IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received July 27, 2021, accepted August 29, 2021, date of publication September 3, 2021, date of current version September 13, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3110242

Dynamic Jobshop Scheduling Algorithm Based on Deep Q Network

YEJIAN ZHAO, YANHONG WANG, YUANYUAN TAN, JUN ZHANG, AND HONGXIA YU
Institute of Artificial Intelligence, Shenyang University of Technology, Shenyang 100870, China

Corresponding author: Yanhong Wang (wangyh_sut@163.com)

This work was supported in part by Liaoning Provincial Key Research and Development Plan of China under Grant 2020JH2/10100041, and in part by the Youth Fund of the National Natural Science Foundation of China under Grant 62003221.

ABSTRACT Jobshop scheduling is a classic instance in the field of production scheduling. Solving and optimizing the scheduling problem of the jobshop can greatly reduce the production cost of the workshop and improve the processing efficiency, thereby improving the market competitiveness of the manufacturing enterprises. In order to make decisions on the complex dynamic scheduling process more accurately and simplify the solution process, the jobshop scheduling problem can be transformed into a reinforcement learning problem based on the Markov decision process. The performance of the adaptive scheduling algorithm in a dynamic manufacturing environment is improved based on the Deep Q Network (DQN). In the proposed scheduling algorithm, five state features of continuous value ranges are designed for input to a Deep Neural Network (DNN), as well as ten well-known heuristic dispatching rules are selected as the action set of the DQN. In the proposed scheduling algorithm, the target network and the prediction network are used to train the parameters. An action selection strategy based on the "softmax" function is designed in DQN. It selects dispatching rules with the largest action value as the execution action, thereby solving the problem that the suboptimal action value is greater than the optimal action Q value in the early learning stage. Furthermore, the non-optimal action is selected with a greater probability in the later learning stage. Ten benchmark jobshop test instances called "LA" used as simulation objects and operated in a simulation environment composed of Python. The simulation results confirm that the proposed scheduling algorithm based on DQN has better performance and universality than a single dispatching rule or traditional Q learning algorithm.

INDEX TERMS Dynamic scheduling, deep Q network, deep reinforcement learning, dispatching rules, job shop scheduling.

I. INTRODUCTION

The jobshop scheduling problem (JSP) is a comprehensive expression and simplified model for real manufacturing environments such as assembly shops, chip and semiconductor process manufacturing plants, and mechanical parts processing plants. The JSP has NP-Hard characteristics [1], it has been extensively studied since the 1950's. Many algorithms based on classical mathematical programming and various heuristic methods have been put forward to solve the JSP. Optimization capability and convergence speed are usually the main goals of scheduling algorithms. The jobshop is modeled as a static manufacturing environment with known production and processing attributes. However, the actual production environment is highly dynamic and rife with uncertain events. Unforeseen emergencies occur in real workshops, such as machine failures, rush orders, changes in processing start times, and shifting customer needs. Therefore, the optimization results obtained by existing scheduling algorithms in a certain scenario at hand cannot generally be applied under other conditions. A scheduling algorithm working well in a static environment is not readily applicable in the dynamic environments common in today's customized manufacturing era. So there is urgent demand for innovative, more effective scheduling methods that are better suited to dynamic production environments. The goal of the present study is to solve the dynamic jobshop scheduling problem (DJSP) by establishing a new scheduling algorithm,

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

and according to the characteristics of DQN and dispatching rules, a DQN-based DJSP scheduling algorithm is constructed.

The main contributions of this work can be summarized as follows. (1) Five state characteristic functions that conform to the actual production conditions are used to minimize the delay time. (2) Ten dispatching rules are imposed under the above performance indicators as the action group of the DQN algorithm. (3) A reward function is defined that reflects the current scheduling state in detail. (4) An action selection strategy based on the ''Softmax'' function is proposed, which has higher randomness of action selection than the traditional ''Greedy Strategy''. These four specific improvements are integrated into the DQN-based DJSP algorithm including the ''Target Network''.

In a dynamic environment such as ''the job arrives at the workshop randomly'', the above scheduling algorithm for solving DJSP can quickly and accurately select the best DRs for each scheduling time to guide the processing tasks in the actual production situation that changes from time to time. This algorithm can make the entire scheduling system run stably in a dynamic environment and achieve the desired performance indicators.

The rest of this paper is arranged as follows. Section II gives a literature review centered on existing DJSP scheduling methods. Section III introduces the basic framework of DJSP scheduling system based on multiple agents. Section IV summarizes the working principles of the RL algorithm and DQN algorithm. Section V presents the mathematical model and constraint conditions of DJSP. Section VI details the design process of the DQN algorithm used here to solve the DJSP. Simulation results are provided in Section VII which validate the proposed scheduling approach. Section VIII provides a conclusion and brief discussion on future research directions.

II. LITERATURE REVIEW

In order to solve the scheduling problem with NP-Hard characteristics, a large number of heuristic algorithms were born: density-based clustering methods [2], heuristic algorithm based on heterogeneous earliest completion time (HEFT) algorithm and list for solving non-preemptive scheduling problems [3] and the recursive algorithm to calculate the maximum processing time of a given task [4] and so on. Although the above method can obtain a better solution in the process of solving the static scheduling problem, it is difficult to adapt to the real-time changing dynamic scheduling environment caused by the interference of external factors. Therefore, the above methods have certain limitations in the process of solving DJSP. Moreover, they require a longer solution time, and at the same time greatly increase the computational complexity of the scheduling system.

Various algorithms have been proposed to solve the DJSP. The dispatching rule (DR) based scheduling algorithm is one of the most effective among them. Jobs to be processed in the next step are selected according to established rules. This reduces the running time and complexity of the classical Jobshop scheduling algorithm, so it is better-suited to DJSP requirements. Many such rules have been created and applied in real manufacturing environments.

The earliest DRs are the priority rules established by Jackson [5] for dual flow processing machines. Iskander [6] summarized 113 DRs. Jones *et al.* [7] divided DRs into simple rules, combined rules, and ordered rule sets. Naidu [8] summarized relevant rules for minimizing delay times. Durasevic and Jakobovic [9] tested some commonly used DRs on nine scheduling performance indicators and four types of scheduling problems to find that the different conditions resulted in various scheduling results and performance indicators.

Holthaus and Rajendran [10] proposed that no one rule can perform well in regards to all performance indicators, rather, one rule can only perform best on one or two indicators. The use of a single DR neither guarantees global optimality nor local optimality. Therefore, an effective scheduling method allows for dynamic selection of multiple DRs to meet job task requirements at different scheduling moments.

In the past, system simulation methods [11] was widely used methods to select DRs dynamically. The system simulation method simulates the various states that may exist in the DJSP in the system when interference occurs, which reveals the dispatching rule with the best performance in a specific production state. The working mechanism of this method is similar to a continuous ''enumeration'' process, but the disadvantage is that it requires a lot of computer simulation time and is difficult to meet the requirements of fast dynamic scheduling.

When dealing with DJSP, meta-heuristic algorithms usually work by converting dynamic problems into a series of static sub-problems, then applying classical intelligent optimization methods such as the Genetic Algorithm (GA) [12], Particle Swarm Optimization (PSO) [13], or Ant Colony Algorithm (ACA) [14] to select DRs and reach a solution. These methods are not flexible enough, however, to solve problems based on a single rule or multiple DRs. Most model information of the production and processing workshop must be incorporated to make the algorithm return accurate results. The actual workshop environment is highly dynamic, uncertain, and constantly changing over time. When the production environment of the workshop changes, the calculation speed, stability, and convergence of traditional intelligent optimization methods decrease; the algorithms are not sufficiently adaptive to modern production environments.

In recent years, machine learning algorithms have been increasingly applied in the DJSP field alongside advancements in artificial intelligence. Machine learning is an active approach to independently learning the patterns and models of an underlying system independently through data [15]. The next processing position of a given job is only related to the current processing position and has nothing to do with the previous processing position, which conforms to the Markov characteristic. Thus, the DJSP can be considered as a Markov decision process (MDP).

Reinforcement learning (RL) [16], a branch of machine learning, is considered a powerful MDP solution method. RL mainly uses its own knowledge in a dynamic environment to perform appropriate intelligent operations [17]. A scheduling method based on RL mainly uses agents to learn a real-time scheduling policy by continuously interacting with the production environment. This allows the best action at each scheduling moment to be properly allocated as the scheduling system completes the entire job process efficiently and accurately.

The RL method is an efficient policy for solving scheduling problems. Aydin and Ztemel [18] improved the Q learning process in traditional RL to build the Q-III RL algorithm, which trains the agent to select the best DRs in real time within different environments and under various conditions in the jobshop. The algorithm has since been used to guide various job processing tasks. Wang and Usher [19] encapsulated the Q learning algorithm in Agents to train a processing machine on a single-machine scheduling problem; they used three DRs as candidate actions of the algorithm to minimize the average delay time. Bouazza *et al.* [20] improved the Q-table in the traditional Q learning algorithm by storing machine selection probabilities and the probability of specific rules, which allocate the most suitable processing machines and the processing sequence of the jobs in a dynamic jobshop. Shiue *et al.* [21] used RL algorithms with Multiple Dispatching Rule (MDR) mechanisms and offline learning modules to maintain the Knowledge Base (KB) of a Real-Time Scheduling System (RTSS) that changes with the workshop environment. The scheduling results of their method were proven more effective than the machine learning scheduling algorithm of a single DR or other meta-heuristic algorithms.

Yang and Yan [22] proposed a scheduling algorithm that combines the Basic Sequential Access (BSA) algorithm with the Q learning algorithm for the DJSP. By clustering the state of the manufacturing system, the learning efficiency and generalization ability of the algorithm are improved and a better scheduling index is obtained. Shahrabi *et al.* [23] used the Q learning algorithm to find the best parameters of the Variable Neighborhood Search (VNS), then solved the DJSP of a job arriving at the workshop randomly. Wang [24] proposed a dynamic greedy search strategy and a new Q function weighted iterative update algorithm to determine optimal DRs, which resolves the blind search problem and has strong convergence and accuracy.

Defined state characteristics are relatively limited and discrete when Q learning is applied to scheduling problems. All Q-values of different actions in different states are usually recorded in a table called the ''Q-table''. In many actual manufacturing processes, the production state quickly becomes large-scale and continuous. This creates a massive quantity of state features and thus a sharp increase in the dimensions of the Q-table. The traditional Q learning algorithm queries the Q-table to obtain the action value. This drives down the versatility of the scheduling algorithm. Recent researchers developed the Deep Q Network (DQN) [25] as a combination of Q learning and neural networks to solve scheduling problems with continuous and large-scale state characteristics [26].

