processes

MDPI

Article

Combining Reinforcement Learning Algorithms with Graph Neural Networks to Solve Dynamic Job Shop Scheduling Problems

Zhong Yang, Li Bi * and Xiaogang Jiao *

College of Information Engineering, Ningxia University, Yinchuan 750021, China; yangz@stu.nxu.edu.cn
* Correspondence: billy68@nxu.edu.cn (L.B.); jiaoxg@nxu.edu.cn (X.J.)

Abstract: Smart factories have attracted a lot of attention from scholars for intelligent scheduling problems due to the complexity and dynamics of their production processes. The dynamic job shop scheduling problem (DJSP), as one of the intelligent scheduling problems, aims to make an optimized scheduling decision sequence based on the real-time dynamic job shop environment. The traditional reinforcement learning (RL) method converts the scheduling problem with a Markov process and combines its own reward method to obtain scheduling sequences in different real-time shop states. However, the definition of shop states often relies on the scheduling experience of the model constructor, which undoubtedly affects the optimization capability of the reinforcement learning model. In this paper, we combine graph neural network (GNN) and deep reinforcement learning (DRL) algorithm to solve DJSP. An agent model from job shop state analysis graph to scheduling rules is constructed, thus avoiding the problem that traditional reinforcement learning methods rely on scheduling experience to artificially set the state feature vectors. In addition, a new reward function is defined, and the experimental results prove that our proposed reward method is more effective. The effectiveness and feasibility of our model is demonstrated by comparing with general deep reinforcement learning algorithms on minimizing the earlier and later completion time, which also lays the foundation for solving the DJSP later.

Keywords: smart factory; dynamic job shop scheduling; Markov process; graph neural network; deep reinforcement learning

Citation: Yang, Z.; Bi, L.; Jiao, X. Combining Reinforcement Learning Algorithms with Graph Neural Networks to Solve Dynamic Job Shop Scheduling Problems. *Processes* **2023**, *11*, 1571. https://doi.org/10.3390/pr11051571

Academic Editor: Luis Puigjaner

Received: 11 April 2023
Revised: 18 May 2023
Accepted: 19 May 2023
Published: 21 May 2023

1. Introduction

In recent years, the smart manufacturing boom has swept across China, becoming the most important initiative to promote the national strategy of "Made in China 2025". Among them, smart factory, as an important practice area of smart manufacturing, has attracted wide attention from manufacturing enterprises and high attention from governments at all levels. All major economies around the world are vigorously promoting the revival of manufacturing industry. Under the boom of Industry 4.0, Industrial Internet, Internet of Things and Cloud Computing, many excellent manufacturing enterprises around the world have carried out smart factory construction practices. DJSP as a representative problem model in the production scheduling industry is proven to be a np-hard problem of most cases [1]. As the modern job shop scheduling problem becomes more complex, the time cost of the exact solution to the problem is too large, making it difficult to apply the solution to a wide range of applications in practical production scheduling. Therefore, the methods of obtaining an approximate solution to the job shop scheduling problem have attracted many scholars. First of all, the rule-based scheduling [2,3], which is designed based on practical experience, is able to respond quickly to the dynamic environment of the job shop with high migration [4]. However, its solution quality cannot be guaranteed, and the literature [5] demonstrates the inability to obtain a locally optimal solution using a single rule-based scheduling, while the super heuristic algorithm [6] can optimize this problem

better. Zhang et al. [7] used genetic planning algorithm to encode the scheduling decision method, made a selection of dominant individuals in the population by value function and updated the local optimal solution by gene iteration. It is finally proved that the method outperforms the regular scheduling in terms of solution quality. However, as the problem size increases, the algorithm tends to fall into local optimal solutions and has poor migration. A new optimal search is needed for the new problem model, which requires a long scheduling time. In summary, it is difficult for heuristic algorithms to achieve better solutions to both solution quality and speed [8].

Recently, the rise of reinforcement learning (RL) algorithm's ability to explore unknown environments in many fields such as robot control and game competition [9] has attracted more and more scholars to try to use it for solving DJSP. Zhao et al. [10] used deep Q-network (DQN) algorithm for the DJSP and verified that the DQN algorithm is significantly superior to other rule-based scheduling methods and heuristic scheduling methods. This proof effectively shows that reinforcement learning has a significant advantage in dealing with dynamic scheduling problems. Luo et al. [11] also used the DQN to solve the flexible DJSP and verified that the proposed algorithm has a stronger superiority in handling continuous states. This also makes reinforcement learning a hot topic in solving scheduling problems. Although the RL algorithm can well compensate the shortcomings of rule-based scheduling and super heuristic algorithms in dealing with dynamic scheduling problems, there are still some problems. Wang et al. [12] proposed that a reasonable definition of the state space, action space, payoff function and policy function of an agent is the core of reinforcement learning for solving job shop scheduling problems. Traditional RL methods rely on the state features of manually designed models, so they need to refer to relevant domain scheduling experiences. In addition, there are differences in the selection of features due to different job shop scheduling environments, which can also affect the migration of the model. The graph neural network has good graph features extraction capability [13,14], and its introduction also makes it possible to optimize the problems existing on job shop state designs.

