processes

MDPI

Article

Deep Reinforcement Learning for Dynamic Flexible Job Shop Scheduling with Random Job Arrival

Jingru Chang [1,2,3], Dong Yu [2,*], Yi Hu [2,4], Wuwei He [1,2] and Haoyu Yu [1,2]

[1] University of Chinese Academy of Sciences, Beijing 100049, China; changjingru@neusoft.edu.cn (J.C.); wuhewei2021@163.com (W.H.); yuhaoyu2021@sina.com (H.Y.)
[2] Shenyang Institute of Computing Technology, Chinese Academy of Sciences, Shenyang 110168, China; huyi@sict.ac.cn
[3] Department of Software Engineering, Dalian Neusoft University of Information, Dalian 116023, China
[4] Shenyang Zhongke CNC Technology Co., Ltd., Shenyang 110168, China
* Correspondence: yudong@sict.ac.cn

Abstract: The production process of a smart factory is complex and dynamic. As the core of manufacturing management, the research into the flexible job shop scheduling problem (FJSP) focuses on optimizing scheduling decisions in real time, according to the changes in the production environment. In this paper, deep reinforcement learning (DRL) is proposed to solve the dynamic FJSP (DFJSP) with random job arrival, with the goal of minimizing penalties for earliness and tardiness. A double deep Q-networks (DDQN) architecture is proposed and state features, actions and rewards are designed. A soft ε-greedy behavior policy is designed according to the scale of the problem. The experimental results show that the proposed DRL is better than other reinforcement learning (RL) algorithms, heuristics and metaheuristics in terms of solution quality and generalization. In addition, the soft ε-greedy strategy reasonably balances exploration and exploitation, thereby improving the learning efficiency of the scheduling agent. The DRL method is adaptive to the dynamic changes of the production environment in a flexible job shop, which contributes to the establishment of a flexible scheduling system with self-learning, real-time optimization and intelligent decision-making.

Keywords: smart factory; flexible job shop scheduling problem; deep reinforcement learning; random job arrival; penalties for earliness and tardiness; double deep Q-networks

Citation: 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. https://doi.org/10.3390/pr10040760

Academic Editors: Kelvin K.L. Wong, Dhanjoo N. Ghista, Andrew W.H. Ip and Wenjun (Chris) Zhang

Received: 16 March 2022
Accepted: 11 April 2022
Published: 13 April 2022

1. Introduction

Industry 4.0, also called the "smart factory" [1], focuses on the integration of advanced technologies such as the Internet of Things, big data and artificial intelligence with enterprise resource planning, manufacturing execution management and process control management. Thus, a smart factory has the capabilities of autonomous perception, analysis, reasoning, decision-making and control. The flexible job shop scheduling problem (FJSP) is an extension of the traditional job shop scheduling problem (JSP). The FJSP provides possibilities and guarantees low variation in diversified and differentiated manufacturing, which is widely used in the semiconductor manufacturing process, the automobile assembly process, mechanical manufacturing systems, etc. [2]. As the core of manufacturing execution management and process control management, the real-time optimization and control of FJSP provides increased flexibility in the management of a smart factory, aiming to improve factory productivity and the efficient utilization of resources in real time [3].

The FJSP breaks through the uniqueness restriction of production resources. Each operation can be assigned on one or more available machines and the processing time is different for different machines [4]. The FJSP reduces the machine constraints and expands the size of the feasible solution search space, so it is a strong NP-hard problem that is more complex than the JSP [5,6]. So far, a large number of studies on the FJSP have assumed that the scheduling takes place in a static production environment, where the shop floor

information is known in advance, and the deterministic scheduling scheme cannot be changed during the entire working process. However, an actual manufacturing shop has dynamic and uncertain characteristics, such as random job arrival, machine breakdowns, order cancellations, urgent order insertions, variations in delivery dates or processing times, etc. The scheduling scheme should be adjusted continuously according to the changes in the production environment [7,8], and the dynamic FJSP (DFJSP) can respond to the unexpected events of the flexible job shop in real time. Therefore, the research into the FJSP cannot meet the actual production demand and more and more scholars are now paying attention to the DFJSP.

