPT+LSO	LPT*TWK	LPT/TWK	LPT/TWKR	LPT*TWKR	LRM	LRPT	LSO	SPT+SSO	SPT*TWK	SPT*TWK	SPT*TWKR	SPT/TWKR	SRM	SRPT	SSO	LB	UB	GA (PRIORITY/RULE)	DDQN	Scheduling score(%)
dmu01.txt (20x15)	4448	4304	3951	4007	4338	4072	4410	3491	3551	3949	4489	4517	4028	4445	3549	4426	4695	4864	**2501**	**2563**	3192/3238	2978	83.98
dmu06.txt (20x20)	6566	5556	4743	5382	5398	5602	5379	4335	4328	4718	5194	5400	4992	5513	4452	5371	5291	5018	**3042**	**3244**	4082/4085	3314	91.79
dmu11.txt (30x15)	6053	5621	5209	5216	5748	5299	5098	4696	4844	4780	5699	6591	4854	6359	5002	5521	6541	5871	**3395**	**3430**	4412/4147	3938	86.21
dmu16.txt (30x20)	6153	5994	6014	5921	5889	6014	5870	4808	4848	5760	6488	6171	6277	6918	5103	6742	6537	6557	**3734**	**3751**	5046/5161	4414	84.59
dmu21.txt (40x15)	7411	6232	6588	6870	6572	6838	6097	5872	5765	6200	6766	8057	6485	7490	5899	7303	7149	6593	**4380**	**4380**	5686/5455	5255	83.35
dmu26.txt (40x20)	7998	6739	7375	7085	7629	8215	6914	5962	5876	7677	8127	8653	7490	8168	6482	8807	8402	7916	**4647**	**4647**	6293/6170	5695	81.60
dmu31.txt (50x15)	8567	7938	8296	8139	7943	8024	7608	6628	6643	7654	8800	9213	8085	9616	7250	8983	9250	8346	**5640**	**5640**	7061/6894	6588	85.61
dmu36.txt (50x20)	9852	8283	8381	8134	8117	8586	7812	7329	7222	7716	9073	10558	8748	9575	7394	9970	10091	9557	**5621**	**5621**	7335/7263	6859	81.95
dmu41.txt (20x15)	5074	5548	4960	5327	5050	5106	5389	4490	4626	4846	5591	5618	5049	5576	4742	5124	5257	5141	**3007**	**3248**	4023/4226	3754	80.10
dmu46.txt (20x20)	6369	6059	5880	6025	6050	6486	6041	5530	5596	6205	6370	6546	5990	6463	5925	7063	6909	6048	**3575**	**4035**	5145/5253	4742	75.39
dmu51.txt (30x15)	6996	6801	6915	6572	7110	6639	6785	6102	6329	6450	6627	6599	6307	6990	6309	6880	6929	6601	**3954**	**4167**	5445/5736	5055	78.22
dmu56.txt (30x20)	8241	7782	7755	7915	7517	7583	7510	7318	7287	7518	7928	7931	7598	7744	7192	8199	7852	7218	**4554**	**4941**	6647/6627	5893	77.28
dmu61.txt (40x15)	8159	8097	8566	8344	8428	8614	8276	8202	8365	7829	7969	8279	7679	8095	8290	8190	8042	8041	**4917**	**5172**	6915/7399	6365	77.25
dmu66.txt (40x20)	8839	9510	8959	8990	9641	9435	8972	9215	8918	8730	9300	8868	8676	8961	8877	8504	8822	8394	**5397**	**5717**	7620/7921	6543	82.49
dmu71.txt (50x15)	9534	10046	9997	10176	9999	9727	9614	9478	9414	9517	9622	9638	9682	9646	9557	9593	9506	9564	**6080**	**6233**	8355/8697	7912	76.85
dmu76.txt (50x20)	11333	11447	11127	10922	11017	11060	11042	10310	10281	10333	10424	11400	10671	10926	10624	10242	10765	11380	**6342**	**6813**	9251/9628	8302	76.39

TABLE 10. Static experiment results.