With the rise of graph neural networks in many fields related to molecular structure, image processing, etc. [15–17], it has also given rise to thoughts about its application of scheduling problems. Zeng et al. [18] first applied graph representation learning to scheduling problems. They used graphs to describe the state space of the scheduling problem, selected graph features by an attention mechanism to speed up training, and used the D3QPG network to obtain output actions. In this experiment, the problem that the state of the traditional reinforcement learning algorithm depends on the manual is broken. The graph representation is introduced into reinforcement learning to solve the scheduling problem, and the effectiveness of the scheme is verified. Junyoung Park et al. [19] used a combination of GNN and RL in solving the JSP. Firstly, a graphical substitution for the environment state is used; then, the features are extracted by GNN, and finally, the standard state description vector is obtained. Then, the optimization of action selection is performed by RL method. Finally, it proves the superiority of the constructed model in solving scheduling problems and also makes GNN more widely concerned and explored on scheduling issues. Feature extraction by GNN can effectively avoid the problem of relying on scheduling experience of the shop state feature values in the traditional RL algorithm and also enhance the network model migration capability in solving problems of the same scale [14,20]. However, all the above schemes only apply GNN to prove the efficiency of the model through experiments, while we build on this and also demonstrate the superiority of GNN as a feature extraction method compared to manual extraction. The contributions of this paper are as follows:

1. A GNN combined with DRL model was constricted for solving DJSP with random job arrivals;
2. A scheduling pool to store the processes that satisfy the following two conditions in the current job shop environment: (1) the process being processed is currently in a executable state; (2) the machine processing the process is also in an idle state;
3. A new reward function is designed to enhance the learning ability of the algorithm for solving the target.

The rest of this paper is organized as follows: in Section 2, we introduce the dynamic scheduling problem perturbation events, such as dynamic arrival of jobs and random weights, as well as the mathematical model and constraints; next, we introduce our DJSP solution model, including GNN's disjunctive graph feature extraction and the DRL algorithm processes combined with GNN in Section 3; in Section 4, we use a related algorithm to compare with our model on minimizing the earlier and later completion time; finally, Section 5 draws the conclusions.

2. Problem Modeling

2.1. Problem Introduction

DJSP is based on the classic JSP and takes into account a variety of disturbing factors that occur during job processing. In this paper, random arrival jobs with weights are used as disturbing factors. In the DJSP of this paper, we are provided with n successively arriving jobs $J = \{J_1, J_2, \ldots, J_n\}$ to be processed on m machines $M = \{M_1, M_2, \ldots, M_m\}$, each of which has to go through multiple different processing sequences, and the processing time OP_{ij} of each process and the processing order constraint on each machine are known. The arrival time of each job is A_i, and the weight of the job is w_i. DJSP should satisfy the assumptions as follows:

(1) All machines are in an executable state at the start of scheduling, and machine failures are not considered;
(2) Multiple processes for the same job need to be processed sequentially in the process order;
(3) Processes for individual jobs must be processed on designated machines and have different priorities between different jobs to account for the degree of urgency between different jobs;
(4) Processes may not be interrupted once they have begun processing;
(5) No consideration is given to the transfer time of jobs between different machines.

2.2. Construction of Mathematical Model

In order to better realize the influence of job priority weights on the solution objective of minimizing the earlier and later completion time in the dynamic scheduling problem, we define the objective function that makes the higher weight job priority scheduling under the condition of minimizing the earlier and later completion time and the relevant constraint expressions in the problem. The relevant symbols involved in the formula are explained in Table 1:

Table 1. DJSP symbol definitions.

Symbol	Meaning Description
i	Index of jobs, $i = 1, 2, 3, \ldots, n$
j	Index of operations belonging to jobs
O_{ij}	Index of operation j belonging to job i
C_i	The finishing time of job i
D_i	The deadline of job i
M_{ij}	The index of machine set for operation O_{ij}

Table 1. *Cont.*

Symbol	Meaning Description
A_i	The arrival time of Job i
OP_{ij}	The processing time of operation O_{ij}
OP^i_{rest}	The remaining processes of job i
$Start_{ij}$	The starting time of O_{ij}
$State_{ij}$	The state of O_{ij}
W_i	The weight of job i
OP_n	The entire process of all jobs
$State_{Mij}$	The state of M_{ij}

The experiment uses minimizing the job earlier and later completion time as the objective function of the dynamic job shop scheduling problem, which is defined by the following equations:

Objective:

$$T = min(\sum_{i=1}^{n} w_i(|C_i - D_i|)) \tag{1}$$

Subject to:

$$OP_{ij} > 0 \tag{2}$$

$$start_{ij} + OP_{ij} \leq start_{i(j+1)} \tag{3}$$

$$start_{ij} \geq A_i \tag{4}$$

$$state_{M_{ij}} = \begin{cases} 1 & if\ state_{ij} = 1 \\ 0 & else \end{cases} \tag{5}$$

$$D_i \geq A_i + \sum_j OP_{ij} \tag{6}$$

Equation (2) indicates that the operation time of the process must symbolize the actual conditions. Equation (3) indicates that the next process of the same job can only start after the completion of the previous process; Equation (4) indicates that the start time of the job must be after the arrival time; Equation (5) indicates that only one job process can be processed by one machine at the same time; Equation (6) indicates that the cut-off time of the job must be greater than the sum of the arrival time and operation time of the job.

3. Model Construction of Reinforcement Learning Algorithm Combined with GNN

Figure 1 shows the scheduling flow diagram for solving the DJSP model, which demonstrates the learning process of an agent that obtains the scheduling rules in the action space by using the dynamic shop parsing graph as input.

3.1. Graph Neural Network

3.1.1. Characterization of the Graph

The graph mainly consists of node information and the connection relationship between them, which is defined as follows: $G = (V, C \cup D)$. V represents the node information in the graph; C represents the directed line segments connected between different processes of the same job in the graph; D represents the undirected line segments between processes with the same processing machine between different jobs. As shown in Figure 2, the start node and end node processing times are 0, which are used to supplement the graph structure and do not count in the scheduling process. O_{ij} indicates the processes of each job, and the same color is used to describe the processes of the same processing machines. In addition, to better describe the DJSP, we add some feature values for the nodes as follows to form the feature vectors of the nodes and simulate the dynamic environment in real time by the change of feature values.