At present, the methods of solving the DFJSP are mainly heuristic [9] and metaheuristic algorithms. Tao et al. [10] proposed an improved dual-chain quantum genetic algorithm, based on the non-dominated ranking method, to solve the multi-objective DFJSP. Nouiri et al. [11] used particle swarm optimization to solve the dynamic flexible job shop scheduling problem under machine breakdowns, to reduce energy consumption. Wu et al. [12] solved the DFJSP with multiple perturbations by the non-dominated sorting genetic algorithm (NSGA) III to minimize the maximum completion time and energy consumption. The heuristic algorithm is simple and efficient, but it often falls into the local optimum and the solution quality is poor due to greed and short-sightedness. The metaheuristic algorithm improves the solution quality through parallel searching and iterative searching, but it is time-consuming. Moreover, there is a strong correlation between algorithm structures and scheduling problems, which leads to the redesign of the algorithm once the production resources, constraints or production objectives change. Therefore, a method of solving the DFJSP urgently needs to be studied on the basis of new methods and new theories that integrate the advantages of the heuristic algorithm's solution time and the metaheuristic algorithm's solution quality.

With the advance of artificial intelligence, reinforcement learning (RL) to solve the production scheduling problem originated in 1995 [13]. In 2018, some scholars applied deep reinforcement learning (DRL) to the scheduling field and then it was widely used, which attracted the attention and competitive research of scholars in China and abroad. The basic components of reinforcement learning are the environment, agents, the behavior policy, the reward and the value function, where the learning process is usually described by a Markov decision process (MDP) [14]. For large-scale problems, it is necessary to parameterize it through a policy network and to balance exploration and exploitation, which ensures that the scheduling agent converges to the optimal or near-optimal solution in a reasonable time, thus improving the adaptability and self-learning of production scheduling in intelligent manufacturing.

Wang et al. [15] applied Q-learning to study a dynamic single-machine scheduling problem (SMSP) with random arrival time and processing time. Fonseca et al. [16] solved the flow job shop scheduling problem (FSP) to minimize the completion time of all jobs via the RL approach. Shahrabi et al. [17] solved the dynamic job shop scheduling problem with random job arrival and machine breakdowns by a variable neighborhood search that dynamically adjusted the parameters by RL to minimize the average flow time. Wang et al. [18] solved the job shop scheduling problem by a weighted Q-learning algorithm based on clustering and dynamic searching to minimize penalties for earliness and tardiness. Wang et al. [7] applied a dual Q-learning to solve an assembly job shop scheduling problem with uncertain assembly times to minimize the total weighted earliness penalty and completion time cost. The top level Q-learning is focused on the dispatching policy and the bottom level Q-learning focuses on global targets. Bouazza et al. [1] utilized the Q-learning algorithm to solve a partially flexible job shop scheduling problem with new job insertions. One Q matrix was used to choose a machine selection rule and the other was focused on a particular dispatching rule. Luo et al. [19] established double deep Q-networks (DDQN) with seven state features and six composite dispatching rules to solve the DFJSP, with the objective of minimizing total tardiness. Luo et al. [20] proposed a two-hierarchy deep reinforcement learning model for solving the FJSP to minimize the total tardiness and

average machine utilization rate. The higher-level DDQN determines the optimization goal and the lower-level chooses a proper dispatching rule. Table 1 summarizes the differences between the aforementioned work and our work.

Table 1. Existing RL methods for dynamic scheduling problem.

Work	Problem	Dynamic Events	Objective	Algorithm	State	Policy
Wang et al. [15]	SMSP	Random job arrival, Random processing time	Makespan, Summed tardiness, Mean flow time	Q-learning	Discrete	ε-greedy
Fonseca et al. [16]	FSP	Sequence dependent setup times	Makespan	Q-learning	Discrete	ε-greedy
Shahrabi et al. [17]	JSP	Random job arrival	Mean flow time	Q-learning	Discrete	ε-greedy
Wang et al. [18]	JSP	Random job arrival	Penalties for earliness and tardiness	Q-learning	Discrete	ε-greedy
Wang et al. [7]	JSP	Uncertain assembly times	Total earliness penalty, Completion time cost	Dual Q-learning	Discrete	ε-greedy
Bouazza et al. [1]	FJSP	Random job arrival	Makespan, Total weighted completion time	Q-learning	Discrete	Annealed linearly ε-greedy
Luo et al. [19]	FJSP	Random job arrival	Total tardiness	DDQN	Continuous	Soft-max
Luo et al. [20]	FJSP	Random job arrival	Total tardiness, Machine utilization rate	DDQN	Continuous	Annealed linearly ε-greedy
Our work	FJSP	Random job arrival	Penalties for earliness and tardiness	DDQN	Continuous	Soft ε-greedy

From the above literature review, the research has mainly focused on single machine scheduling, flow job shop scheduling and job shop scheduling. Research on DRL for solving the DFJSP has not been explored deeply. Moreover, the DFJSP, with random job arrival and penalties for earliness and tardiness criteria, has not been solved by DRL. In addition, the DRL methods are not compared with traditional metaheuristics in most of the literature. It is unclear whether DRL approaches outperform traditional metaheuristics in terms of solution quality and generalization. For instance, in the work of Luo et al. [19], the proposed DRL demonstrated superiority only when compared with heuristic rules as well as the Q-learning agent.