The DQN is an expression of the Deep Reinforcement Learning (DRL) algorithm. The Deep Neural Network (DNN) is used as a non-linear function fitter to estimate the value or selection probability of candidate actions in the action group. In existing DQN algorithms based on DJSP, the output form of the neural network is the output action value of an output action being selected. The output action value form performs better and is more commonly used in DQN algorithm designs. In DQN algorithms, all state features are used as inputs, Q-values are approximated by a neural network, and the action values of each action are output. The action selection strategy can be used to select the appropriate action to act on the scheduling process, so the Q-table does not need to store a large number of Q-values.

Shi *et al.* [27] applied a DQN-based scheduling algorithm to linear, parallel, and turn-back production lines in a discrete simulation environment. The algorithm showed more stable convergence and robustness than the traditional heuristic scheduling algorithm. Waschneck *et al.* [28] built a DQN-based scheduling algorithm for the coordinated training of multiple agents. Each agent learns the policy of different DRs at different scheduling moments so that the entire scheduling system reaches a globally optimal solution. Mao *et al.* [29] proposed a multi-resource cluster scheduler called DeepRM for scheduling problems, and built a system that can learn scheduling strategies directly from previous scheduling experience. Many resource scheduling problems can be transformed into DRL problems by this scheduling system, which showed fast convergence and high solution accuracy in simulation experiments.

Hu *et al.* [30] transformed the real-time scheduling of the AGV (Automatic Guided Vehicle) into an MDP for resource handling and scheduling of the flexible jobshop of an AGV. They developed a DQN learning scheduling algorithm as a learning policy. By using different DRs at different decision-making moments, the AGV scheduling task allocation problem was solved. Lin *et al.* [31] proposed an edge computing system framework that integrates the DQN algorithm and DRs for an intelligent manufacturing JSP, which expands the output range of the DQN algorithm and provides scheduling strategies for various production equipment. Their work showed that multiple DRs perform better than a single DR. Palombarini and Martinez [32] stores the learned policy for rescheduling in a DQN, which continuously learns scheduling processes from high-dimensional input data.

In summary, the DQN algorithm can better guide the scheduling process of jobs in the jobshop. Therefore, it is necessary to design a dynamic scheduling algorithm based on DQN to solve DJSP, so that the algorithm can be more effective and superior than previous intelligent algorithms.

TABLE 1. Problem description parameter set.

Parameter	Parameter Description
n	Total number of jobs
i, l	Job index, $i, l = [1,2,\ldots,n]$
m	Total number of machines
M_k	*kth* machine
k, h	Machine index, $k, h = [1,2,\ldots,m]$
J_i	*ith* job
j	Operations index
O_{ij}	*jth* operation of J_i
n_i	Number of operations in J_i
t	Current scheduling time

TABLE 2. Mathematical description parameter set.

Parameter	Parameter Description
t_{iM_k}	Processing time of J_i on M_k
c_{iM_k}	Processing completion time of J_i on M_k
$a_{iM_hM_k}$	When M_h processes J_i before M_k, $a_{iM_hM_k} = 1$, otherwise $a_{iM_hM_k} = 0$
b_{ilM_k}	When J_i is processed on the machine before J_l, b_{ilM_k}=1, otherwise $b_{ilM_k} = 0$
A_i	Release time of J_i
D_i	Estimated processing completion time of J_i
C_i	Actual processing completion time of J_i
f	Delay factor

III. PROBLEM DESCRIPTION

In the static JSP model, all jobs are considered to arrive at the production workshop simultaneously, and the task-scheduling process must be consistent. Before the end of the previous scheduling task cycle, new jobs are not allowed to enter the shop. The dynamic JSP model allows new jobs to-be-processed to enter the production workshop continuously and in time, including the production workshop where scheduling tasks are in progress. Thus, the DJSP refers to the classic JSP based on a variety of interference factors in processing operations (e.g., urgent orders, machine failures). Jobs arriving at the workshop in turn and being processing randomly on the machine is a typical case of DJSP.

DJSP can be described as follows. There are n jobs to be processed $J = \{J_1, J_2, \ldots, J_n\}$ on m machines $M = \{M_1, M_2, \ldots, M_m\}$ in a dynamic environment containing various disturbances. Each job J_i passes through n_i different processing sequences. The time taken for each process and the processing order constraints of J_i on each machine are known. The task goal of DJSP is to clearly select a suitable processing machine for each process of each job under various constraints and interference factors; its start time and the processing sequence of the job on each processing machine must be determined so that the scheduling result can fulfill the expected performance indicators.

The following constraints are imposed here to simplify the DJSP. (1) The preparation time of all machines is 0 and all jobs can be processed immediately after they arrive at the machine. (2) According to the processing technology regulations, each process only occurs after the previous process is completed. (3) Each job must be processed on the machine in a specified order according to the process route and different jobs have the same priority. (4) Each machine can only process one job at a time and each job can only be processed by one machine at a time. (5) A process cannot be interrupted once it has begun. (6) Subsequent operations must wait if the previous operation is not completed. (7) The preparation time and completion time of each job are included together in the processing time.

The parameters utilized in DJSP are listed in Table 1. Note that the time when a process is completed or a new job arrives is regarded as the "current scheduling time".

On-time delivery is one of the most important performance indicators for production operation management as well as a key factor in customer satisfaction and optimal workshop efficiency. The delay time T of all jobs was selected as the performance indicator of the DJSP accordingly. Appropriate constraints for job J_i were defined according to the characteristics of the DJSP. The performance indicator and constraints together form the DJSP mathematical model:

$$Min\,T = Min \max\{C_i - D_i, 0\} \quad (1)$$

$$c_{iM_k} \geq t_{iM_k} \quad (2)$$

$$c_{iM_k} - t_{iM_k} + M(1 - a_{iM_hM_k}) \geq c_{iM_h} \quad (3)$$

$$c_{lM_k} - c_{iM_k} + M(1 - b_{ilM_k}) \geq t_{lM_k} \quad (4)$$

Eq. (1) is the performance indicator (smallest delay time) function, Eq. (2) indicates that the completion time of any process cannot be less than its processing time, Eq. (3) is the order constraint between adjacent processes of the same job, and Eq. (4) shows where a new processing task can only be started after completing the previous processing task on the same machine M_k.

The mathematical parameters used in this analysis are listed in Table 2.

D_i is expressed as:

$$D_i = A_i + f \sum_{j=1}^{n_i} PT_{ik} \quad (5)$$

PT_{ik} is the processing time of J_i on M_k at time t.

IV. DJSP SCHEDULING SYSTEM FRAMEWORK

A. INTERACTIONS BETWEEN AGENT AND ENVIRONMENT

The constant interaction between the agent and the environment is one of the foundations for solving DJSP. In order to support the interaction between the scheduling algorithm and the environment, three types of agents were designed in this study: a Multi-agents Scheduling System (MSS), which includes a Management Agent (MA), as well as a Scheduling Agent (SA) and Resource Agent (RA). The functions of each agent were defined to make them into information processing modules with certain intelligent behavior and learning abilities. The MA is responsible for the release and coordination of scheduling tasks, the SA encapsulates the scheduling algorithm module, and the RA acts as the agent of the target machine equipment. Communication and cooperation between the three agents enable the MSSM to achieve job scheduling and resource optimization by selecting appropriate actions in a dynamic production environment.

When the jobs enter the scheduling system, the MA informs the SA to execute dynamic scheduling. And then the SA learns strategies by interacting with the production environment. It selects actions online and determines the corresponding job sequence and set of available machines. The goal of the SA is to select the most ideal actions dynamically based on the real-time conditions at the job site. It then instructs the RA to determine the allocation of different jobs on the machine and the processing sequence to achieve the scheduling processing index with the minimum delay time.

The working process of the MSS can be summarized as follows. When the MA obtains the scheduling task, it provides the SA with information (e.g., the queue of jobs to be processed, the set of available machines), then the SA obtains the corresponding scheduling policy by interacting with the job shop environment. The RA receives the actions given by the SA, then selects the corresponding job for processing.

The MSS based on MA, SA, and RA is shown in Fig. 1.

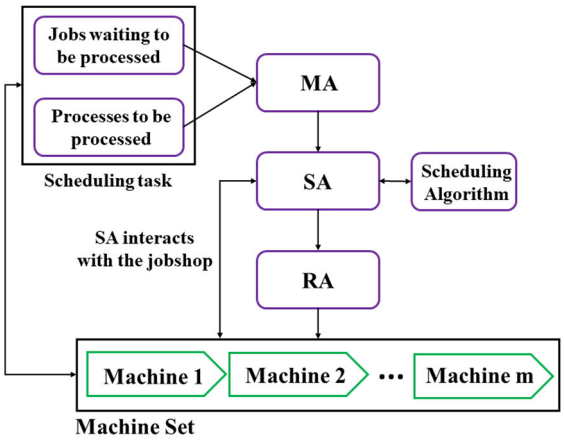


FIGURE 1. Multi-agents MSS framework.

B. SCHEDULING AGENT LEARNING MECHANISM

In the dynamic jobshop environment, job tasks continually enter the jobshop according to customer order requirements. The SA needs to determine the best DRs quickly and accurately within a constantly changing production environment. RL provides an effective scheduling tool for SA. The SA uses the RL-based encapsulated scheduling algorithm to make specific actions according to the current situation of the production environment during each semester cycle. These actions are continuously modified according to their return values. In the rule-based DJSP, the SA selection action involves executing an ideal DR. The SA selects an action based on the maximum sum of the expected reward value of the action rather than maximizing an immediate return value. Therefore, the SA performs actions repeatedly during production activities to learn the best DRs at any certain scheduling state.

The interaction mechanism between the SA and the dynamic jobshop environment is shown in Fig. 2.

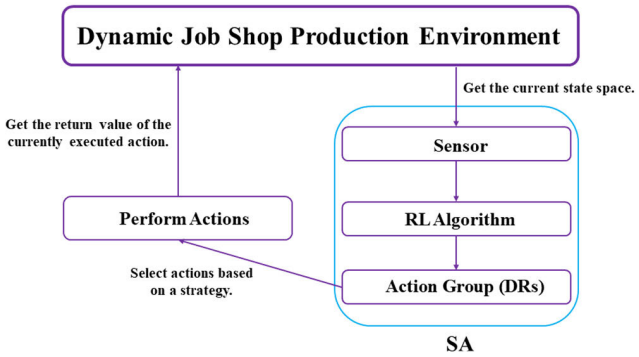


FIGURE 2. SA-environment interaction mechanism.

V. REINFORCEMENT LEARNING AND DEEP REINFORCEMENT LEARNING ALGORITHM

RL is a machine learning algorithm for solving model-free tasks. The Temporal-Difference (TD) in RL can learn an optimal policy during algorithm training processes, while the Monte Carlo Method (MCM) can learn the optimal policy only after algorithm training. The TD has a wider application scope than the MCM and is better suited to solving decision-making problems in dynamic environments. The DRL algorithm improves the performance of the traditional RL algorithm and uses a neural network to fit Q-values to manage decision-making problems with continuous state variables and large-scale data samples.