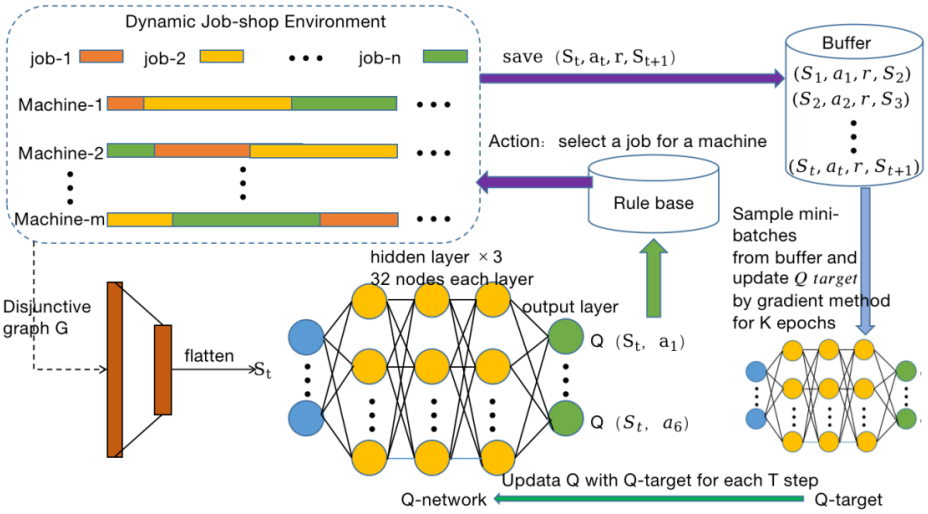

Figure 1. GNN + DRL model training flow chart.

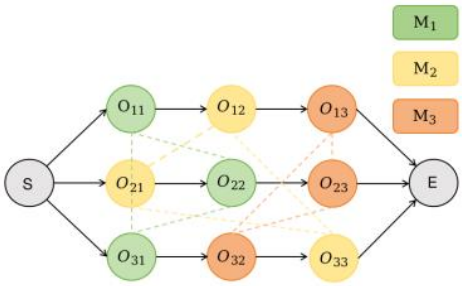

Figure 2. Scheduling environment analysis diagram.

1-i indicates the job number of the current node;
2-j indicates the process number of the current node;
3-M_{ij} indicates the machine number for processing the process;
4-A_i indicates the arrival time of the job;
5-OP_{ij} indicates the operation time of the process;
6-D_i, the latest submission time of the job;
7-$Start_{ij}$, the time when the process is started to be executed;
8-$State_{ij}$ indicates the scheduling status of the process; 0 means not scheduled; 1 means being scheduled; 2 means the process has been scheduled;
9-w_i indicates the urgency of the job.

3.1.2. Advantages and Methods of Job Shop Extraction of Graphs

The state spaces of traditional reinforcement learning algorithms for solving job shop states are mostly defined based on relevant scheduling knowledge. The state feature vector of the job shop is formed by selecting variables that can describe the current state of the shop, such as the current average completion rate of each job, the load rate of each machine, the average remaining completion time of each job and so on [21–24]. However, this method relies on manual scheduling experience, and its migratory nature can be affected for shop floor operations in different environments. In contrast, the graph neural network can effectively extract the node features and the information of its surrounding nodes effectively, and the feature vector of the job shop state is obtained by using the job shop scheduling parse map as the input of the graph neural network; the process can be described by the following equations:

$$h^0 = \sum v_i, fori \in N^v \quad (7)$$

First, the information of all nodes in the graph is obtained to form the node set h^0. As shown in Equation (7), N^v denotes the set of all nodes in the graph. V_i is the feature vector of node $I \in N^v$.

$$h^{t+1} = \sigma\left(D^{\frac{-1}{2}} A D^{\frac{-1}{2}} h^t W^t\right) \tag{8}$$

The information aggregation operation is shown in Equation (8); the node set and the adjacency matrix of the graph are used as inputs, and A denotes the adjacency matrix with self-connections; D is the degree matrix of A; h^t denotes the set of node information of the previous round; W^t is the parameter matrix; and finally, the aggregated information h^{t+1} is output by the activation function σ to obtain the final result after this round of feature aggregation.

As shown in Figure 3a, the relationship matrix is multiplied with the node information matrix to obtain the new feature vector of each node, and the resulting updated node contains the feature information of itself and its surrounding nodes. As shown in Figure 3b, it is the information aggregation process of the node X_2 from t to $t+1$ times. X_2^{t+1} contains X_2^t as well as the information of its neighboring nodes.

$$NormA = \frac{A}{\sqrt{D_i}\sqrt{D_j}} \tag{9}$$

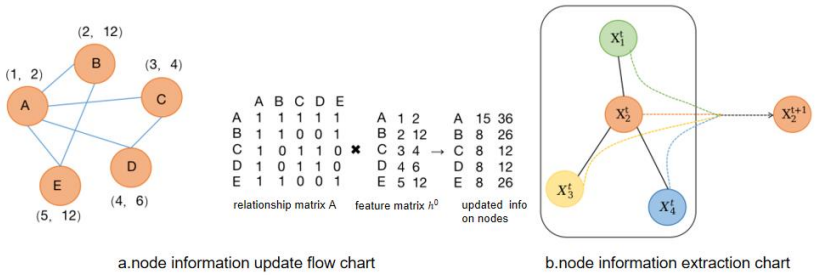

Figure 3. Node update.