In most of the RL-based methods mentioned above, Q-learning is mostly used, which requires the problem to have discrete and finite state space. To maintain a lookup Q table and reduce computational complexity, model accuracy is often sacrificed when dealing with continuous-state problems. For instance, in the work of Shahrabi et al. [17], the number of machines/jobs/operations chosen as state features is unlimited and extremely large. There is no efficient theoretical guidance on how to determine the proper number of states, so the drawback of compulsive state discretization is obvious. In the work of Luo et al. [19,20], the DDQN-based scheduling agent is designed, whereas there are strong correlations between hand-crafted features, which may mislead the neural networks and increases many invalid computations. Without loss of generality, ε-greedy or annealed linearly ε-greedy are used for most of the literature above. With the rapid growth of the scheduling solution space, the fixed ε and fixed linear annealing rates are not conducive to searching for the optimal or near-optimal solution.

For the reasons mentioned above, a DRL method is proposed to solve the DFJSP with random job arrival, to minimize penalties for earliness and tardiness in this study, so as to realize the real-time optimization and decision-making of the DFJSP. The experimental results indicate that the proposed DRL outperforms other reinforcement learning algorithms, heuristics and metaheuristics in terms of solution quality and generalization. The three contributions of this research are as follows.

(1) To the best of our knowledge, this is the first attempt to solve the DFJSP with random job arrival, to minimize the total penalties for earliness and tardiness using DRL. The work can thus fill a research gap regarding solving the DFJSP by DRL.
(2) A DDQN algorithm model of flexible dynamic scheduling is proposed and state features, actions and rewards for the scheduling agent have been designed.
(3) A soft ε-greedy behavior policy is proposed, which reasonably balances exploration and exploitation according to the solution space of strong NP-hard problems, thus improving the learning speed of the scheduling agent.

The remainder of this study is organized as follows: The mathematical model of DFJSP with random job arrival is established in Section 2. Section 3 presents the background of DDQN and gives the implementation details. Section 4 provides the results of numerical experiments. Section 5 discusses the findings and the implications and gives future research directions. Finally, conclusions are drawn in Section 6.

2. Problem Formulation

2.1. Problem Description

We describe the dynamic flexible job shop scheduling problem with random job arrival using symbols defined as follows: There are n successively arriving jobs $J = \{J_1, J_2, \ldots, J_n\}$, which should be processed on m machines $M = \{\mathrm{M}_1, M_2, \ldots, M_m\}$. Each job J_i consists of a predetermined sequence of h_i operations. O_{ij} is the jth operation of job J_i, which can be processed on a compatible machine set. The processing time of O_{ij} on machine $\mathrm{M_k}$ is denoted t_{ijk}. The arrival time of job J_i is A_i and the due date is D_i. In this study, the assumptions and constraints were as follows:

(1) Each machine can process only one operation at a time.
(2) The order of precedence of operations belonging to the same job must be followed and there are no precedence constraints among the operations of different jobs.
(3) The operation must be processed without interruption.
(4) Jobs are independent and no priorities are assigned to any job.
(5) The setup time of the equipment, the transportation time between operations and the breakdown time of the machine are negligible.
(6) An unlimited buffer between machines is assumed.

2.2. Mathematical Model

In order to meet the needs of the just-in-time production mode, the basic requirement of the scheduling problem to minimize penalties for earliness and tardiness [21] is as follows: From the perspective of the economic benefit of an enterprise, the processing of products should meet the requirements of delivery time, with neither delays nor a principle of "the sooner the better". A mathematical model of the DFJSP was established to minimize penalties for earliness and tardiness with random job arrival. The notation used in this model is as follows:

i,r: Index of jobs, $i = 1, 2, 3 \ldots n$;
j,t: Index of operations belonging to job J_i and J_t;
k: Index of machines, $k = 1, 2, 3 \ldots m$;
h_i: The number of operations of J_i;
t_{ijk}: The processing time of operation O_{ij} on machine M_k;
s_{ij}: The starting time of O_{ij};
m_{ij}: The available machine set for operation O_{ij};
f_i: The delivery relaxation factor of J_i;
w_i^e: Unit (per day) earliness cost of J_i;
w_i^t: Unit (per day) tardiness cost of J_i;
A_i: The arrival time of J_i;
D_i: The due date of J_i;
C_i: The completion time of J_i;
Z: A large enough positive number.

In an actual production environment, the swift completion of products results in more inventory pressure and financial costs, whereas delays in completing the job result in financial damage [22]. Therefore, here, the objective was to obtain a schedule that has the least penalties for earliness and tardiness (PET) in the DFJSP with new job insertions. The objective function is given by Equation (1) and some constraints are given in Equations (2)–(8).