	SPT	LPT	LPT+LSO	LPT*TWK	LPT/TWK	LPT/TWKR	LPT*TWKR	LRM	LRPT	LSO	SPT+SSO	SPT*TWK	SPT*TWK	SPT*TWKR	SPT/TWKR	SRM	SRPT	SSO	LB	UB	GA (PRIORITY/RULE)	DDQN	Scheduling score(%)
ta01.txt (15x15)	1872	1812	1892	1762	1973	1984	1860	1574	1562	1957	2199	1926	1647	2204	1664	2163	2148	2148	**1231**	**1231**	1419/1441	1315	93.61
ta02.txt (15x15)	1913	1562	1950	1598	1761	1764	1757	1484	1529	1759	2104	1933	1778	1957	1538	1814	2114	1905	**1244**	**1244**	1425/1414	1336	93.11
ta11.txt (20x15)	2273	2117	2471	2073	2163	2237	1930	1789	1788	2216	2045	2358	2037	2184	1886	2353	2442	2343	**1357**	**1357**	1633/1707	1516	89.51
ta12.txt (20x15)	2527	2213	2174	2142	2179	2207	1908	1813	1798	2187	2176	2406	2160	2556	1969	2459	2160	2253	**1367**	**1367**	1681/1696	1586	86.19
ta21.txt (20x20)	2488	2691	2443	2648	2770	2732	2802	2107	2145	2647	2898	3019	2338	2789	2206	3071	2955	2610	**1642**	**1642**	1991/2025	1952	84.12
ta22.txt (20x20)	2510	2515	2500	2613	2422	2545	2348	1990	2058	2522	2678	2666	2788	2822	2111	2796	2726	2636	**1561**	**1600**	1986/1998	1870	83.48
ta31.txt (30x15)	2993	2589	2670	2622	2565	2636	2491	2419	2415	2478	2976	3001	2619	3154	2435	3101	3156	2916	**1764**	**1764**	2226/2267	1986	88.82
ta32.txt (30x15)	3050	2624	2773	2681	2509	2649	2562	2380	2412	2634	2847	3271	2663	2981	2512	3166	3272	2890	**1774**	**1784**	2272/2282	2135	83.09
ta41.txt (30x20)	3105	3155	3129	3158	3222	3315	3014	2761	2784	2873	3148	3362	3266	3269	2898	3482	3232	3058	**1906**	**2005**	2665/2633	2450	77.80
ta42.txt (30x20)	3772	3356	3199	3348	3194	3043	2954	2629	2605	3096	3358	3738	3361	3965	2813	3641	3624	3528	**1884**	**1937**	2494/2479	2351	80.14
ta51.txt (50x15)	4456	3881	3873	3902	3859	3956	3833	3731	3650	3844	4099	4460	4058	4590	3768	4174	4443	4418	**2760**	**2760**	3337/3339	3263	84.58
ta52.txt (50x15)	4179	3891	3878	3900	4023	3976	3678	3579	3447	3715	4187	4556	3724	4694	3588	4588	4371	4059	**2756**	**2756**	3380/3353	3229	85.35
ta61.txt (50x20)	4500	4467	4432	4409	4258	4283	3943	3550	3620	4188	4523	4813	4174	4933	3752	5024	5041	4520	**2868**	**2868**	3711/3678	3367	85.18
ta62.txt (50x20)	4933	4416	4551	4365	4485	4494	4000	3618	3643	4217	4699	4863	4323	5310	3925	4764	4821	4757	**2869**	**2869**	3797/3644	3489	82.23
ta71.txt (100x20)	7830	6949	7065	6874	7067	7264	6819	6232	6247	6754	7638	8186	7445	7912	6705	7916	8118	7594	**5464**	**5464**	6438/6415	5908	92.48
ta72.txt (100x20)	7611	6675	6689	6758	7199	7011	6471	5814	5870	6674	7601	7640	6910	7643	6351	7607	7639	7077	**5181**	**5181**	6309/6165	5746	90.17

REFERENCES

[1] Y. Wang, K. Hong, J. Zou, T. Peng, and H. Yang, "A CNN-based visual sorting system with cloud-edge computing for flexible manufacturing systems," *IEEE Trans. Ind. Informat.*, vol. 16, no. 7, pp. 4726–4735, Jul. 2020, doi: 10.1109/tii.2019.2947539.

[2] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010, doi: 10.1016/j.comnet.2010.05.010.

[3] D. Boyd and K. Crawford, "Critical questions for big data: Provocations for a cultural, technological, and scholarly phenomenon," *Inf., Commun. Soc.*, vol. 15, no. 5, pp. 662–679, Jun. 2012, doi: 10.1080/1369118x.2012.678878.