A. REINFORCEMENT LEARNING AND Q LEARNING

Markov Properties (MPs) indicate that the feedback of the environment and the next state of the environment are only related to the current state, not related to the previous step or earlier states. The DJSP conforms to Markov characteristics, so the learning mechanism of the SA in MSS can be described as an RL process. The MDP uses the SA to find a policy π through continuous sampling in the environment. When the MDP has obtained multiple strategies, it measures their respective pros and cons. The measurement standard is to maximize the reward value obtained by the SA in selecting a policy π over a long period of time. The RL serves to find the optimal policy π^* with the largest cumulative reward value through the MDP.

In the actual environment, most RL tasks are model-free tasks that do not require complete environmental information. The TD method was developed to efficiently solve RL problems by learning to generate experience trajectories. As a branch of TD, Q learning (QL) uses a non-fixed policy. That is, in the process of updating the action value function, the QL algorithm is more inclined to select actions with larger action values. The Q-values of all actions are stored in a limited table. Eq. (6) is the iterative update formula of the QL action value function:

$$Q(s,a) = Q(s_t,a_t) + \alpha[r_{t+1} + \lambda \max_a Q(s_{t+1},a_t) - Q(s_t,a_t)] \quad (6)$$

where α is the learning rate, which determines the degree of update to the old value. In order to make the algorithm tend to converge, α is usually small (e.g., 0.08). λ is the discount coefficient. If λ approaches 0, the agent tends to select the immediate reward value; if λ approaches 1, the agent tends to select the future reward value. The core of the QL algorithm can be expressed as follows:

$$r_{t+1} + \lambda \max_{a} Q(s_{t+1}, a_t) \tag{7}$$

B. DEEP REINFORCEMENT LEARNING AND DEEP Q NETWORK

When using QL algorithm to solve DJSP, excessive and continuous state characteristics force the Q-table to store so many Q-values that it grows excessively large in scale, i.e., there is a ''dimensional disaster''. It is possible to reduce the dimensions of state features. The DQN algorithm, for example, combines QL algorithm an d DNN as a branch of DRL algorithms. It approximates $Q(s, a)$ in a high-dimensional and continuous state. When the state space of the learning task is continuous and large, the DNN can effectively fit the value of $Q(s, a)$. The DQN algorithm was used as the main scheduling tool to solve the DJSP in this study.

The DQN algorithm uses the state characteristics of the system as the input value of the neural network's input layer. The Q-value of each ''state-action'' pair is the output value of the neural network's output layer, which is well suited to continuous, complicated decision-making problems. In the process of training DQN algorithm samples, to preserve the computational performance of the algorithm, the order of the training samples is usually shuffled; the training samples are stored in the shuffled order and a large number of continuous, highly correlated and non-static samples are generated as the input.

The strong correlation and non-staticity among input samples make the convergence of the DQN algorithm model difficult. It also may result in continuous fluctuations in the loss value. To resolve these problems, the DQN algorithm adopts an Experience Return Visit Mechanism (ERVM), and Min-batch Stochastic Gradient Descent (Min-batch SGD) training samples. The DQN algorithm stores the mini-batch experience samples after each training algorithm model into the Experience Pool (EP), then obtains a group of mini-batch continuous training samples in a fixed order. In this way, the correlation between training samples can be eliminated. Each time the DQN algorithm uses the Min-batch SGD to train a network model, it reuses this sequence. That is, the DQN algorithm uses training sets with variable group sizes and updates the neural network parameters θ by using them multiple times. The return value can also be saved.

The DQN algorithm neural network parameter training process is shown in Algorithm 1.

The Experience Reply (ER) mechanism is an important part of the DQN algorithm. First, an EP called D is established with a capacity of V. The DQN algorithm places the $\{\varphi_t, a_t, r_t, \varphi_{t+1}\}$ obtained by the agent interacting with the environment into the EP at each time step. Once the capacity of the EP is full, old experience samples are replaced with new ones. When it is necessary to use Min-batch SGD for network parameter training, the DQN algorithm randomly samples small batch experience samples from the EP as training samples.

Algorithm 1 DQN Algorithm Training Process

Initialize	1. Experience pool D and its Capacity V, the weight parameter θ of the predictive network Q
For	2. Experience trajectory, from 1 to L
	3. Initialize the state input vector $\varphi_1 = \{s_1\}$ of state s_1
For	4. Time step in experience trajectory, $t = 1 \sim T$
	5. Choose random action a_t with probability ε
	6. Otherwise perform action a_t according to $a_t = max_a Q(\varphi_t, a, \theta)$
	7. Perform action a_t, get the current reward $R(t)$, enter to the next state s_{t+1}, get the input vector of state s_{t+1}, $\varphi_{t+1} = \{s_{t+1}\}$
	8. Store experience samples $\{\varphi_t, a_t, r_t, \varphi_{t+1}\}$ in the D
	9. Randomly sample small batches of storage samples $\{\varphi_t, a_t, r_t, \varphi_{t+1}\}$ from the EP
	$y_j = r_j$, if the current state is the end state φ_{j+1}
	$y_j = r_j + \gamma \max_{a_t} Q(\varphi_{j+1}, a_t, \theta)$, if the current state is not finished
	10. Update the loss function
	$Loss = (y_j - Q(\varphi_j, a_j, \theta))^2$ by using SGD
End for	
End for	

VI. IMPROVED DQN ALGORITHM TO SOLVE DJSP

Designing a DQN-based scheduling algorithm for solving the DJSP necessitates defining state characteristics, establishing behavioral action groups, establishing a reward function, and developing action selection strategies. The state space and reward function affect the accuracy of the scheduling algorithm. The action group provides the actions that the scheduling algorithm can select. The optimal policy affects the final convergence of the DQN algorithm in the learning process.

In this section, some parameters used by state characteristics are defined, and mathematical formulas are used to describe the representation method of state characteristics at first; several DRs that are commonly used under the performance indicator were selected from the DR base as the behavior action group of the DQN algorithm, and the reward function used to calculate the reward value at each scheduled time is designed. Then, an action selection strategy with strong randomness was developed. The activation function used by the DQN algorithm for training was selected as well. Finally, the above supplementary points are integrated into the DQN algorithm containing the target network, and a complete DQN algorithm training framework is given.

A. STATE CHARACTERISTICS

The most important step in multi-stage decision-making processes when applying DQN algorithms is where the system

recognizes the state characteristics. The state characteristics are generally defined according to (1) the main features and changes of the scheduling environment, including system global characteristics and local characteristics; (2) expressions of various scheduling problems; (3) numerical representations of state variables; and (4) how easy and straightforward they are to calculate.

Judging the current system state is necessary to select the actions of the scheduling system. The correct state variable can accurately represent the current system state. In previous research, state characteristics are frequently defined as existing parameters in the job processing process and their derived variables, including the number of machines, the number of jobs, the longest processing time, the shortest processing time, the number of jobs in the waiting queue, and the number of work in process.

The disadvantages of these state space definitions are that the data is relatively discrete, the values are large numbers, and the description of the production status of the system is not comprehensive. If above state characteristics are input to the DQN algorithm, unfavorable factors such as high-dimensionality would emerge. The trained algorithm is not general and the algorithm does not readily converge, resulting in poor versatility within the industrial production process.

To solve this problem, a state variable representation method is proposed here which describes the various production states of the system continuously and comprehensively. At time t, the average utilization of all machines is $U_{ave}(t)$, the standard deviation of all machine utilization is $U_{std}(t)$, the average completion rate of all jobs is $CR_{ave}(t)$, the average remaining process waiting rate of all jobs is $RPWR_{ave}(t)$, and the load rate of all machines is $P(t)$. These variables are input to the DQN algorithm. Their value ranges are all [0, 1]. This reflects the production state of the entire system by the state variables and the processing situation of the scheduling system under different production states in detail.

The main parameters reflective of relevant state characteristics can be defined as follows.

$NPM_i(t)$: Number of passing machines;

$PCR_i(t)$: Processing completion rate;

$CTFP_i(t)$: Completion time of the final process;

PT_{ik} : Processing time;

$U_k(t)$: Utilization;

$EAST(t)$: Estimated average slack time;

$EART(t)$: Estimated average remaining processing time.

The specific descriptions of each parameter are given in Table 3.

The mathematical expressions of $U_k(t)$, $EAST(t)$, and $EART(t)$ are:

$$U_k(t) = \frac{\sum_{i=1}^{n} PT_{ik}}{CTFP_i(t)} \tag{8}$$

$$EAST(t) = \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{k=1}^{n_i - NPM_i(t)} PT_{ik} - (D_i - t)\right) \tag{9}$$

$$EART = \frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{n_i - NPM_i(t)} PT_{ik} \tag{10}$$

TABLE 3. State space parameter set.

Parameters	Parameter Description
$NPM_i(t)$	The number of machines that J_i flows through at time t
$PCR_i(t)$	Processing completion rate of J_i at time t
$CTFP_i(t)$	Completion time of the last process assigned on M_k at time t
PT_{ik}	Processing time of J_i on M_k at time t
$U_k(t)$	At time t, the utilization rate of M_k
$EAST(t)$	Estimated average slack time of remaining processes of all jobs
$EART(t)$	Estimated average remaining processing time of all jobs

TABLE 4. State space set.

Variables	State Variable Description
$U_{ave}(t)$	Average utilization of all machines at time t
$D_{std}(t)$	Standard deviation of utilization of all machines at time t
$P(t)$	Work load rate of all machines at time t
$CR_{ave}(t)$	Average completion rate of all jobs at time t
$RPWR_{ave}(t)$	Average remaining process waiting rate of all jobs at time t

The detailed description and expression of the state characteristics at each scheduling moment t are given in Table 4.

The five state characteristics are expressed as:

$$U_{ave}(t) = \sum_{k=1}^{m} U_k(t) \tag{11}$$

$$D_{std}(t) = \sqrt{\frac{\sum_{k=1}^{m}(U_k(t) - U_{ave}(t))^2}{m}} \tag{12}$$

$$P(t) = \frac{EART(t)}{EAST(t) + \mu} \tag{13}$$

$$CR_{ave}(t) = \frac{\sum_{i=1}^{n} NPM_i(t)}{\sum_{i=1}^{n} n_i} \tag{14}$$

$$RPWR_{ave}(t) = \frac{\sum_{i=1}^{n}(n_i - NPM_i(t))}{\sum_{i=1}^{n} n_i} \tag{15}$$

B. REWARD FUNCTION

The reward function should be designed according to the scheduling performance indicators. The design of the reward function should (1) reflect the immediate impact and reward of the action, (2) reflect the objective function value, and (3) be applied to problems of different scales, (4) let the scheduling algorithm converge as quickly as possible.