When the graph convolution layer is deepened, considering only the constant aggregation of information can lead to excessive information values of newly generated nodes. To avoid changing the original distribution of features after matrix multiplication, it is necessary to make Laplace changes to the relationship matrix A. As shown in Equation (9), it can effectively balance the importance of too-large nodes. In addition, the influence of the number of graph convolution layers on the experimental results needs to be considered. The literature [19] points out that the number of graph convolution layers for extracting image features is generally set to 2 or 3 layers. Finally, the current scheduling state of the job shop is obtained by flattening the output, as shown in Equation (10).

$$s_t = flatten\left(h^{t+1}\right) \tag{10}$$

3.2. Deep Reinforcement Learning

Deep reinforcement learning (DRL) is an algorithm that combines deep learning and reinforcement learning. Traditional reinforcement learning algorithms have achieved good results in areas such as robot control and competitive gaming [9]. However, when first introduced to solve job shop scheduling problems, such as the Q-learning algorithm, it needs to rely on tables to record the rewards corresponding to different actions in different states. As the scale and complexity of the scheduling problem increases, the shortcomings of Q-table are exposed. In the face of the explosive increase in the shop floor state space, the Q-table is difficult to meet the scheduling demand. Therefore, deep learning is introduced into the scheduling problem. The network can be fitted due to the same dimensionality of the input feature vector in different states. The original problem of oversized tables due to defining job shop states in discrete space is solved [10].

3.2.1. Markov Process and Reinforcement Learning

The essence of the solution process of applying reinforcement learning algorithm to the shop scheduling problem is the continuous interaction between the agent and the shop environment. The agent makes a decision to choose the process according to the current job shop environment, while the environment rewards the agent in time based on its behavioral feedback. The final process with the goal of the agent learning to maximize the reward is shown in Figure 4. S_t, A_t and R_t represent the status, action and reward of the current moment, respectively.

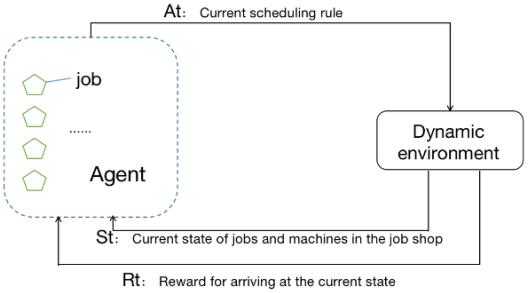

Figure 4. Reinforcement learning flow chart.

The process can be modeled as a Markov decision process with a 5-tuple (S, A, P, R, γ). S represents the current state of the job shop scheduling environment; A represents the action executed by the intelligence in the current state; P represents the probability of moving from the current state to the next state, and R represents the reward received by the current intelligence after executing action A. γ is the discount factor, which is used to play a weakening effect on the visionary reward. The more backward action has less influence on the decision in the current state—therefore, the greater the discount of its reward. The discount factor usually takes values between 0 and 1. The intelligence interacts with the environment according to an initial policy, and at each decision point, the intelligence observes the current state $S_t \in S$ and executes an action $A_t \in A$ according to its own strategy. Then, it enters a new state S_{t+1} and a new state transfers probability P_{t+1} and receives a timely reward R_t. The goal of the intelligence is to find the optimal strategy π_* that maximizes the sum of the expected long-term reward values of the action A in state S and is measured by the action valued function, and the strategy is solved as in Equation (11).

$$\mathrm{Q}_{\pi_*}(S, A) = \max \mathrm{Q}_{\pi}(S, A) \tag{11}$$

3.2.2. Double Deep Q Network

Double Deep Q Network (DDQN) is proposed on the basis of Deep Q Network (DQN). The traditional DQN method has only one Q-network for learning the value function of each action, and the update formula is Equation (12).

$$\mathrm{Q}(S_t, A_t) \leftrightarrow r_t + \gamma \max \mathrm{Q}(S_{t+1}, A_{t+1}) \tag{12}$$

$\mathrm{Q}(S_t, A_t)$ denotes the current action value function, and the update of the Q network is each time to select the action corresponding to the optimal reward in the current state, so the final learned prediction of the Q network is large compared to the general condition. DDQN uses two Q-networks: learning-Q-network and target-Q-network ($\mathrm{Q_t}$).

$$\mathrm{Q}(S_t, A_t) \leftrightarrow r_t + \gamma \mathrm{Q_t}(S_{t+1}, \mathrm{argmax}\mathrm{Q}(S_{t+1}, A_{t+1})) \tag{13}$$

As shown in Equation (13), there are two Q functions: Q and $\mathrm{Q_t}$. When the Q function selects the action corresponding to the maximum reward, the resulting action value is closer to the actual situation by the network computation that is not updated for the time being, compared to the Q network. This design better alleviates the problem that the traditional

DQN algorithm has a large network output due to the need for a single Q network to do both prediction and learning operations. In this paper, the DQN algorithm is combined with GNN to construct a new reinforcement learning model, and the pseudo-code of the method is shown in Algorithm 1:

Algorithm 1. GNN + DDQN

Initialize replay memory D;
Initialize online network action-value Q with random weights θ;
Initialize target network action-value Q_t with $\theta_t = \theta$;
For episode = 1:E do:
Initialize state $S_1 \leftarrow \mathrm{GNN}(G_1)$
  For t = 1:T do:
    If $p < \varepsilon$:
      select a random action a;
    Else:
      $a = \mathrm{argmax}_a Q(S_t, A_t; \theta)$;
    Execute a, observe the next state: $S_{t+1} \leftarrow \mathrm{GNN}(G_{t+1})$, get the immediate reward r;
    Store (S_t, A_t, r, S_{t+1}) in D;
    Get a random minibatch from D;
    Select the optimal action by Q: $A_j^{\max}\left(S_j;\theta\right) = \mathrm{argmax}_{A_j} Q\left(S_j, A_j;\theta\right)$;
    Update Q as fellow: $y_j = \begin{cases} r; & \text{If for the last step} \\ r_j + \gamma Q_t\left(S_j, A_j^{max}(S_j;\theta);\theta_t\right); & \text{else} \end{cases}$
    Execution of gradient descent: $\left(y_j - Q\left(S_{j+1}, A_{j+1};\theta\right)\right)^2$;
    Each m step, update Q_t with $\theta_t \leftarrow \theta$;
End;