[4] S. John Walker, "Big data: A revolution that will transform how we live, work, and think," *Int. J. Advertising*, vol. 33, no. 1, pp. 181–183, Jan. 2014, doi: 10.2501/ija-33-1-181-183.

[5] E. A. Lee, "Cyber physical systems: Design challenges," in *Proc. 11th IEEE Symp. Object/Component/Service-Oriented Real-Time Distrib. Comput.*, Los Alamitos, CA, USA, 2008, pp. 363–369.

[6] S. Wang, J. Wan, D. Li, and C. Zhang, "Implementing smart factory of industrie 4.0: An outlook," *Int. J. Distrib. Sensor Netw.*, vol. 12, no. 1, Jan. 2016, Art. no. 3159805, doi: 10.1155/2016/3159805.

[7] L. Li, "China's manufacturing locus in 2025: With a comparison of made-in-China 2025 and Industry 4.0," *Technol. Forecasting Social Change*, vol. 135, pp. 66–74, Oct. 2018, doi: 10.1016/j.techfore.2017.05.028.

[8] K. R. Baker, "Sequencing rules and due-date assignments in a job shop," *Manage. Sci.*, vol. 30, no. 9, pp. 1093–1104, Sep. 1984, doi: 10.1287/mnsc.30.9.1093.

[9] S. Chan Park, N. Raman, and M. J. Shaw, "Adaptive scheduling in dynamic flexible manufacturing systems: A dynamic rule selection approach," *IEEE Trans. Robot. Autom.*, vol. 13, no. 4, pp. 486–502, Dec. 1997, doi: 10.1109/70.611301.

[10] A. Janiak, E. Kozan, M. Lichtenstein, and C. Oäuz, "Metaheuristic approaches to the hybrid flow shop scheduling problem with a cost-related criterion," *Int. J. Prod. Econ.*, vol. 105, no. 2, pp. 407–424, Feb. 2007, doi: 10.1016/j.ijpe.2004.05.027.

[11] G. I. Zobolas, C. D. Tarantilis, and G. Ioannou, "Minimizing makespan in permutation flow shop scheduling problems using a hybrid Metaheuristic algorithm," *Comput. Oper. Res.*, vol. 36, no. 4, pp. 1249–1267, Apr. 2009, doi: 10.1016/j.cor.2008.01.007.

[12] Y. Huo and J. Y.-T. Leung, "Fast approximation algorithms for job scheduling with processing set restrictions," *Theor. Comput. Sci.*, vol. 411, nos. 44–46, pp. 3947–3955, Oct. 2010, doi: 10.1016/j.tcs.2010.08.008.

[13] M. H. Fazel Zarandi, A. A. Sadat Asl, S. Sotudian, and O. Castillo, "A state of the art review of intelligent scheduling," *Artif. Intell. Rev.*, vol. 53, no. 1, pp. 501–593, Jan. 2020, doi: 10.1007/s10462-018-9667-6.

[14] F. Yoon-Pin Simon and Takefuji, "Stochastic neural networks for solving job-shop scheduling. II. architecture and simulations," in *Proc. IEEE Int. Conf. Neural Netw.*, Jul. 1988, pp. 283–290, doi: 10.1109/ICNN.1988.23940.

[15] W. Remus and T. Hill, "Neural network models of managerial judgment," in *Proc. Annu. Hawaii Int. Conf. Syst. Sci.*, vol. 4, Jan. 1990, pp. 340–344, doi: 10.1109/HICSS.1990.205276.

[16] D. E. Akyol and G. M. Bayhan, "A review on evolution of production scheduling with neural networks," *Comput. Ind. Eng.*, vol. 53, no. 1, pp. 95–122, Aug. 2007, doi: 10.1016/j.cie.2007.04.006.

[17] G. R. Weckman, C. V. Ganduri, and D. A. Koonce, "A neural network job-shop scheduler," *J. Intell. Manuf.*, vol. 19, no. 2, pp. 191–201, Apr. 2008, doi: 10.1007/s10845-008-0073-9.

[18] J. Lim, M.-J. Chae, Y. Yang, I.-B. Park, J. Lee, and J. Park, "Fast scheduling of semiconductor manufacturing facilities using case-based reasoning," *IEEE Trans. Semicond. Manuf.*, vol. 29, no. 1, pp. 22–32, Feb. 2016, doi: 10.1109/tsm.2015.2511798.