The selection of variables in the reward function affects the reward value and penalty value of the scheduling system in the current state. The reward function is generally defined based on the pros and cons of the state space in the job process. For instance, the reward value of a good state space is positive, the reward value of a bad state space is negative, and the reward value of other situations is 0.

The reward function broadly describes the actual production situation of each state space, but is often implemented after the formation of a certain processing state space; it does not apply in the formation of the state space, so it cannot fully reflect the pros and cons of the state space during the entire process or accurately reflect the dynamic characteristics of the DJSP.

In order to solve this problem, a reward function with the nature of "processing urgency" is proposed. Reward function variables are established in this study around the performance indicator of minimal delay time. When the scheduling system is under load, the deadline assigned by the scheduling system to the jobs to be processed will become more and more pressing. The $EAST(t)$ and $EART(t)$ are set as reward function variables. As the running time of the scheduling system increases and the scheduling actions are executed one by one, the $EAST(t)$ continually increases as the $EART(t)$ decreases. The reward function is expressed as:

$$R = \frac{EAST(t)}{EART\ (t) + 0.1} \tag{16}$$

According to Eq. (16), as time goes by, the return value in each state will gradually increase. This way of defining the reward function can more accurately describe the current situation at each scheduling moment.

C. ACTION MECHANISM SELECTION

The DQN algorithm selects different actions at different times and different manufacturing scenarios. The ultimate goal is to allow the SA to maximize the return value. The maximum return value determines the optimal action selected at each scheduling time window. The DQN algorithm seeks a trade-off between exploration and utilization when solving the optimal strategy.

In QL, action selection strategy is generally based on the greedy strategy ($\varepsilon-greedy$). But if the trained DQN algorithm is applied to actual production tasks, it should select actions with higher Q values with a higher probability. At the same time, the DQN algorithm cannot always select the action with the optimal Q value during the entire training process, so as not to fall into the local optimum.

The greedy strategy is improved to prevent these phenomena in this study. An action selection strategy is developed based on an improved "Softmax" function:

$$P\left(\varphi_t, a_i\right) = \frac{\exp(\mu Q(\varphi_t, a_i, \theta))}{\sum_{a\in A} \exp(\mu Q(\varphi_t, a, \theta))} \tag{17}$$

where μ is a scalar. When the Q-value of the initial state is 0, the probability of each action being selected is equal. Thus, the early action selection is more random. As the algorithm learning progresses over time, the reward value of each action is different and the updated Q-values also differ. The Q-value is higher when the reward value of an action is greater, and vice versa. At the beginning of learning, each action can be explored at random. As learning progresses, the system is more inclined to actions with higher Q values. This improves the rationality and effectiveness of the DQN algorithm's learning process.

D. ACTION GROUP

In the DJSP, the SA calculates a processing priority value for each job to be processed according to the conditions of the processing machine and production attributes of the processing task (e.g., delivery date, remaining processing time, delayed time). The jobs to be processed are sorted to complete the scheduling tasks. DRs are powerful means for calculating the job processing priority values.

The production process environment of the jobshop is dynamic, so it is necessary to dynamically deploy various DRs according to different production states. Dynamically selecting different DRs according to the changes in the state of the production/processing flow is better than using one DR throughout the entire flow. Therefore, ten DRs were used in this study as the behavior action group of the DQN algorithm. A large number of DRs can provide more solution strategies for DJSP.

1) MWKR (LONGEST REMAINING PROCESSING TIME RULE)

The job with the longest remaining processing time is preferred:

$$Z = -\sum\nolimits_{j=j'}^{n_i} (PT_{ik}) \tag{18}$$

2) LWKR (LEAST REMAINING PROCESSING TIME RULE)

The job with the least remaining processing time is preferred:

$$Z = \sum\nolimits_{j=j'}^{n_i} (PT_{ik}) \tag{19}$$

3) SPT (SHORTEST PROCESSING TIME RULE)

The job with the shortest process time is preferred:

$$Z = PT_{ik} \tag{20}$$

4) LPT (LONGEST PROCESSING TIME RULE)

The job with the longest process time is preferred:

$$Z = -PT_{ik} \tag{21}$$

5) LOPNR (LEAST NUMBER OF REMAINING PROCESSES RULE)

The job with the least number of remaining processes is preferred:

$$Z = n_i - NPM_i(t) \tag{22}$$

6) MOPNR (MOST REMAINING PROCESSES RULE)

The job with the highest number of remaining processes is preferred:

$$Z = -(n_i - NPM_i\,(t)) \tag{23}$$

7) SLACK (THE SMALLEST SLACK RULE)

The job with the smallest amount of slack is preferred, its expression is given as Eq. (24):

$$Z = D_i - t - \sum\nolimits_{k=1}^{n_i - NPM_i(t)} PT_{ik} \tag{24}$$

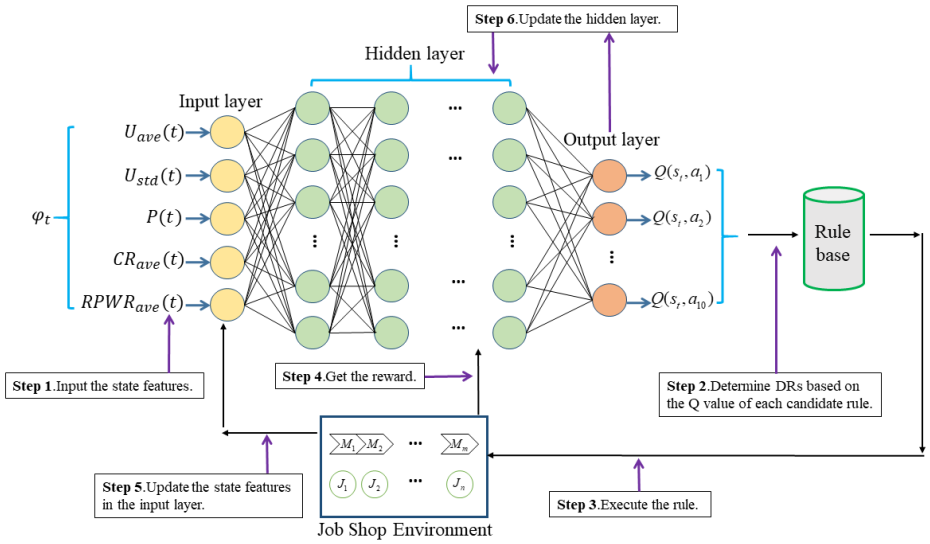


FIGURE 3. The DQN framework for solving DJSP.

8) S/WKR (SMALLEST RATIO OF SLACK PER WORK REMAINING RULE)

The jobs with the smallest remaining slack rate is preferred:

$$Z = \frac{D_i - t - \sum_{k=1}^{n_i - NPM_i(t)} PT_{ik}}{\sum_{k=1}^{n_i - NPM_i(t)} PT_{ik}} \tag{25}$$

9) CR (SMALLEST CRITICAL RATIO RULE)

The job with the smallest critical ratio is preferred:

$$Z = \frac{D_i - t}{\sum_{k=1}^{n_i - NPM_i(t)} PT_{ik}} \tag{26}$$

10) RANDOM

A job is selected at random.

E. UPDATED DQN ALGORITHM TRAINING PROCESS

In the QL algorithm, the predicted Q-values and target Q-values use the same parameter model:

$$Q(s,a) = Q(s_t,a_t) + \alpha[r_{t+1} + \lambda \max_a Q(s_{t+1},a_t) - Q(s_t,a_t)] \tag{27}$$

The target Q-values increase as the predicted Q-values increase, which increases the possibility of oscillation and divergence of the model to a certain extent. Therefore, when DQN algorithm uses the predictive network Q, an additional neural network called the target network $\tilde{Q}$ is used at the same time as the predictive network Q. The predictive network Q evaluates the current value of (s, a). The target network $\tilde{Q}$ generates the target value.

The DQN algorithm updates the parameter θ in the prediction network Q according to the loss function and assigns it to the parameter $\tilde{\theta}$ of the target network after a period of iteration. The target network $\tilde{Q}$ allows the target Q-values to remain unchanged for a period of time, which reduces the correlation between the predicted Q-values and target Q-values to a certain extent, thus minimizing oscillation and divergence of the loss value during training and improving stability and versatility of the whole algorithm framework.

The scheduling flow shown in Algorithm 2 was obtained by integrating the target network $\tilde{Q}$ and the various components of the above designed DQN algorithm into the DQN algorithm (Table 1).

Based on the above framework of DQN algorithm for solving DJSP, the framework model of Algorithm 2 as shown in the Fig. 3 is obtained:

F. ACTIVATION FUNCTION

In order to improve the expression and processing capabilities of neuron models in DNN, a nonlinear function called ''activation function'' was added to the neuron to improve the characterization and data-processing abilities of each neural network in the DNN. A neuron with nonlinear modeling ability can better process complex data, learn, and adapt in the DNN. Commonly used DNN activation functions include the Sigmoid, Hyperbolic Tangent function, and ReLU function. The ReLU function was used in this study to improve the prediction accuracy and convergence of the DNN framework:

$$ReLU(x) = \max\{0, x\} \tag{28}$$

G. PEFORM THE ENTIRE SCHEDULING PROCESS ACCORDING TO THE OUTPUT VALUE OF THE DQN ALGORITHM

In the DQN algorithm, the output layer of the DNN has 10 neurons, corresponding to the 10 candidate dispatching rules in the action group. At each scheduling moment, the five set state characteristics will be used as the input of DQN, which are mapped by the hidden layer. Finally, DNN get a set of data

Algorithm 2 DQN Algorithm Training Process With the Target Network

Initialize
1. Experience pool D and its Capacity V, the weight parameter θ of the predictive network Q, the weight parameter $\bar{\theta} = \theta$ of the predictive network $\tilde{Q}$

For
2. Experience trajectory, from 1 to L
3. Initialize the state input vector $\varphi_1 = \{U_{ave}(1), U_{std}(1), \mathrm{P}(1), CR_{ave}\}(1), RPWR_{ave}(1)\}$ of state s_1

For
4. Time step in experience trajectory, $t = 1 \sim T$
5. Choose random action a_t with "Softmax"
6. Perform action a_t, get the current reward $R(t)$, enter to the next state s_{t+1}, get the input vector of state s_{t+1},

$$\varphi_{t+1} = \{U_{ave}(t+1), U_{std}(t+1), \mathrm{P}(t+1), CR_{ave}(t+1), RPWR_{ave}(t+1)\}$$

7. Store experience samples $\{\varphi_t, a_t, r_t, \varphi_{t+1}\}$ in the D
8. Randomly sample small batches of storage samples $\{\varphi_t, a_t, r_t, \varphi_{t+1}\}$ from the EP
$y_j = r_j$, if the current state is the end state φ_{j+1}
$y_j = r_j + \gamma \max_{a_t} Q(\varphi_{j+1}, a_t, \bar{\theta})$, if the current state is not finished
9. Update the loss function $Loss = (y_j - Q(\varphi_j, a_j, \theta))^2$ by using SGD
10. Reset $\bar{\theta} = \theta$ every C steps

End for
End for

TABLE 5. Dispatching rule base.