Objective:

$$\mathrm{PET} = \min\left\{\sum_{i=1}^{N}\left(w_i^e \times \max(D_i - C_i, 0) + w_i^t \times \max(C_i - D_i, 0)\right)\right\} \quad (1)$$

Subject to:

$$s_{ij} \geq 0\,, s_{i1} - A_i \times x_{i1k} >= 0 \;\; i = 1,\ 2,\ 3 \ldots n; j = 1,2,3 \ldots h_i; k = 1,\ 2,\ 3 \ldots m; \quad (2)$$

$$s_{ij} + t_{ij} \leq s_{i(j+1)} \;\; i = 1,2,3 \ldots n; j = 1,2,3 \ldots h_i; \quad (3)$$

$$s_{ij} + t_{ijk} \leq s_{rt} + Z \times \left(1 - y_{ijrtk}\right) \;\; i = 1,\ 2,\ 3 \ldots n; r = 1,\ 2,\ 3 \ldots n; j = 1,\ 2,\ 3 \ldots h_i; t = 1,2,3 \ldots h_r; k = 1,\ 2,\ 3 \ldots m; \quad (4)$$

$$\sum_{k=1}^{m_{ij}} x_{ijk} = 1 \; i = 1,\ 2,\ 3 \ldots n; j = 1,\ 2,\ 3 \ldots h_i; \quad (5)$$

$$x_{ijk} = \begin{cases} 1, \; If\ O_{ij}\ is\ assigned\ to\ \ M_k \\ 0, \; else \end{cases} \quad (6)$$

$$y_{ijrtk} = \begin{cases} 1, & If\ O_{ij}\ is\ processed\ on\ M_k\ before\ O_{rt} \\ 0, & else \end{cases} \quad (7)$$

$$D_i = A_i + f_i \times \sum_{j=1}^{h_i} t_{ij} \quad (8)$$

Equation (2) makes sure that a job can only be processed after its arrival time. Equation (3) indicates that the order of precedence between the operations of each job must be followed. Equation (4) ensures that a machine can only process one job at a time. Equation (5) ensures that a job can only be processed by one machine at the same time.

3. Proposed DRL

3.1. DQN and DDQN

Deep Q-networks (DQN) combine reinforcement learning with non-linear value functions for the first time, in which the neural network is trained through reinforcement learning to have the ability to master difficult control and decision-making policies [23]. However, there is an incompatible gap between reinforcement learning and deep learning. For example, most deep learning methods assume that the data samples are independent of each other, with no sequence correlation and a fixed underlying distribution, while in reinforcement learning, sequences of highly correlated states are typically encountered and the data distribution is unstable under the influence of selective actions. Inspired by the experience replay of the hippocampus [24] in a biological neural network, the state transition tuple (s_t, a_t, r_t, s_{t+1}) generated at each time-step in reinforcement learning is stored in a replay memory and the tuple data are randomly sampled to adjust the parameter θ_t of the neural network (the iterative updating formula is shown in Equation (9)) by minibatch updates. Therefore, the goal of maximizing the Q-value function of RL is realized and the loss function of deep learning is minimized at the same time. The DQN is a milestone in creating a general artificial intelligence to complete a varied range of challenging tasks with a single algorithm.

$$\theta_{t+1} = \theta_t + \eta \times (y_t - Q(s_t, a_t; \theta_t)) * \nabla_{\theta_t} Q(s_t, a_t; \theta_t) \quad (9)$$

where η is the learning rate used by the stochastic gradient descent algorithm. The target y_t must be designed in unsupervised learning, for which the iterative formula of each time-step is shown in Equation (10), while the sample target y_t is known in supervised learning; γ is the discount factor in the Q-learning algorithm.

$$y_t^{DQN} = r_t + \gamma \times argmax_{a'} Q(s_{t+1}, a'; \theta_t) \tag{10}$$

It can be seen from Equation (10) that selecting an action and evaluating an action use the same values in the DQN. There is a neural network where the current parameter is θ_t, the state s_{t+1} is the input and the number of Q values at the output layer is $|A|$, so the set expression is $\{Q(s_{t+1}, a_1; \theta_t), Q(s_{t+1}, a_2; \theta_t) \ldots . Q\left(s_{t+1}, a_{|A|}; \theta_t\right)$, where $|A|$ is the number of actions$\}$. If the Q value of a' is the largest, then action a' is chosen via an $\varepsilon - greedy$ behavior policy, while the evaluation uses the value $Q(s_{t+1}, a'; \theta_t)$. This results in a large number of overoptimistic value estimates.