[19] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[20] Y.-H. Dong and J. Jang, "Production rescheduling for machine breakdown at a job shop," *Int. J. Prod. Res.*, vol. 50, no. 10, pp. 2681–2691, May 2012, doi: 10.1080/00207543.2011.579637.

[21] X. Li, Z. Peng, B. Du, J. Guo, W. Xu, and K. Zhuang, "Hybrid artificia l bee colony algorithm with a rescheduling strategy for solving flexible job shop scheduling problems," *Comput. Ind. Eng.*, vol. 113, pp. 10–26, Nov. 2017, doi: 10.1016/j.cie.2017.09.005.

[22] H.-S. Choi, J.-S. Kim, and D.-H. Lee, "Real-time scheduling for reentrant hybrid flow shops: A decision tree based mechanism and its application to a TFT-LCD line," *Expert Syst. Appl.*, vol. 38, no. 4, pp. 3514–3521, Apr. 2011, doi: 10.1016/j.eswa.2010.08.139.

[23] K. Yanai, M. Yooa, and T. Yokoyama, "A proposal of real-time scheduling algorithm based on RMZL and schedulability analysis," *Procedia Comput. Sci.*, vol. 24, pp. 9–14, 2013.

[24] E. J. Anderson and C. N. Potts, "Online scheduling of a single machine to minimize total weighted completion time," *Math. Oper. Res.*, vol. 29, no. 3, pp. 686–697, Aug. 2004, doi: 10.1287/moor.1040.0092.

[25] J. Tian, Q. Wang, R. Fu, and J. Yuan, "Online scheduling on the unbounded drop-line batch machines to minimize the maximum delivery completion time," *Theor. Comput. Sci.*, vol. 617, pp. 65–68, Feb. 2016, doi: 10.1016/j.tcs.2016.01.001.

[26] K. R. Baker, "Minimizing earliness and tardiness costs in stochastic scheduling," *Eur. J. Oper. Res.*, vol. 236, no. 2, pp. 445–452, Jul. 2014, doi: 10.1016/j.ejor.2013.12.011.

[27] B. Liu, Y. Fan, and Y. Liu, "A fast estimation of distribution algorithm for dynamic fuzzy flexible job-shop scheduling problem," *Comput. Ind. Eng.*, vol. 87, pp. 193–201, Sep. 2015, doi: 10.1016/j.cie.2015.04.029.

[28] X.-N. Shen and X. Yao, "Mathematical modeling and multi-objective evolutionary algorithms applied to dynamic flexible job shop scheduling problems," *Inf. Sci.*, vol. 298, pp. 198–224, Mar. 2015, doi: 10.1016/j.ins.2014.11.036.

[29] Z. Dong, M. Y. Jiang, and Z. L. Pei, "Research on dynamic shop scheduling problem based on genetic algorithm," *Basic Clin. Pharmacol. Toxicol.*, vol. 124, p. 50, Dec. 2018. [Online]. Available: WOS:000452533800080

[30] J. Park, Y. Mei, S. Nguyen, G. Chen, and M. Zhang, "An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling," *Appl. Soft Comput.*, vol. 63, pp. 72–86, Feb. 2018, doi: 10.1016/j.asoc.2017.11.020.

[31] W. Chen, H. Yang, and Y. Hao, "Scheduling of dynamic multi-objective flexible enterprise job-shop problem based on hybrid QPSO," *IEEE Access*, vol. 7, pp. 127090–127097, 2019, doi: 10.1109/access.2019.2938773.

[32] M. Shahgholi Zadeh, Y. Katebi, and A. Doniavi, "A heuristic model for dynamic flexible job shop scheduling problem considering variable processing times," *Int. J. Prod. Res.*, vol. 57, no. 10, pp. 3020–3035, May 2019, doi: 10.1080/00207543.2018.1524165.

[33] S. X. Yang and D. W. Wang, "A new adaptive neural network and heuristics hybrid approach for job-shop scheduling," *Comput. Oper. Res.*, vol. 28, no. 10, pp. 955–971, Sep. 2001, doi: 10.1016/s0305-0548(00)00018-6.

[34] R. Varela, C. R. Vela, J. Puente, and A. Gomez, "A knowledge-based evolutionary strategy for scheduling problems with bottlenecks," *Eur. J. Oper. Res.*, vol. 145, no. 1, pp. 57–71, Feb. 2003, doi: 10.1016/s0377-2217(02)00205-9.