3.2.3. Action Space

Inspired by the selection of general scheduling rules as the action space in the traditional RL algorithm for solving job shop scheduling problems [10,22], we select some rules as the scheduling action space for our intelligence. In addition, to enhance the generalization ability of the model, we introduced the ε-greedy strategy [25,26] to select actions. It defines a decrease in the randomness probability of the model's action selection from 0.6 at the beginning of training to 0.01 at the end of training, thus enhancing the randomness of the model's action selection at the beginning of training and being more favorable to the optimal search of the scheduling problem. The following is the Rule Base:

Dispatching Rule 1: Shortest Processing Time (SPT), the machine gives priority to the process with the shortest processing time.

Dispatching Rule2: Longest Processing Time (LPT), the machine gives priority to the process with the longest processing time.

Dispatching Rule3: First In First Out (FIFO), the machine gives priority to the process with the earliest arrived.

Dispatching Rule4: Last In First Out (LIFO), the machine gives priority to the process with the latest arrived.

Dispatching Rule5: Most Operations Remaining (MOR), the machine gives priority to the process with the most operations remained.

Dispatching Rule6: Least Operations Remaining (LOR), the machine gives priority to the process with the least operations remained.

3.2.4. Reward Function

DDQN is a value-based reinforcement learning algorithm, and the job shop scheduling problem has the following two characteristics: 1. The sum of locally optimal solutions is generally not equal to the global optimum; 2. The scheduling problem can only obtain the earlier and later time of the whole scheduling process after the completion of the last

process. Therefore, the definition of reward needs to be set reasonably with the state at the completion of the last process. The reward function is defined as in Equation (14):

$$r = \begin{cases} -0.01; & unfinished\ dispatch \\ op_n; & finidhed, now_{yt} <= best_{yt} \\ \frac{1}{ln\left(now_{yt} - best_{yt} + e\right)} * op_n; & finished, best_{yt} < now_{yt} < best_{yt} * 1.05 \\ -5; & finished, now_{yt} > best_{yt} * 1.05 \end{cases} \quad (14)$$

OP_n denotes the number of processes; NOW_{yt} denotes the earlier and later completion time at the end of the current round, and $Best_{yt}$ denotes the shortest earlier and later completion time learned in the whole training cycle. In order to prevent the difference between the two from being too large, we use a logarithmic function to deflate them. We introduce a natural constant e to ensure that the denominator is greater than 1. Finally, we use an elite retention strategy [27], which gives a positive reward to solutions with an earlier and later completion time within 105% of the current optimal solution and a negative reward to other solutions so as to enhance the learning of the better solution by the agent.

4. Numerical Experiments

The GNN designed was based on torch_geometric. The experimental data are randomly generated using Poisson distribution under the constraint of meeting the actual production conditions, and two groups of 10×10 and 20×15 are mainly chosen for testing. The settings of some key parameters are based on the experience in the experiments. The hyper parameters in Table 2 were set during the experiments:

Table 2. Hyper parameter setting.

Hyperparameter	Value
Epoch	400
Learning rate	0.001
Experience pool	2000
Qtarget update step	20
Batchsize	10
Q network layers	5
Graph convolution layers	2
Discount factor	0.95
ε-greedy	{0.01~0.6}
Activation function	Tanh()

As shown in Figure 5, the box plots of the fluctuation range of the optimal solution for different GNN convolutional layers in 400 training cycles under the dataset Bs5, it can be seen that when the number of convolutional layers is two, the retrieval range and the search effect of the model on the minimized earlier and later completion time are optimal compared with other numbers of convolutional layers, which also lays the theoretical foundation for the design of the model with two convolutional layers in this paper.

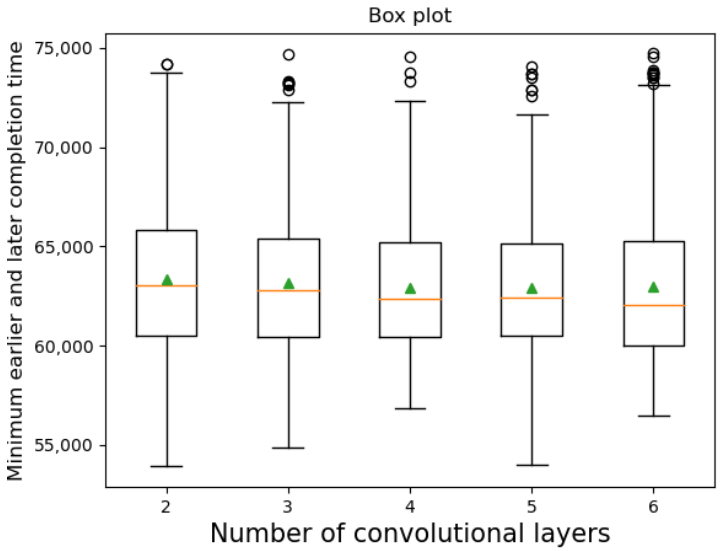

Figure 5. The relationship between the number of layers of the graph neural network and the quality of the solution.