The overoptimistic value estimates themselves are not necessarily a problem, but they have a negative impact on the quality of the learned policy in some cases. If the Q value of each action is overestimated evenly, the action selection will not be affected and the agent learning will not be affected. On the contrary, if the overestimation of Q is uneven, the action with the overestimated value will be preferred during action selection. The action affects the environmental state distribution in RL, so the overestimation of Q affects the state data's distribution. If the agent learns less from those states, the learning quality of the agent will be greatly reduced and overoptimism is not conducive to the stability of learning.

In order to reduce overestimations, Hasselt et al. designed the DDQN [25] from the idea of double Q-learning [26,27]. The online network and the target network are designed to decouple the selection from the evaluation. The Q value from the online network provides the basis on which the behavior policy can select an action and the target network $\hat{Q}$ value is used for evaluating the action. The iterative formula of y_t is shown in Equation (11). The update of the target network $\hat{\theta}_t$ remains a periodic copy of the online network θ_t.

$$y_t^{DDQN} = r_t + \gamma \times \hat{Q}\left(s_{t+1}, argmax_{a'} Q\left(s_{t+1}, a'; \theta_t\right); \hat{\theta}_t\right) \tag{11}$$

3.2. Model Architecture

The general process of reinforcement learning to solve the production scheduling problem is as follows: Firstly, the scheduling problem type, constraint conditions and dynamic attributes are defined according to the manufacturing environment, which generates the production scheduling instance. Secondly, the instance is expressed as a MDP according to the production state, scheduling action and reward. Lastly, the agent then continuously interacts with the MDP to obtain production data samples and the reinforcement learning algorithm is trained to learn the strategy.

The model of the proposed DRL is shown in Figure 1, including the flexible job shop production environment, the agent and the reinforcement learning process. The architecture of the online network and the target network in the agent is the same. Deep neural networks are trained in the DDQN, which consists of five fully connected layers with one input layer, one output layer and three hidden layers. The number of nodes in the input and output layers is equal to the number of state features (four) and the number of actions is also four. Each hidden layer consists of 30 nodes. The activation function is Relu. The learning process is as follows.

(1) The agent obtains the current state of the flexible job shop environment s_t;
(2) The agent determines the scheduling rule a_t according to the Q value of the online network and the behavior policy to select an operation O_{ij} and select a feasible machine M_k;
(3) The flexible production shop performs a_t: O_{ij} is processed on M_k and the production environment state is transferred to s_{t+1};

(4) The agent obtains the instant reward r_t from the production environment and the experience tuple (s_t, a_t, r_t, s_{t+1}) is stored in the experience replay D;
(5) Randomization of the samples is performed in D and the target and the loss function are calculated according to the target network and the online network to update the parameters θ_t of the online network.