[35] R. Rangsaritratsamee, W. G. Ferrell, and M. B. Kurz, "Dynamic rescheduling that simultaneously considers efficiency and stability," *Comput. Ind. Eng.*, vol. 46, no. 1, pp. 1–15, Mar. 2004, doi: 10.1016/j.cie.2003.09.007.

[36] K. K. Lee, "Fuzzy rule generation for adaptive scheduling in a dynamic manufacturing environment," *Appl. Soft Comput.*, vol. 8, no. 4, pp. 1295–1304, Sep. 2008, doi: 10.1016/j.asoc.2007.11.005.

[37] N. Kundakc and O. Kulak, "Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem," *Comput. Ind. Eng.*, vol. 96, pp. 31–51, Jun. 2016, doi: 10.1016/j.cie.2016.03.011.

[38] T. Ning, M. Huang, X. Liang, and H. Jin, "A novel dynamic scheduling strategy for solving flexible job-shop problems," *J. Ambient Intell. Hum. Comput.*, vol. 7, no. 5, pp. 721–729, Oct. 2016, doi: 10.1007/s12652-016-0370-7.

[39] L. Wang, C. Luo, and J. Cai, "A variable interval rescheduling strategy for dynamic flexible job shop scheduling problem by improved genetic algorithm," *J. Adv. Transp.*, vol. 2017, pp. 1–12, 2017, doi: 10.1155/2017/1527858.

[40] H. Xiong, H. Fan, G. Jiang, and G. Li, "A simulation-based study of dispatching rules in a dynamic job shop scheduling problem with batch release and extended technical precedence constraints," *Eur. J. Oper. Res.*, vol. 257, no. 1, pp. 13–24, Feb. 2017, doi: 10.1016/j.ejor.2016.07.030.

[41] Y. Zhang, J. Wang, S. Liu, and C. Qian, "Game theory based real-time shop floor scheduling strategy and method for cloud manufacturing," *Int. J. Intell. Syst.*, vol. 32, no. 4, pp. 437–463, Apr. 2017, doi: 10.1002/int.21868.

[42] H. Piroozfard, K. Y. Wong, and W. P. Wong, "Minimizing total carbon footprint and total late work criterion in flexible job shop scheduling by using an improved multi-objective genetic algorithm," *Resour., Conservation Recycling*, vol. 128, pp. 267–283, Jan. 2018, doi: 10.1016/j.resconrec.2016.12.001.

[43] I. Chaouch, O. B. Driss, and K. Ghedira, "A novel dynamic assignment rule for the distributed job shop scheduling problem using a hybrid ant-based algorithm," *Int. J. Speech Technol.*, vol. 49, no. 5, pp. 1903–1924, May 2019, doi: 10.1007/s10489-018-1343-7.

[44] M. E. Aydin and E. Öztemel, "Dynamic job-shop scheduling using reinforcement learning agents," *Robot. Auto. Syst.*, vol. 33, no. 2, pp. 169–178, Nov. 2000, doi: 10.1016/S0921-8890(00)00087-7.

[45] Y.-C. Wang and J. M. Usher, "Learning policies for single machine job dispatching," *Robot. Computer-Integrated Manuf.*, vol. 20, no. 6, pp. 553–562, Dec. 2004, doi: 10.1016/j.rcim.2004.07.003.

[46] Y.-C. Wang and J. M. Usher, "Application of reinforcement learning for agent-based production scheduling," *Eng. Appl. Artif. Intell.*, vol. 18, no. 1, pp. 73–82, Feb. 2005, doi: 10.1016/j.engappai.2004.08.018.

[47] C. D. Paternina-Arboleda and T. K. Das, "A multi-agent reinforcement learning approach to obtaining dynamic control policies for stochastic lot scheduling problem," *Simul. Model. Pract. Theory*, vol. 13, no. 5, pp. 389–406, Jul. 2005, doi: 10.1016/j.simpat.2004.12.003.

[48] B. C. Csaji and L. Monostori, "Stochastic reactive production scheduling by multi-agent based asynchronous approximate dynamic programming," in *Multi-Agent Systems and Applications* (Lecture Notes in Artificial Intelligence), vol. 3690, M. Pechoucek, P. Petta, and L. Z. Varga Eds. Berlin, Germany: Springer-Verlag, 2005, pp. 388–397.