Rules' Name	Rules' Description
MWKR	Select the job with the longest remaining processing time.
LWKR	Select the job with the shortest remaining processing time.
SPT	Select the job with the shortest processing time in the next process.
LPT	Select the job with the longest processing time in the next process.
LOPNR	Select the job with the least number of remaining operations.
MOPNR	Select the job with the lost number of remaining operations.
S/WKR	Select the job with the smallest remaining relaxation rate.
SLACK	Select the job with the least scheduling slack.
CR	Select the job with the smallest critical ratio.
RANDOM	Select a job randomly.

output from the output layer. In DJSP based on dispatching rules, each output data of DNN represents the action value of each dispatching rule. Then the "Softmax" function will reasonably select actions based on these action values. At this time, the production environment of the jobshop has changed; the values of the five state features are changed iteratively as the input layer is updated, then the relevant parameters of the hidden layer are updated. The output value of the output layer at the next scheduling time is obtained, and the cycle is repeated.

After the set number of training sessions, the fitting trend of the DQN algorithm gradually tends to converge, Thus, the algorithm has learned the policy of the corresponding optimal dispatching rules in each production state, and can dynamically allocate production tasks across the entire jobshop production process. This yields the expected the performance indicator under various constraints.

TABLE 6. Parameter set used in DQN training.

Parameter	Assignment
Number of neurons in input layer n_p	5
Number of neurons in output layer n_o	10
Number of neurons in each hidden layer n_h	50,100
Hidden layers N_h	5
Learning rate α	0.01
Discount Rate γ	0.87
Total training step	6000
EP capacity	3000
Batch per sample	128
μ of "Softmax"	0.7
Uniform distribution	[1,10]

VII. CASE STUDY

The proposed DQN algorithm was validated based on the DJSP with a variety of conditions and parameters. First, we calculated the performance indicator of each test instance under the action of the DQN algorithm with different delay factors f by comparison against the performance indicator obtained with the 10 DRs respectively. Next, we analyzed the convergence of each case under the action of the DQN algorithm by modifying the delay factor f. We then analyzed the convergence of the DQN algorithm when as number of neurons in each of its hidden layers increased. Finally, we explored the differences between the DQN algorithm and traditional QL algorithm in terms of convergence.

A. EXPERIMENT ENVIRONMENT

All the simulation experiments were completed on a desktop computer with a I7 7700K CPU, an acceleration frequency of 4.6 GHZ, and 16 GB memory. We used Python programming language. The dynamic nature of DJSP is reflected in the property that jobs arrive at the workshop randomly. Therefore, in the simulation process, the time for the jobs to arrive on the machine to start processing must conform to a certain degree of uniform distribution.

Ten classic "LA" test cases were selected as simulation objects. LA01 (10 * 5), LA02 (10 * 5), LA06 (15 * 5), LA10 (15 * 5), LA11 (20 * 5), LA13 (20 * 5), LA21 (15 * 10), LA22 (15 * 10), LA27 (20 * 10), and LA37 (15 * 15). X * Y denotes a X job scale with Y machines. Among the 40 test instances in LA, these ten most comprehensively represent the production scale of the jobshop in most cases. We can selected one or two calculation instances from each production scale test and compare their results. This method can avoid accidents caused by huge differences in the solution process and increase the accuracy of the scheduling results. By controlling a certain variable of the production scale (such

TABLE 7. Scheduling results of DQN and DRs.

LA	Size	f	DQN	LOPNR	LPT	LWKR	MOPNR	MWKR	Random	S/WKR	SLACK	SPT	CR
LA01	10*5	$f = 0.5$	**395.5**	560.5	627	507.5	579	553.5	695.5	580	433.5	509.5	580
		$f = 1.0$	**354**	426	516	365	426	437	386	384	372	391	384
		$f = 1.5$	**247**	415	405	322	309.5	320.5	373	348	302	372.5	348
LA02	10*5	$f = 0.5$	**413**	649.5	624	504	616.5	628.5	733	494	489	823.5	494
		$f = 1.0$	341	555	427	**311**	499	511	424	353	356	729	353
		$f = 1.5$	**232.5**	460.5	430	277.5	381	393.5	339	345	304	634.5	245
LA06	15*5	$f = 0.5$	**467**	559	751.5	783	762.5	766.5	815.5	631	503.5	828	631
		$f = 1.0$	**420**	435	631	654	644	631	581	463	497	728	433
		$f = 1.5$	**236**	335	510.5	525	495.5	408	498	396	318.5	628	298
LA10	15*5	$f = 0.5$	**440.5**	629.5	734.5	488.5	753.5	806	609.5	712.5	513.5	871.5	712.5
		$f = 1.0$	**425**	531	598	534	668	692	523	661	508	786	561
		$f = 1.5$	**203.5**	432.5	461.5	401.5	582.5	578	497.5	389.5	344.5	700.5	395.5
LA11	20*5	$f = 0.5$	**513.5**	545.5	785	810	822.5	584	756.5	564	545	764	564
		$f = 1.0$	**443**	480	674	699	704	549	671	644	590	635	645
		$f = 1.5$	**220.5**	374.5	563	588	585.5	384	466.5	440	437.5	506	540
LA13	20*5	$f = 0.5$	**530.5**	584.5	852.5	668	856.5	850.5	738.5	718.5	611.5	779.5	718.5
		$f = 1.0$	**516**	547	733	577	766	731	618	546	578	660	546
		$f = 1.5$	**253.5**	459.5	613.5	392.5	675.5	611.5	488.5	412.5	372.5	540.5	312.5
LA21	15*10	$f = 0.5$	**554**	737.5	720.5	634	713.5	600	660.5	667	657	702	667
		$f = 1.0$	**435**	532	485	446	452	559	540	480	487	484	580
		$f = 1.5$	**289**	426.5	449.5	358.5	396.5	418	488	391	396.5	372.5	291
LA22	15*10	$f = 0.5$	**578**	698.5	748	681	635.5	725	691	684.5	699.5	642.5	684.5
		$f = 1.0$	**443**	501	553	475	484	519	493	594	534	450	489
		$f = 1.5$	**273**	303.5	358	469	332.5	313	341.5	484.5	378.5	357.5	484.5
LA27	20*10	$f = 0.5$	**601.5**	717.5	633.5	704	689	647.5	677	641.5	696.5	718.5	641.5
		$f = 1.0$	**526**	537	588	488	579	586	486	532	564	605	532
		$f = 1.5$	**306.5**	356.5	482	412	483.5	324.5	336.5	450.5	391	491.5	450.5
LA37	20*10	$f = 0.5$	**583.5**	568.5	633.5	649.5	680	653.5	642.5	596.5	613.5	598.5	596.5
		$f = 1.0$	**475**	517	482	501	642	617	622	511	559	547	556
		$f = 1.5$	**316**	468	454	435	579	563	551	449	513	498	431

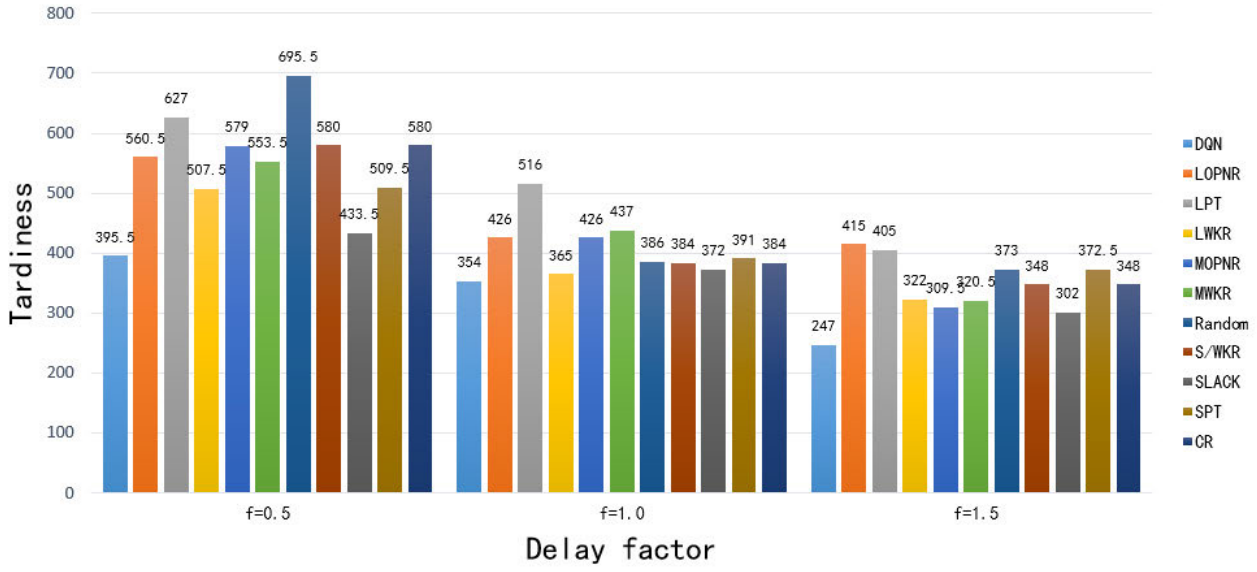


FIGURE 4. Comparison of the performance indicator obtained by DQN and DRs in LA01.

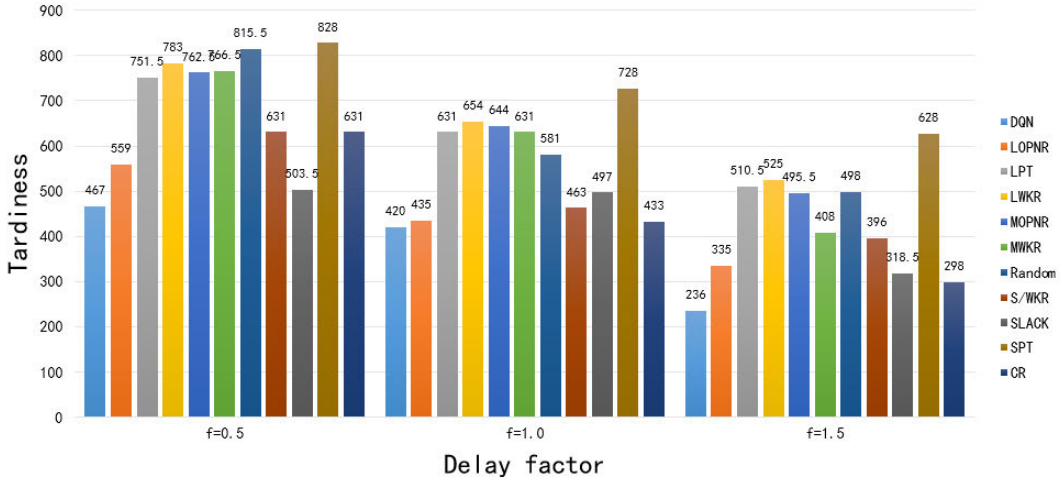


FIGURE 5. Comparison of the performance indicator obtained by DQN and DRs in LA06.