Because DDQN is a value-based reinforcement learning algorithm, defining the appropriate reward function is of pivotal importance to the optimization-seeking effect of the model. The literature [10] also uses job random arrival as a perturbation factor, so we try to compare the two reward functions. The reward function in the literature [10] is shown in Equation (15): $EAST(t)$ denotes the average slack time of all remaining processes; $EART(t)$ denotes the expected average remaining process processing time, and R gives a single-step reward based on the state of the real-time job shop. Our proposed reward function is Equation (14): a global reward method based on the length of the earlier and later completion time. The experimental results are compared on datasets Bs2, Bs3, Bs5, and Bs6, as shown in Figure 6.

$$\begin{aligned} EAST(t) &= \frac{1}{n}\sum_{i=1}^{n}\Big(\sum_{j=1}^{OP^i_{rest}} OP_{ij} - (D_i - t)\Big) \\ EART(t) &= \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{OP^i_{rest}} OP_{ij} \\ R &= \frac{EAST(t)}{EART(t)+0.01} \end{aligned} \tag{15}$$

In the four sets of comparison experiments, the box plots on the left of the images are the experimental results of our proposed reward functions, and the right are the results corresponding to the reward methods in the literature [10]. The distribution states of the earlier and later completion time are obtained by 400 rounds of training. The experiments are only adjusted in the settings of the reward functions. Among them, both Figure 6a,b are the results on the 10 × 10 scale dataset. At this point, the two reward methods have comparable effects in terms of retrieval range and search ability, and both can obtain the local optimal solution, but it can be seen from the average optimal solution that our reward function works better. Figure 6c and d are the experimental results for the 20 × 15 dataset, and it can be seen that our incentive approach has better solution quality when finding the optimal earlier and later completion time on larger datasets. It indicates that the ability of the reward function in the literature [10] to guide the intelligence in the search for the optimum decreases with the proliferation of the types of states in the scheduling shop on a larger scale dataset and also proves the rationality and efficiency of the design of our proposed reward function.

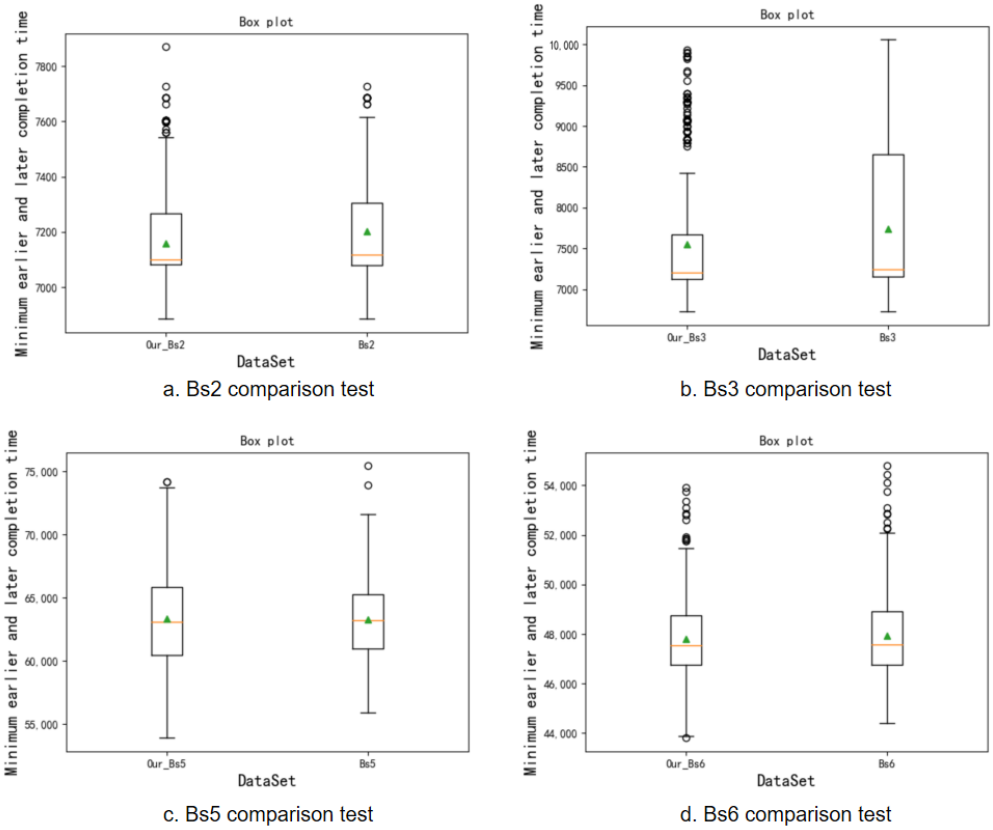

Figure 6. (**a**–**d**) are comparison tests on different datasets.

After an illustration of the efficiency of the reward function, we need to test the learning effect of the model. The experiments focus on minimizing the earlier and later completion time as the goal, taking into account the dynamic arrival of jobs and job weights. The comparison experiments consist of two main groups: first, the six basic rule scheduling methods to solve the DJSP, which are the rules that make up the reinforcement learning action space. Second, the DDQN model is used to solve the DJSP, which has a state space as defined in literature [10]. Table 3 below shows the performance effects of the different methods on each dataset:

Table 3. Minimum earlier and later completion time results.