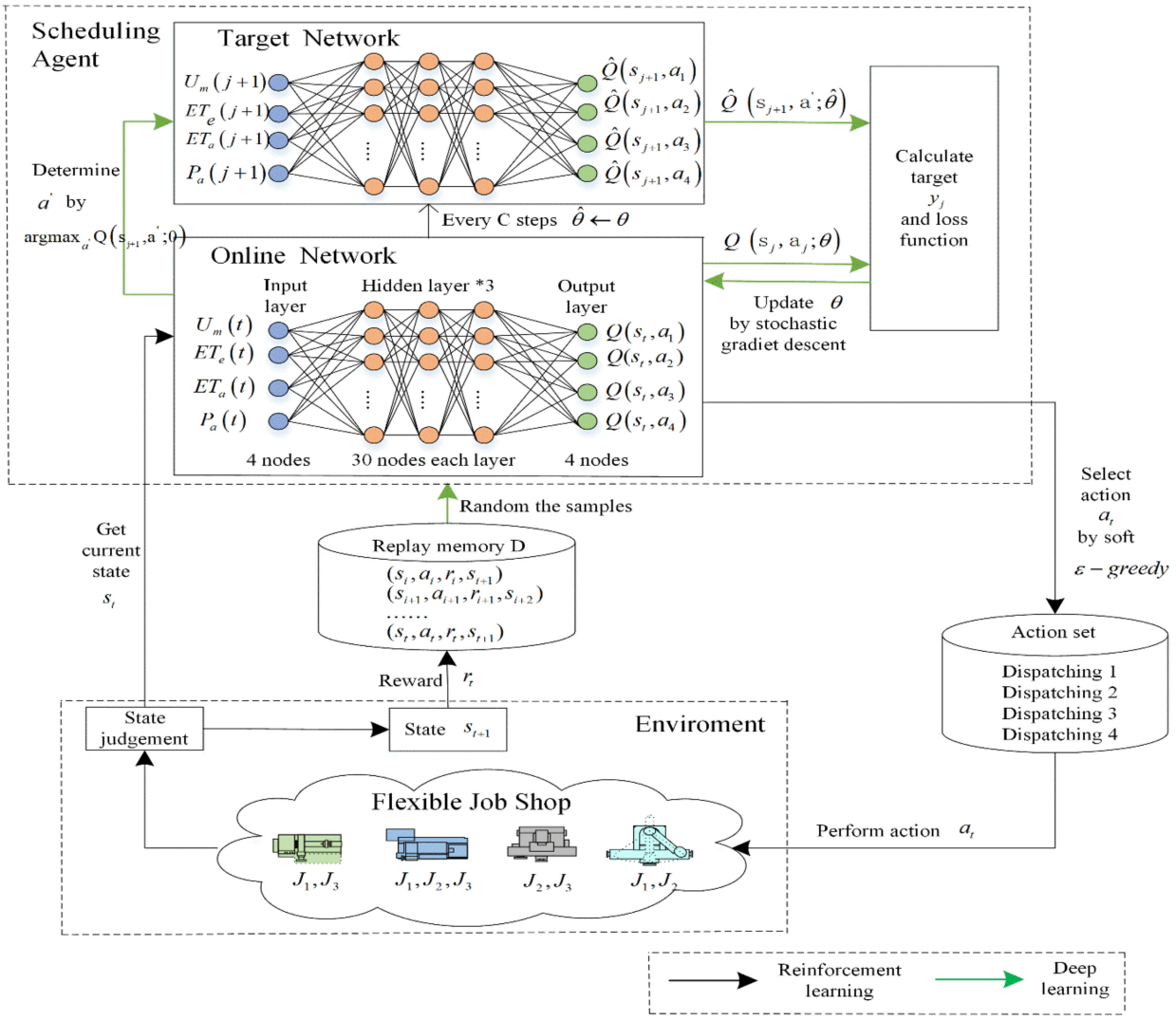

Figure 1. The model architecture of solving the DFJSP using the DDQN.

3.3. State Features

In the field of reinforcement learning applications, the design quality of the environmental state features plays a key role and influences the performance of an RL algorithm. In a production scheduling method based on RL, the characteristics of the scheduling attributes are defined as the production state characteristics, such as the number of jobs, the number of operations, the number of machines, the remaining working hours, the number of remaining operations, the load of machine tools, the total processing time and other factors [1,17]. These characteristic attributes have an infinite range of values and the decomposition and partition of the state space can easily be subjective and lack the guidance of objective data [28]. In a flexible manufacturing environment, production information is complex and constantly changing and excessive quantitative eigenvalues are prone to overfitting [29]. In order to ensure that different actions are selected adaptively according to the state of the production environment, most reinforcement learning algorithms solve the production scheduling problem by establishing the relationship between the production

features' attributes and the production objectives. For these reasons, this study designed four production state features with values of [0, 1], which are defined below:

(1) Average utilization rate $\overline{U}_m(t)$

The average utilization rate of the machines $\overline{U}_m(t)$ is calculated by Equation (12). $CT_k(t)$ is the completion time of the last operation on machine M_k at rescheduling point t and $OP_i(t)$ is the current number of completed operations of job Ji at the current time t.

$$\overline{U}_m(t) = \frac{\sum_{k=1}^{m}\left(\frac{\sum_{i=1}^{n}\sum_{j=1}^{OP_i(t)} t_{ijk}\times X_{ijk}}{CT_k(t)}\right)}{m} \tag{12}$$

(2) Estimated earliness and tardiness rate $ET_e(t)$

T_{cur} is the average completion time of the last operations on all machines at rescheduling point t and T_{left} is the estimated remaining processing time of J_i. If $T_{cur} + T_{left} > D_i$, J_i is estimated to be delayed. If $T_{cur} + T_{left} < D_i$, J_i is estimated to be completed in advance. The number of estimated early and tardy jobs is equal to the number of estimated early jobs NJ_{early} plus the number of estimated tardy jobs NJ_{tard}. The estimated earliness and tardiness rate $ET_e(t)$ is equal to the number of estimated early and tardy jobs divided by the number of all jobs. The method of calculating this is given in Algorithm 1.