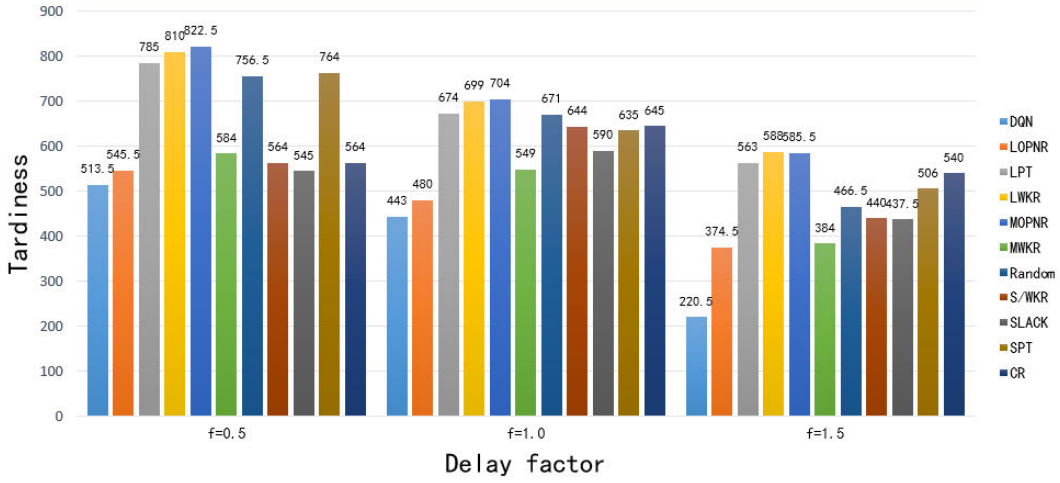


FIGURE 6. Comparison of the performance indicator obtained by DQN and DRs in LA11.

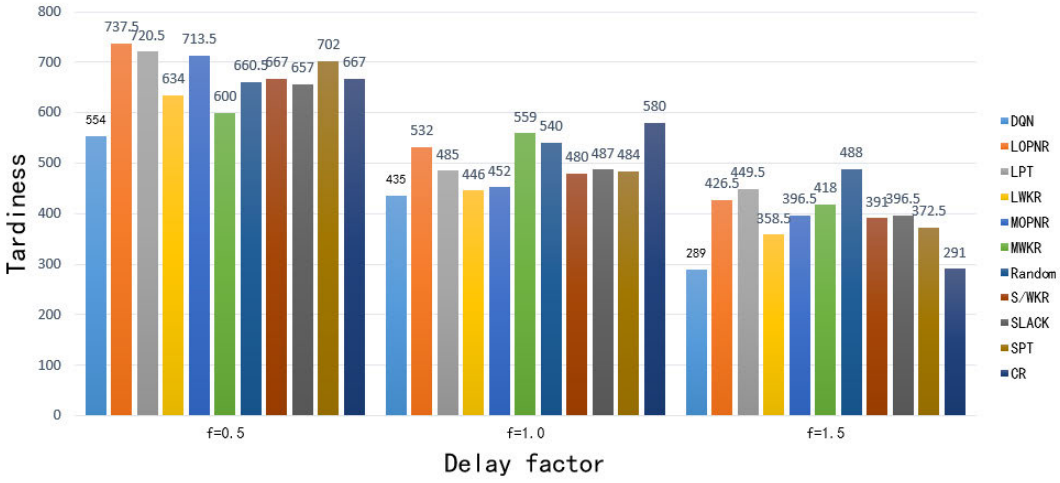


FIGURE 7. Comparison of the performance indicator obtained by DQN and DRs in LA21.

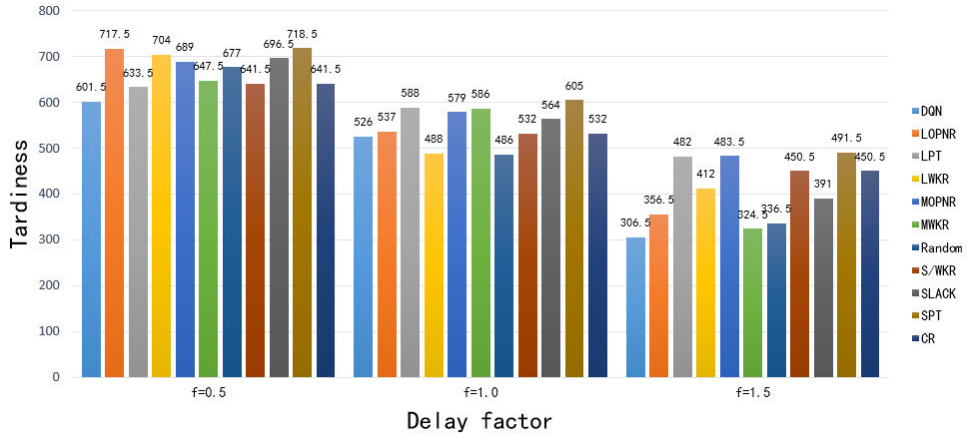


FIGURE 8. Comparison of the performance indicator obtained by DQN and DRs in LA27.

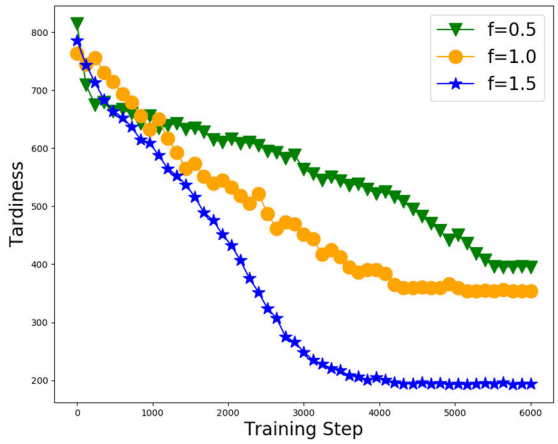


FIGURE 9. Different f values affect scheduling results of DQN in LA01.

as the same number of machines or different numbers of jobs), it is possible to specifically analyze different calculation results relevant to different variables. We can further analyzed the iterative process of the proposed scheduling algorithm and its convergence accordingly. The same 10 DRs (Table 5) were used in each instance as the action group. The parameters we used in this experiment are listed in Table 6.

B. SIMULATION ANALYSIS

1) SIMULATION SCHEME 1

The value of the delay factor f reflects the urgency of the jobs to be processed. A smaller f value indicates higher priority of the job to be processed. The value range of f is [0-1.5]. We need to select some representative f values in this range to test its impact on the scheduling performance indicator. In this study, we used 10 test instances to determine difference between the use of a single rule and the use of the DQN algorithm in terms of scheduling results with different f values. The simulation results are shown in the Table 7. The results show that no single dispatching rule provides optimal scheduling performance in all production environments. In effect, the DQN algorithm learns the correct and most efficient policy for selecting the appropriate DRs at different dispatching moments. The DQN algorithm has a relatively low delay time in almost all situations. Overall, it showed very strong performance suggesting that is effective and readily applicable.

The results on LA01, LA06, LA11, LA21, and LA27 were formulated as histograms for an intuitive comparison as shown in Figs. 4-8. The DQN algorithm's ability to dynamically use 10 DRs in dealing with the DJSP was better than using a single DR as evidenced by the f values.

In order to analyze the impact of the delay factor f on the performance indicator more intuitively, LA01, LA06, LA11, LA21and LA27 are taken as examples, f is set to 0.5, 1, 1.5, and the DQN iteration curves are obtained in the Fig. 9, Fig. 10, Fig. 11, Fig. 12 and Fig. 13 respectively. It can be known from these figures that with the gradual increase of f, the urgency of processing the jobs on the machine gradually increases, and the performance indicator gradually decreases. A larger f value can make the processing process of each job more urgent, effectively reducing the delay time and thereby improving the production efficiency of the workshop. The iterative trajectory of DQN algorithm will fluctuate to a certain extent, the fluctuation of the curve is caused by repeated selection of different DRs by the action selection strategy, which reflects the dynamics of DQN algorithm in selecting dispatching rules in different states. The fluctuation can also help DQN algorithm get the better scheduling results. When the curve converges, it means that DQN algorithm has learned the scheduling policy of selecting DRs in different states.

2) SIMULATION SCHEME 2

LA01, LA06, and LA11 were observed with the same number of processing machines and f of 1.0. The number of jobs was set to 10, 15, and 20. Fig. 14 shows that when the number of machines is the same and the number of jobs gradually increases, the performance indicators gradually increases as

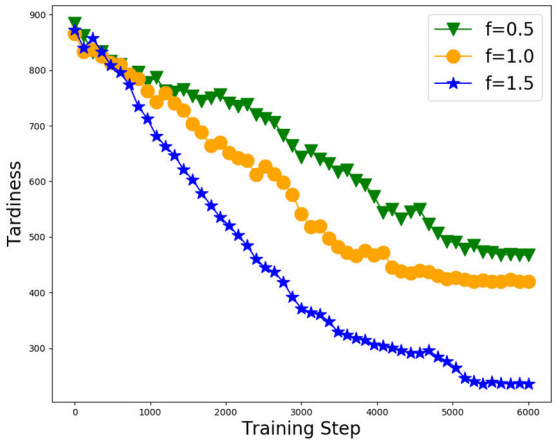


FIGURE 10. Different f values affect scheduling results of DQN in LA06.

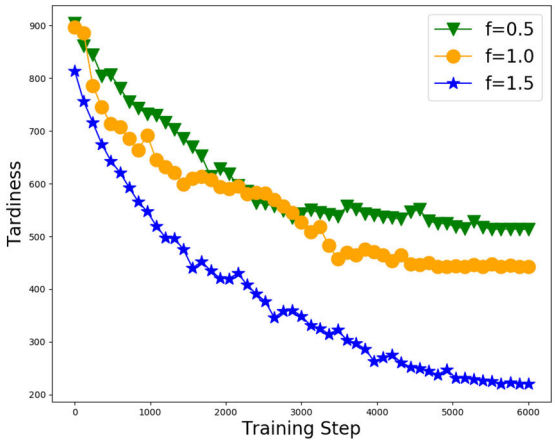


FIGURE 11. Different f values affect scheduling results of DQN in LA11.