[49] Z. Zhang, L. Zheng, and M. X. Weng, "Dynamic parallel machine scheduling with mean weighted tardiness objective by Q-Learning," *Int. J. Adv. Manuf. Technol.*, vol. 34, nos. 9–10, pp. 968–980, Sep. 2007, doi: 10.1007/s00170-006-0662-8.

[50] Z. Zhang, L. Zheng, N. Li, W. Wang, S. Zhong, and K. Hu, "Minimizing mean weighted tardiness in unrelated parallel machine scheduling with reinforcement learning," *Comput. Oper. Res.*, vol. 39, no. 7, pp. 1315–1324, Jul. 2012, doi: 10.1016/j.cor.2011.07.019.

[51] H.-B. Yang and H.-S. Yan, "An adaptive approach to dynamic scheduling in knowledgeable manufacturing cell," *Int. J. Adv. Manuf. Technol.*, vol. 42, nos. 3–4, pp. 312–320, May 2009, doi: 10.1007/s00170-008-1588-0.

[52] Y. Z. Wei and M. Y. Zhao, "Composite rules selection using reinforcement learning for dynamic job-shop scheduling," in *Proc. Conf. Robot., Automat. Mechatronics*, New York, NY, USA, vol. 1, 2004, pp. 1083–1088.

[53] S. Qu, T. Chu, J. Wang, J. Leckie, and W. Jian, "A centralized reinforcement learning approach for proactive scheduling in manufacturing," in *Proc. IEEE 20th Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2015, pp. 1–5.

[54] S. Qu, J. Wang, and G. Shivani, "Learning adaptive dispatching rules for a manufacturing process system by using reinforcement learning approach," in *Proc. IEEE 21st Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2016, pp. 1–5.

[55] H.-X. Wang and H.-S. Yan, "An interoperable adaptive scheduling strategy for knowledgeable manufacturing based on SMGWQ-learning," *J. Intell. Manuf.*, vol. 27, no. 5, pp. 1085–1095, Oct. 2016, doi: 10.1007/s10845-014-0936-1.

[56] J. Shahrabi, M. A. Adibi, and M. Mahootchi, "A reinforcement learning approach to parameter estimation in dynamic job shop scheduling," *Comput. Ind. Eng.*, vol. 110, pp. 75–82, Aug. 2017, doi: 10.1016/j.cie.2017.05.026.

[57] Y.-F. Wang, "Adaptive job shop scheduling strategy based on weighted Q-learning algorithm," *J. Intell. Manuf.*, vol. 31, no. 2, pp. 417–432, Feb. 2020, doi: 10.1007/s10845-018-1454-3.

[58] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*. [Online]. Available: http://arxiv.org/abs/1312.5602

[59] V. Mnih, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 33–529, Feb. 26 2015, doi: 10.1038/nature14236.

[60] H. van Hasselt, A. Guez, D. Silver, and Aaai, "Deep reinforcement learning with double Q-learning," in *Proc. Conf. Artif. Intell.*, Palo Alto, CA, USA, 2016, pp. 2094–2100.

[61] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.

[62] J. C. Caicedo and S. Lazebnik, "Active object localization with deep reinforcement learning," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 2488–2496.

[63] J. Li, W. Monroe, A. Ritter, M. Galley, J. Gao, and D. Jurafsky, "Deep reinforcement learning for dialogue generation," 2016, *arXiv:1606.01541*. [Online]. Available: http://arxiv.org/abs/1606.01541

[64] J. A. Palombarini and E. C. Martinez, "Automatic generation of rescheduling knowledge in socio-technical manufacturing systems using deep reinforcement learning," in *Proc. Biennial Congr. Argentina*, New York, NY, USA, 2018, pp. 1–5.

[65] J. A. Palombarini and E. C. Martínez, "Closed-loop rescheduling using deep reinforcement learning," *IFAC-PapersOnLine*, vol. 52, no. 1, pp. 231–236, 2019, doi: 10.1016/j.ifacol.2019.06.067.

[66] B. Waschneck, A. Reichstaller, L. Belzner, T. Altenmuller, T. Bauernhansl, A. Knapp, and A. Kyek, "Deep reinforcement learning for semiconductor production scheduling," in *Proc. 29th Annu. SEMI Adv. Semiconductor Manuf. Conf. (ASMC)*, Apr. 2018, pp. 301–306.