Dataset	Rule1	Rule2	Rule3	Rule4	Rule5	Rule6	DDQN	GNN + DRL
Bs1 10 × 10	8942	9918	9656	**8482**	**8482**	9747	**8482**	**8482**
Bs2 10 × 10	7118	7081	7211	7100	6893	7448	**6885**	**6885**
Bs3 10 × 10	7170	7594	9072	7170	**6726**	7268	**6726**	**6726**
Bs4 10 × 10	5740	5647	5732	**5400**	5640	5845	**5400**	**5400**
Bs5 20 × 15	61,679	61,549	73,736	58,476	60,784	72,205	53,946	**53,913**
Bs6 20 × 15	47,322	46,992	47,645	48,128	45,028	51,541	43,786	**42,970**
Bs7 20 × 15	40,484	36,944	41,448	36,292	34,938	41,160	34,269	**34,011**
Bs8 20 × 15	62,649	61,156	62,272	60,803	61,800	63,624	59,565	**59,543**

The difference between DDQN and GNN + DRL lies only in the way of describing the job shop states. The former uses the state definition in the literature [10] when solving DJSP. The latter extracts the features of the analytic graph of the dynamic job shop by GNN. The other rule scheduling is several conventional job shop scheduling methods. The above are the optimal solutions obtained by different scheduling algorithms when solving different cases. It can be seen that the results of the general rule scheduling are more different because of the different scheduling criteria followed. This is still different from the optimal

solution derived by the deep reinforcement learning algorithm and shows the advantage of reinforcement learning algorithms in solving dynamic scheduling problems. However, the optimal solutions obtained by the above traditional solutions on different datasets are inferior to the optimal solutions obtained by the model constructed in this paper, which effectively proves the efficiency of graph neural networks for image feature extraction and the feasibility and effectiveness of the GNN plus DRL model constructed in this paper.

Figure 7 shows the experimental results of the deep reinforcement learning algorithm of GNN plus DDQN model on the arithmetic example Bs5, which is the average of the optimal solutions learned by the agents in 10 times of 400 rounds of training. The convergence curves of the images show that the model constructed in this paper has an advantage over the DDQN algorithm in terms of average convergence efficiency. It shows that the graph neural network has a significant advantage in feature selection compared with the traditional deep reinforcement learning algorithm. It also proves that the learning efficiency of the model is improved on the basis of the traditional DRL.

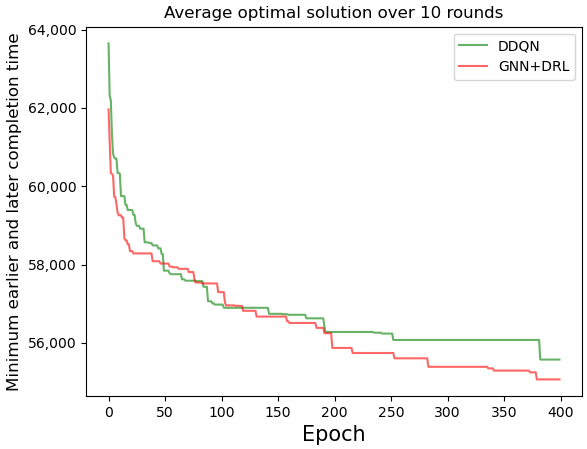

Figure 7. 10 rounds of learning the mean value of the optimal solution.

Figure 8 shows the graphs of earlier and later completion time duration versus iteration period for DDQN and GNN plus DDQN models on Bs2. The nodes indicate the average of the minimum earlier and later completion time for every 20 rounds of training, and it can be seen that, in general, the model in this paper outperforms the ordinary deep reinforcement learning model in terms of the optimization seeking effect. It proves that the GNN plus DDQN model has a better learning ability and better convergence in terms of the model's merit-seeking ability.

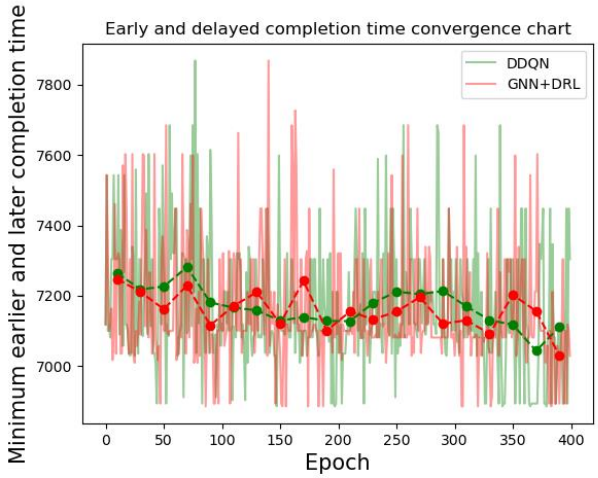

Figure 8. Convergence diagram of earlier and later completion time iterations.

5. Conclusions

In this paper, graph neural networks are integrated with deep reinforcement learning algorithms to construct a model framework that is different from traditional reinforcement learning algorithms. Then, by defining the DJSP environment, the scheduling problem is transformed into a Markov process solution, while the associated action space and new reward function are defined, which makes the model have considerable advantages in solving DJSP.

The effectiveness of the constructed model is demonstrated by comparing the results with those of the general rule-based scheduling method and the traditional deep reinforcement learning algorithm on different cases of the minimum earlier and later completion time. However, there is a lack of research on the complexity of the scheduling problem and its related perturbation factors, such as machine failure and job rescheduling. In the future, flexibility or other perturbation factors can be considered to be introduced into the model, and further research can be conducted on the multi-objective solution in order to obtain a better solution set and improve the solution quality.

Author Contributions: Methodology, writing, editing, and original draft preparation, Z.Y.; conceptualization, project administration, reviewing and funding acquisition, L.B. and X.J. All authors have read and agreed to the published version of the manuscript.