Algorithm 1 Procedure of calculating the estimated earliness and tardiness rate $ET_e(t)$

Input: $CT_k(t)$, $OP_i(t)$, D_i
Output: $ET_e(t)$
$T_{cur} \leftarrow \frac{\sum_{k=1}^{m} CT_k(t)}{m}$
$NJ_{tard} \leftarrow 0$
$NJ_{early} \leftarrow 0$
for $i = 1$: n **do**
  if $OP_i(t) < h_i$ **then**
    $T_{left} \leftarrow 0$
    for $j = OP_i(t) + 1$: h_i **do**
      $t_{ij} = mean_{k \in m_{ij}} t_{ijk}$
      $T_{left} \leftarrow T_{left} + t_{i,j}$
      if $T_{cur} + T_{left} > D_i$ **then**
        $NJ_{tard} \leftarrow NJ_{tard} + 1$
        break
    end if
  end for
  if $T_{cur} + T_{left} < D_i$ **then**
    $NJ_{early} \leftarrow NJ_{early} + 1$
  end if
end if
end for
$ET_e(t) \leftarrow (NJ_{tard} + NJ_{early})/n$
Return $ET_e(t)$

(3) Actual earliness and tardiness rate $ET_a(t)$

$ET_i(t)$ is the completion time of the completed operations of J_i at rescheduling point t and thus $ET_i(t)[OP_i(t)]$ represents the completion time of the last completed operation of J_i. If $ET_i(t)[OP_i(t)] > D_i$, J_i is delayed; if $ET_i(t)[OP_i(t)] + T_{left} < D_i$, J_i is completed in advance. The actual number of early and tardy jobs is equal to the actual number of early jobs NJ_{a_early} plus the actual number of tardy jobs NJ_{a_tard}. The actual earliness and tardiness rate $ET_e(t)$ is equal to the number of actual early and tardy jobs divided by the number of all jobs. The method of calculating this is given in Algorithm 2.

Algorithm 2 Procedure of calculating the actual earliness and tardiness rate $ET_a(t)$

Input: $OP_i(t)$, D_i, $ET_i(t)$
Output: $ET_a(t)$
$NJ_{a\_tard} \leftarrow 0$
$NJ_{a\_early} \leftarrow 0$
for $i = 1$: n **do**
  if $OP_i(t) < h_i$ **then**
    $T_{left} \leftarrow 0$
    if $ET_i(t)[OP_i(t)] > D_i$ **then**
      $NJ_{a\_tard} \leftarrow NJ_{a\_tard} + 1$
      continue
    else
      for $j = OP_i(t) + 1$: h_i **do**
        $t_{ij} = mean_{k \in m_{ij}} t_{ijk}$
        $T_{left} \leftarrow T_{left} + t_{i,j}$
        if $ET_i(t)[OP_i(t)] + T_{left} > D_i$ then
          $NJ_{a\_tard} \leftarrow NJ_{a\_tard} + 1$
          break
        end if
      end for
      if $ETi(t)[OP_i(t)] + T_{left} < D_i$ **then**
        $NJ_{a\_early} \leftarrow NJ_{a\_early} + 1$
      end if
    end if
  end if
end for
$ET_a(t) \leftarrow (NJ_{a\_tard} + NJ_{a\_early})/n$
Return $ET_a(t)$

(4) Actual earliness and tardiness penalty $P_a(t)$

P [i] is the actual earliness and tardiness penalty of $J_{i,}$ and its value is equal to the unit time penalty coefficient of J_i multiplied by the actual earliness/tardiness of J_i. The actual earliness and tardiness penalty *Pa(t)* is normalized by [0,1). Its normalization equation is $P_a(t) = \frac{\sum_{i=1}^{n} P[i]}{\sum_{i=1}^{n} P'[i]}$, where $\sum_{i=1}^{n} P'[i] = \sum_{i=1}^{n} P[i] + Z$ and Z is a constant related to n, $Z \in [n, n*10]$. If there are no early and tardy jobs at the rescheduling point t, $\sum_{i=1}^{n} P[i]$ is equal to 0 and the value of $P_a(t)$ is also 0; otherwise, a large number of jobs are early or tardy, for which the $\sum_{i=1}^{n} P[i]$ is greater and $P_a(t)$ is closer to 1. The specific calculation method is shown in Algorithm 3.

Algorithm 3 Procedure of calculating the actual earliness and tardiness penalty cost $P_a(t)$