[67] C.-C. Lin, D.-J. Deng, Y.-L. Chih, and H.-T. Chiu, "Smart manufacturing scheduling with edge computing using multiclass deep q network," *IEEE Trans. Ind. Informat.*, vol. 15, no. 7, pp. 4276–4284, Jul. 2019, doi: 10.1109/tii.2019.2908210.

[68] C.-L. Liu, C.-C. Chang, and C.-J. Tseng, "Actor-critic deep reinforcement learning for solving job shop scheduling problems," *IEEE Access*, vol. 8, pp. 71752–71762, 2020, doi: 10.1109/ACCESS.2020.2987820.

[69] E. Balas, "Machine sequencing via disjunctive graphs: An implicit enumeration algorithm," *Operations Res.*, vol. 17, no. 6, pp. 941–957, Dec. 1969, doi: 10.1287/opre.17.6.941.

[70] H. Fisher and G. Thompson, "Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules," in *Industrial Scheduling, J. F. Muth and G. L. Thompson Eds. Upper Saddle River, NJ, USA: Prentice-Hall*, vol. 1963, pp. 225–251.

[71] J. Adams, E. Balas, and D. Zawack, "The shifting bottleneck procedure for job shop scheduling," *Manage. Sci.*, vol. 34, no. 3, pp. 391–401, Mar. 1988, doi: 10.1287/mnsc.34.3.391.

[72] L. S. Li, "Resouce constrained project scheduling: An experimental investigation of heuristic scheduling techniques (supplement)," School of Ind. Admin., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. ORNL/Sub-7654/1, 1984.

[73] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA J. Comput.*, vol. 3, no. 2, pp. 149–156, May 1991, doi: 10.1287/ijoc.3.2.149.

[74] Y. T. Li, "A genetic algorithm applicable to large-scale job-shop instances," in *Parallel Problem Solving from Nature*, R. Manner and B. Manderick Eds. Brussels, Belgium: North-Holland, 1992, pp. 281–290.

[75] E. Demirkol, S. Mehta, and R. Uzsoy, "Benchmarks for shop scheduling problems," *Eur. J. Oper. Res.*, vol. 109, no. 1, pp. 137–141, 1998, doi: 10.1016/s0377-2217(97)00019-2.

[76] R. H. Storer, S. D. Wu, and R. Vaccari, "New search spaces for sequencing problems with application to job shop scheduling," *Manage. Sci.*, vol. 38, no. 10, pp. 1495–1509, Oct. 1992, doi: 10.1287/mnsc.38.10.1495.

[77] E. Taillard, "Benchmarks for basic scheduling problems," *Eur. J. Oper. Res.*, vol. 64, no. 2, pp. 278–285, 1993.

[78] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *ICML Workshop Deep Learn. Audio, Speech Lang. Process.*, 2013, pp. 1–5.

[79] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, *arXiv:1502.03167*. [Online]. Available: http://arxiv.org/abs/1502.03167

[80] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012.

BAO-AN HAN was born in Qingdao, Shandong, China, in 1988. He received the B.S. and M.S. degrees from the School of Material Science and Mechanical Engineering, Beijing Technology and Business University, Beijing, China, in 2011 and 2014, respectively. He is currently pursuing the Ph.D. degree with the Laboratory of the Intelligent Manufacturing Technology and Systems, Department of Industrial and Manufacturing Systems Engineering, School of Mechanical Engineering and Automation, Beihang University, Beijing.

His research interests include production scheduling, deep reinforcement learning, as well as intelligent manufacturing and optimization.

JIAN-JUN YANG was born in Fuzhou, Jiangxi, China, in 1960. He received the B.S. and M.S. degrees in manufacturing engineering from Beihang University, Beijing, China, in 1986 and 1989, respectively.

From 1993 to 2002, he was an Associate Professor with the Institute of Manufacturing Systems. Since 2003, he has been a Professor with the School of Mechanical Engineering and Automation, Beihang University, where he has served as the Vice President of the College and the Director of the Department of Industrial and Manufacturing Systems Engineering. He is the author of one book, more than 100 articles. His research interests include manufacturing resource planning and intelligent optimization, production scheduling and hyper-heuristic algorithm, manufacturing process information integration, and control technology.