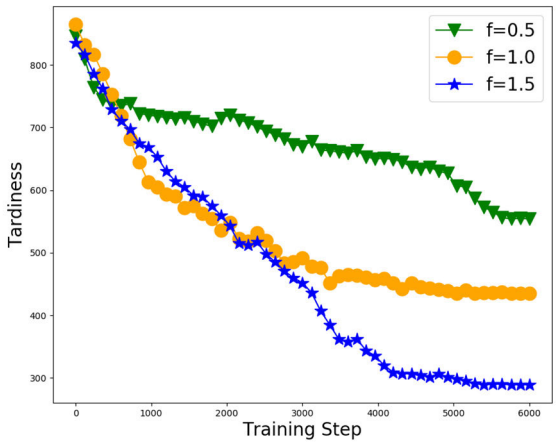


FIGURE 12. Different f values affect scheduling results of DQN in LA21.

well. The DQN algorithm takes longer to converge as the number of jobs increases. The reason is that when the number of processing machines is the same and there are more jobs to be processed, the average processing resources of each job decrease, which increases the delay time of the processing system.

TABLE 8. State space of QL algorithm.

	State Interval	Action 1	Action 2
0	$EAST(t) \leq 0$	$Q(0,0)$	$Q(0,1)$
1	$0 \leq EAST(t) < EART(t) * h$	$Q(1,0)$	$Q(1,1)$
2	$h * EART(t) \leq EAST(t) < EART(t) * 2h$	$Q(2,0)$	$Q(2,1)$
3	$2h * EART(t) \leq EAST(t) < EART(t) * 3h$	$Q(3,0)$	$Q(3,1)$
4	$3h * EART(t) \leq EAST(t) < EART(t) * 4h$	$Q(4,0)$	$Q(4,1)$
5	$4h * EART(t) \leq EAST(t)$	$Q(5,0)$	$Q(5,1)$

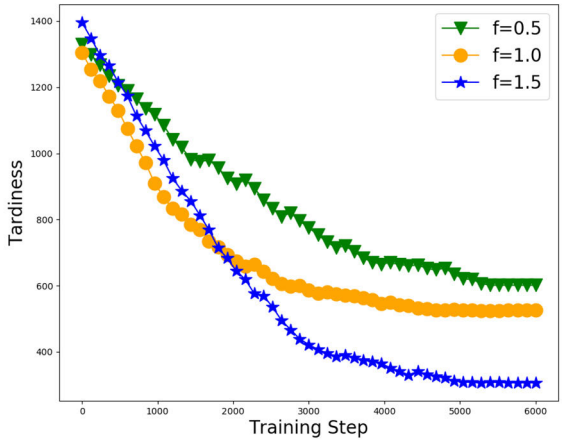


FIGURE 13. Different f values affect scheduling results of DQN in LA27.

3) SIMULATION SCHEME 3

We set f to 0.5 and tested LA01 and LA06 while adjusting the number of neurons n_h in each hidden layer of the DNN from 50 to 100. As shown in Figs. 15 and 16, the DNN has better fitting ability when there are more neurons in the hidden layer. Because an increase in the number of neurons in the hidden layer gives the DNN have better fitting ability and higher-order nonlinear function mapping ability; it can then characterize high-dimensional and continuous input data, which enhances the computing power of the DQN algorithm. Better convergence also results in better performance indicator.

4) SIMULATION SCHEME 4

We compared the DQN algorithm with the traditional QL algorithm without DNN in solving the DJSP. The performance indicator of the two algorithms are set to minimize the delay time and the action group is set to a rule base composed of 10 DRs (Table 5). We set the state space of the QL algorithm to the form shown in Table 8; its state variables were all the same parameters used in the DQN algorithm. The delay factor f was set to 0.5 for both algorithms and other QL algorithm parameters were consistent with a previous study [33].

In order to better control the variables, the action selection strategies of the two algorithms to the "Softmax" strategy as mentioned above to further explore their performance. We focused on the effectiveness and superiority of DNN within the DQN algorithm. As shown in Table 9, the performance indicators of the DQN algorithm are significantly better than those of the traditional QL algorithm. As shown in Table 9 and Figure 17, DNN as an input state fitter has

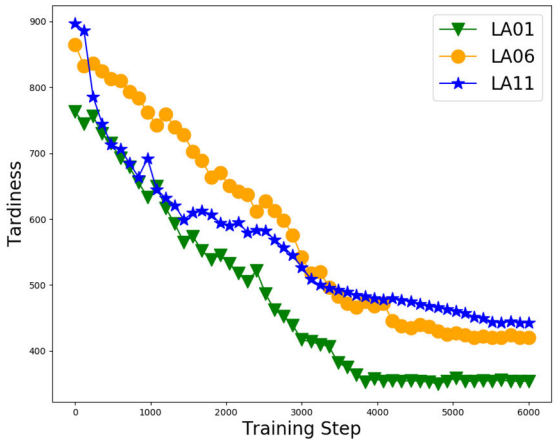


FIGURE 14. Number of jobs affect the performance indicator with same number of machines.

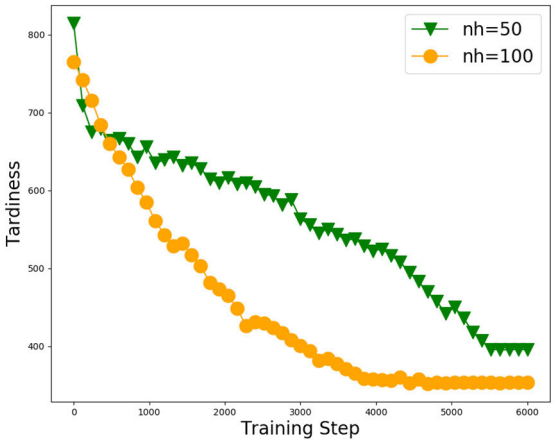


FIGURE 15. Number of neurons affects hidden layer of DQN on LA01.

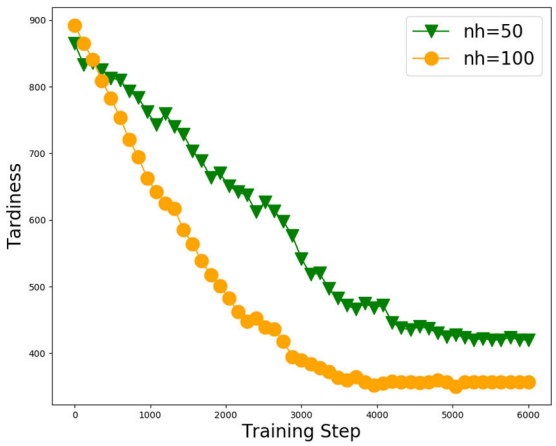


FIGURE 16. Number of neurons affects hidden layer of DQN on LA06.

a stronger ability to fit large-scale continuous state input data. The DQN algorithm can manage high-dimensional and continuous input states, can describe the production state of the production system at various moments in a comprehensive and detailed manner, and effectively avoids the ''dimensional disaster'' that often occurs in QL algorithms. As opposed to

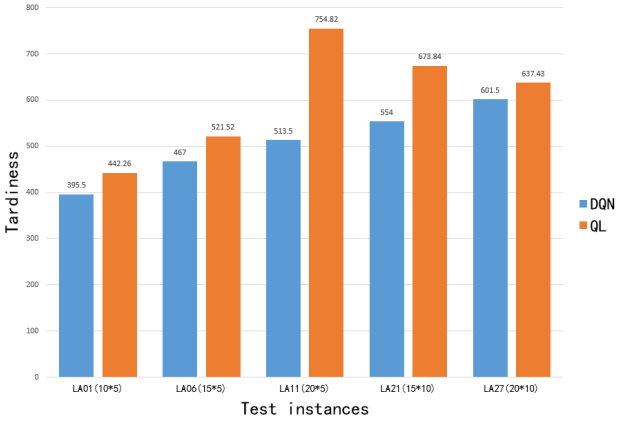


FIGURE 17. Comparison of the performance indicator obtained by DQN and QL respectively.

TABLE 9. Comparison of performance indicators obtained by DQN and QL.

	Size	DQN	QL
LA01	10*5	**395.5**	442.26
LA06	15*5	**467**	521.52
LA11	20*5	**513.5**	754.82
LA21	15*10	**554**	673.84
LA27	20*10	**601.5**	637.43

TABLE 10. Comparison of performance indicators obtained by DQN and other baseline algorithms.

	Size	DQN	HIA	GA	Tabu	IPSO
LA01	10*5	**395.5**	733	715	762	736.5
LA06	15*5	**467**	841	773	854	823
LA11	20*5	**513.5**	1176	1176	1179	1025
LA21	15*10	**554**	922	878	922	936
LA27	20*10	**601.5**	943	957	1028	1147

the traditional QL algorithm, only a few state variables are necessary to describe the relatively limited state space. The ERVM in the DQN algorithm effectively solves the strong correlation between the training samples, which allows the DQN algorithm to gradually converge during training.

5) SIMULATION SCHEME 5

This experiment also tested the proposed DQN algorithm in comparison to other representative intelligent optimization methods: the HIA [13] algorithm that combines PSO and AIS, the GA algorithm [34], the Tabu algorithm [35], and the IPSO algorithm [36]. Each was operated on LA01, LA06, LA11, LA21, and LA27 as test instances. The relevant parameters of the DQN algorithm were the values mentioned above; f was set to 0.5 and other algorithms were programmed with the same parameters of their original references. Table 10 and Figure 18 show that the performance of the DQN algorithm improved by an average of 44.59%, 42.94%, 46.25%, and 45.56% compared to HIA, GA, Tabu, and IPSO, respectively. This generally reflects the problem of DNN in dealing with complex state input data. The strong non-linear fitting ability of the DQN algorithm highlights its superior ability to solve DJSP.

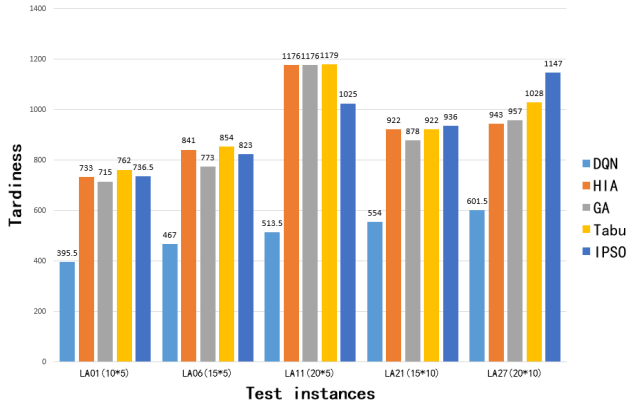


FIGURE 18. Comparison of the performance indicator obtained by DQN and other baseline algorithms.