Funding: This study was supported by the National Natural Science Foundation of China (No. 62266034).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: After the paper is accepted, we will upload the experimental data and all the related codes. The following information was supplied regarding data availability: the dataset is available at GitHub. The code is available at GitHub.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Jackson, J.R. Simulation research on job shop production. *Nav. Res. Logist. Q.* **1957**, *4*, 287–295. [CrossRef]
2. Zhong, J.W.; Shi, Y.Q. Smart factory job shop scheduling based on DQN. *Mod. Manuf. Eng.* **2021**, *492*, 17–23+93. [CrossRef]
3. Zhang, C.; Song, W.; Cao, Z.; Zhang, J.; Tan, P.S.; Chi, X. Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning. *Adv. Neural Inf. Process. Syst. (NeurIPS)* **2020**, *33*, 1621–1632.
4. Zhang, K.; Bi, L.; Jiao, S.G. Research on flexible job shop scheduling problem with integrated reinforcement learning algorithm. *China Mech. Eng.* **2023**, *34*, 201–207.
5. Holthaus, O.; Rajendran, C. Effificient dispatching rules for scheduling in a job shop. *Int. J. Prod. Econ.* **1997**, *48*, 87–105. [CrossRef]
6. Qu, X.H.; Ji, F.; Meng, G.J.; Ding, B.R.; Wang, J. Research on green scheduling problem of flexible job shop with superheuristic genetic algorithm. *Mechatron. Eng.* **2022**, *39*, 255–261.
7. Zhang, S.; Wang, Y.; Ji, Z. A dynamic job shop scheduling method based on superheuristic genetic planning. *J. Syst. Simul.* **2020**, *32*, 2494–2506. [CrossRef]
8. Burggräf, P.; Wagner, J.; Saßmannshausen, T.; Ohrndorf, D.; Subramani, K. Multi-agent-based deep reinforcement learning for dynamic flexible job shop scheduling. *Procedia CIRP* **2022**, *112*, 57–62. [CrossRef]
9. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing Atari with Deep Reinforcement Learning. *arXiv* **2013**, arXiv:1312.5602.
10. Zhao, Y.; Wang, Y.; Tan, Y.; Zhang, J.; Yu, H. Dynamic jobshop scheduling algorithm based on deep q network. *IEEE Access* **2021**, *9*, 122995–123011. [CrossRef]
11. Luo, S. Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. *Appl. Soft Comput.* **2020**, *91*, 106208. [CrossRef]
12. Wang, X.; Zhang, L.; Ren, L.; Xie, K.; Wang, K.; Ye, F.; Chen, Z. A brief study on job shop scheduling problem based on reinforcement learning. *J. Syst. Simul.* **2021**, *33*, 2782–2791. [CrossRef]
13. Zhou, J.; Cui, G.; Hu, S.; Zhang, Z.; Yang, C.; Liu, Z.; Wang, L.; Li, C.; Sun, M. Graph neural networks: A review of methods and applications. *AI Open* **2020**, *1*, 57–81. [CrossRef]
14. Hameed, M.S.A.; Schwung, A. Reinforcement learning on job shop scheduling problems using graph networks. *arXiv* **2020**, arXiv:2009.03836.
15. Yu, B.; Xie, H.; Xu, Z. PN-GCN: Positive-negative graph convolution neural network in information system to classification. *Inf. Sci.* **2023**, *632*, 411–423. [CrossRef]

16. Zhao, K.; Du, C.; Tan, G. Enhancing Basketball Game Outcome Prediction through Fused Graph Convolutional Networks and Random Forest Algorithm. *Entropy* **2023**, *25*, 765. [CrossRef]
17. Zhong, H.; Wang, M.; Zhang, X. HeMGNN: Heterogeneous Network Embedding Based on a Mixed Graph Neural Network. *Electronics* **2023**, *12*, 2124. [CrossRef]
18. Zeng, Y.; Liao, Z.; Dai, Y.; Wang, R.; Li, X.; Yuan, B. Hybrid intelligence for dynamic job-shop scheduling with deep reinforcement learning and attention mechanism. *arXiv* **2022**, arXiv:2201.00548.
19. Park, J.; Chun, J.; Kim, S.H.; Kim, Y.; Park, J. Learning to schedule job-shop problems: Representation and policy learning using graph neural network and reinforcement learning. *Int. J. Prod. Res.* **2021**, *59*, 3360–3377. [CrossRef]
20. Chen, S.; Huang, Z.; Guo, H. An End-to-End Deep Learning Method for Dynamic Job Shop Scheduling Problem. *Machines* **2022**, *10*, 573. [CrossRef]
21. Wang, W.; Luo, S. A review of research on intelligent shop floor scheduling strategies based on reinforcement learning. *Comput. Appl. Res.* **2022**, *39*, 1608–1614. [CrossRef]
22. Zhang, Y.; Zhu, H.; Tang, D.; Zhou, T.; Gui, Y. Dynamic job shop scheduling based on deep reinforcement learning for multi-agent manufacturing systems. *Robot. Comput.-Integr. Manuf.* **2022**, *78*, 102412. [CrossRef]
23. Luo, Z.; Jiang, C.; Liu, L.; Zheng, X.; Ma, H. Research on intelligent job shop scheduling method based on deep reinforcement learning. *J. Internet Things* **2022**, *6*, 53–64.
24. Liu, C.L.; Chang, C.C.; Tseng, C.J. Actor-critic deep reinforcement learning for solving job shop scheduling problems. *IEEE Access* **2020**, *8*, 71752–71762. [CrossRef]
25. Wang, H.; Sarker, B.R.; Li, J.; Li, J. Adaptive scheduling for assembly job shop with uncertain assembly times based on dual Q-learning. *Int. J. Prod. Res.* **2021**, *59*, 5867–5883. [CrossRef]
26. Chang, J.; Yu, D.; Hu, Y.; He, W.; Yu, H. Deep reinforcement learning for dynamic flexible job shop scheduling with random job arrival. *Processes* **2022**, *10*, 760. [CrossRef]
27. Wang, Y.; Ding, Y. Dynamic flexible job shop optimal scheduling and decision making method. *J. Syst. Simul.* **2020**, *32*, 2073–2083. [CrossRef]