VIII. CONCLUSION

The real-time dynamic manufacturing environment is highly challenging to manage. Intelligent jobshop techniques have been developed to meet today's needs. A dynamic scheduling algorithm for solving the DJSP based on the DQN algorithm was established in this study. We used n state characteristic functions with continuous values between [0, 1] to describe the input state at each scheduling moment. We designed a two-layer DQN algorithm containing the target network and prediction network as the main learning strategy. A new selection strategy based on the "Softmax" function was used to enhance the randomness of the algorithm in selecting actions at the early stage of the learning strategy. The results of our experiment show that the DQN algorithm, by selecting different DRs at different scheduling times, has stronger performance indicators than those obtained by using a single DR. The proposed method also outperformed the QL algorithm using Q-tables to store data.

The proposed scheduling approach still has some shortcomings. Other unpredictable interference factors that may appear in the actual production process should be considered in the future. It is also likely possible improve the function of the DQN algorithm to output Q-values of each action equal to the number of candidate actions for each machine at each scheduling moment. An action selection strategy may be used to select an action to act on each machine as well. This could guide each machine to complete production and processing tasks autonomously, thereby enhancing the production efficiency of the entire workshop.

REFERENCES

[1] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA J. Comput.*, vol. 3, no. 2, pp. 149–156, 1991.
[2] J. Chen, C. Du, Y. Zhang, P. Han, and W. Wei, "A clustering-based coverage path planning method for autonomous heterogeneous UAVs," *IEEE Trans. Intell. Transp. Syst.*, early access, Mar. 24, 2021, doi: 10.1109/TITS.2021.3066240.
[3] J. Chen, C. Du, P. Han, and X. Du, "Work-in-progress: Non-preemptive scheduling of periodic tasks with data dependency upon heterogeneous multiprocessor platforms," in *Proc. IEEE 40th Real-Time Syst. Symp. (RTSS)*, Dec. 2019, pp. 540–543, doi: 10.1109/RTSS46320.2019.00059.
[4] J. Chen, C. Du, F. Xie, and B. Lin, "Scheduling non-preemptive tasks with strict periods in multi-core real-time systems," *J. Syst. Archit.*, vol. 90, pp. 72–84, Oct. 2018, doi: 10.1016/j.sysarc.2018.09.002.
[5] J. R. Jackson, "Scheduling a production line to minimize maximum tardiness," *Manage. Sci. Res. Projects*, vol. 43, 1955.
[6] S. S. Panwalkar and W. Iskander, "A survey of scheduling rules," *Oper. Res.*, vol. 25, no. 1, pp. 45–61, 1977.
[7] A. Jones, L. C. Rabelo, and A. T. Sharawi, "Survey of job shop scheduling techniques," in *Wiley Encyclopedia of Electrical and Electronics Engineering*. Hoboken, NJ, USA: Wiley, 1999, pp. 1–12.
[8] J. T. Naidu, "A note on a well-known dispatching rule to minimize total tardiness," *Omega*, vol. 31, no. 2, pp. 137–140, Apr. 2003, doi: 10.1016/S0305-0483(03)00020-3.
[9] M. Durasević and D. Jakobović, "A survey of dispatching rules for the dynamic unrelated machines environment," *Expert Syst. Appl.*, vol. 113, pp. 555–569, Dec. 2018, doi: 10.1016/j.eswa.2018.06.053.
[10] O. Holthaus and C. Rajendran, "Efficient dispatching rules for scheduling in a job shop," *Int. J. Prod. Econ.*, vol. 48, no. 1, pp. 87–105, Jan. 1997.
[11] M. Jahangirian, T. Eldabi, A. Naseer, L. K. Stergioulas, and T. Young, "Simulation in manufacturing and business: A review," *Eur. J. Oper. Res.*, vol. 203, no. 1, pp. 1–13, May 2010, doi: 10.1016/j.ejor.2009.06.004.
[12] R. Qing-Dao-Er-Ji and Y. Wang, "A new hybrid genetic algorithm for job shop scheduling problem," *Comput. Oper. Res.*, vol. 39, no. 10, pp. 2291–2299, Oct. 2012, doi: 10.1016/j.cor.2011.12.005.
[13] H.-W. Ge, L. Sun, Y.-C. Liang, and F. Qian, "An effective PSO and AIS-based hybrid intelligent algorithm for job-shop scheduling," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 38, no. 2, pp. 358–368, Mar. 2008, doi: 10.1109/TSMCA.2007.914753.
[14] M. A. Tawfeek, A. El-Sisi, A. E. Keshk, and F. A. Torkey, "Cloud task scheduling based on ant colony optimization," in *Proc. Int. Conf. Comput. Eng. Syst.*, Nov. 2013, pp. 64–69.
[15] D. L. Poole and A. K. Mackworth, *Artificial Intelligence: Foundations of Computational Agents*. Cambridge, U.K.: Cambridge Univ. Press, 2010.
[16] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *Robotica*, vol. 17, no. 2, pp. 229–235, 1999.
[17] M. Dorigo and M. Colombetti, "Robot shaping: Developing autonomous agents through learning," *Artif. Intell.*, vol. 71, no. 2, pp. 321–370, Dec. 1994.
[18] M. E. Aydin and E. Öztemel, "Dynamic job-shop scheduling using reinforcement learning agents," *Robot. Auton. Syst.*, vol. 33, nos. 2–3, pp. 169–178, Nov. 2000.
[19] Y.-C. Wang and J. M. Usher, "Learning policies for single machine job dispatching," *Robot. Comput.-Integr. Manuf.*, vol. 20, no. 6, pp. 553–562, Dec. 2004.
[20] W. Bouazza, Y. Sallez, and B. Beldjilali, "A distributed approach solving partially flexible job-shop scheduling problem with a *Q*-learning effect," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 15890–15895, Jul. 2017.
[21] Y.-R. Shiue, K.-C. Lee, and C.-T. Su, "Real-time scheduling for a smart factory using a reinforcement learning approach," *Comput. Ind. Eng.*, vol. 125, pp. 604–614, Nov. 2018.
[22] 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.
[23] 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.
[24] 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.
[25] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.
[26] Y. Wei, L. Pan, S. Liu, L. Wu, and X. Meng, "DRL-scheduling: An intelligent QoS-aware job scheduling framework for applications in clouds," *IEEE Access*, vol. 6, pp. 55112–55125, 2018, doi: 10.1109/ACCESS.2018.2872674.
[27] D. Shi, W. Fan, Y. Xiao, T. Lin, and C. Xing, "Intelligent scheduling of discrete automated production line via deep reinforcement learning," *Int. J. Prod. Res.*, vol. 58, no. 11, pp. 3362–3380, Jan. 2020, doi: 10.1080/00207543.2020.1717008.

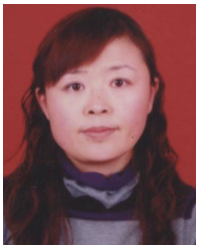
[28] B. Waschneck, A. Reichstaller, L. Belzner, T. Altenmüller, T. Bauernhansl, A. Knapp, and A. Kyek, "Optimization of global production scheduling with deep reinforcement learning," *Procedia CIRP*, vol. 72, pp. 1264–1269, Jan. 2018.
[29] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. 15th ACM Workshop*, Nov. 2016, pp. 50–56.
[30] H. Hu, X. Jia, Q. He, S. Fu, and K. Liu, "Deep reinforcement learning based AGVs real-time scheduling with mixed rule for flexible shop floor in industry 4.0," *Comput. Ind. Eng.*, vol. 149, Nov. 2020, Art. no. 106749, doi: 10.1016/j.cie.2020.106749.
[31] 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.
[32] 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.
[33] Y. Z. Wei and M. Y. Zhao, "Reinforcement learning-based approach to dynamic job-shop scheduling," *Acta Autom. Sinica*, vol. 31, no. 5, pp. 765–771, 2005, doi: CNKI:SUN:MOTO.0.2005-05-016.
[34] F. D. Croce, R. Tadei, and G. Volta, "A genetic algorithm for the job shop problem," *Comput. Oper. Res.*, vol. 22, no. 1, pp. 15–24, Jan. 1995.
[35] E. Nowicki and C. Smutnicki, "A fast taboo search algorithm for the job shop problem," *Manage. Sci.*, vol. 42, no. 6, pp. 797–813, Jun. 1996.
[36] H. Yang, "Optimization of job shop scheduling based on improved particle swarm algorithm," *Mech. Des. Manuf. Eng.*, vol. 48, no. 2, pp. 77–80, 2019, doi: CNKI:SUN:JXZZ.0.2019-02-019.

YEJIAN ZHAO was born in Liaoyang, Liaoning, China, in 1995. He received the bachelor's degree from the School of Physics and Electronic Engineering, Hainan Normal University, Hainan, in 2017. He is currently pursuing the master's degree with Shenyang University of Technology. His research direction in postgraduate stage is workshop scheduling. His main research interests include jobshop scheduling, machine learning algorithms, and enterprise integrated automation.

YANHONG WANG received the B.S. and M.S. degrees in electrical engineering and automation from Shenyang University of Technology, Shenyang, China, in 1989 and 1992, respectively, and the Ph.D. degree in control theory and control engineering from Jilin University, Changchun, China, in 2003. She is currently a Professor with the School of Information Science and Engineering, Shenyang University of Technology. Her research interests include production planning and scheduling in manufacturing enterprises, and intelligent factory and application of artificial intelligence in manufacturing enterprises.

YUANYUAN TAN received the B.S. degree in information and computing science from Bohai University, Jinzhou, China, in 2007, and the M.S. and Ph.D. degrees from the College of Information Science and Engineering, Northeastern University, Shenyang, China, in 2009 and 2013, respectively. She is currently an Associate Professor with the School of Information Science and Engineering, Shenyang University of Technology. Her research interests include scheduling and production plan in steelmaking, continuous casting and hot rolling, and intelligent optimization algorithm. She has published over several journal articles and conference proceedings papers in the above research areas.

JUN ZHANG was born in Shenyang, Liaoning, China, in 1986. He received the B.S. degree in automation from Shenyang University of Technology, Shenyang, in 2008, and the M.S. and Ph.D. degrees in control theory and control engineering from Northeastern University, Shenyang, China, in 2010 and 2015, respectively. He is currently an Associate Professor with the School of Artificial Intelligence, Shenyang University of Technology. His current research interest includes modeling, optimization, and control of complicated industrial production processes.



HONGXIA YU received the Ph.D. degree from Shenyang Institute of Automation, Chinese Academy of Sciences, in 2012. She is currently a Master Supervisor in control science and control engineering with Shenyang University of Technology. Her current research interests include intelligent optimization, and state monitor and control of induction motor.