Input: $OP_i(t)$, D_i, $ET_i(t)$
Output: $P_a(t)$
$P' \leftarrow 1$
$P \leftarrow 0$
for $i = 1$: n **do**
  if $OP_i(t) < h_i$ **then**
    $T_{left} \leftarrow 0$
    for $j = OPi(t) + 1$: hi **do**
      $t_{ij} = mean_{k \in m_{ij}} t_{ijk}$
      $T_{left} \leftarrow T_{left} + t_{i,j}$
    end for
    if $ET_i(t)[OP_i(t)] > D_i$ **then**
      $P[i] \leftarrow w_i^t * \left(ET_i(\mathrm{t})[OP_i(\mathrm{t})] + T_{left} - Di\right)$
      $P\prime[i] \leftarrow w_i^t * \left(ET_i(\mathrm{t})[OP_i(\mathrm{t}) + T_{left} - Di\right) + 10$
    end if
    if $ET_i(t)[OP_i(t)] + T_{left} < D_i$ **then**
      $P[i] \leftarrow w_i^e * \left(Di - ET_i(\mathrm{t})[OP_i(\mathrm{t})] - T_{left}\right)$
      $P\prime[i] \leftarrow w_i^e * \left(Di - ET_i(\mathrm{t})[OP_i(\mathrm{t})] - T_{left}\right) + 10$
    end if
  end if
end for
$P_a(t) = \frac{\sum_{i=1}^{n} P[i]}{\sum_{i=1}^{n} P\prime[i]}$
Return $P_a(t)$

3.4. Action Set

The FJSP problem includes two subproblems: operation sequencing and machine selection. Therefore, the four scheduling rules were designed to complete two tasks: first selecting an operation and then selecting a machine from the set of feasible machines. $UC_{job}(t)$ is the set of unfinished jobs at rescheduling point t and M_{ij} represents the set of suitable machines for O_{ij}. The four comprehensive dispatching rules are as follows.

(1) Dispatching Rule 1: Firstly, according to Equation (13), the job J_i with the minimum redundancy time is selected from the uncompleted job set $UC_{job}(t)$ and the operation $O_{i(OPi(t)+1)}$ is selected. The machine is then allocated for $O_{i(OPi(t)+1)}$ and the minimum completion time is the allocation principle. Selection of the machine considers not only the available time of the machine but also the completion time of the prior $O_{i\,OPi(t)}$ and the processing time of $O_{i(OPi(t)+1)}$. When J_i arrives dynamically at rescheduling time t, if a feasible machine is idle, its available time is the rescheduling time t; otherwise, its available time is the time when it completes the processing operation. Therefore, the machine is selected according to Equation (14).

$$\min_{i \in UC_{job}(t)} \{D_i - \frac{\sum_{k=1}^{m} CT_k(t)}{m}\} \tag{13}$$

$$min\{\max_{k \in M_{i(OP_i(t)+1)}} \{CT_k(t), C_{iOP_i(t)}, A_i\} + t_{i(OP_i(t)+1)k}\} \tag{14}$$

(2) Dispatching Rule 2: Firstly, 2according to Equation (15), the job J_i with the largest estimated remaining processing time is selected from the uncompleted jobs and its operation $O_{i(OPi(t)+1)}$ is selected. A suitable machine for $O_{i(OPi(t)+1)}$ then is selected according to Equation (14).

$$\max_{i \epsilon UC_{job}(t)} \{\sum_{j=OP_i t+1}^{n_i} mean_{k \epsilon M_{ij}} t_{ijk}\} \tag{15}$$

(3) Dispatching Rule 3: Firstly, according to Equation (16), the job J_i with the largest penalty coefficient is selected from the uncompleted jobs and its operation $O_{i(OPi(t)+1)}$ is selected. The suitable machine with the smallest load for $O_{i(OPi(t)+1)}$ is selected according to Equation (17).

$$\max_{i \epsilon UC_{job}(t)} \left\{0.2 \times w_i^e + 0.8 \times w_i^t\right\} \tag{16}$$

$$\min_{k \epsilon M_{i,OP_i(t)+1}} \{\sum_{i=1}^{n} \sum_{j=1}^{OP_i(t)} t_{ijk} x_{ijk}\} \tag{17}$$

(4) Dispatching Rule 4: Firstly, according to Equation (18), the job J_i with the smallest estimated remaining processing time is selected from the uncompleted jobs and its process $O_{i(OPi(t)+1)}$ is selected. A suitable machine for $O_{i(OPi(t)+1)}$ is then selected according to Equation (14).

$$\min_{i \epsilon UC_{job}(t)} \{\sum_{j=OP_i t+1}^{n_i} mean_{k \epsilon M_{ij}} t_{ijk}\} \tag{18}$$

3.5. Rewards

In this study, the goal of production scheduling was to minimize penalties for earliness and tardiness, while the goal of the DDQN algorithm was to maximize the cumulative reward. Therefore, the reward function keeps the increasing direction of the cumulative reward consistent with the decreasing direction of the optimization goal. In order to

improve the learning efficiency of agents, this study designed a heuristic immediate reward function, which is calculated by Equation (19).

$$r_t = P_a(t) - P_a(t+1) \tag{19}$$

If $P_a(t+1) < P_a(t)$, this indicates that the scheduling optimization objective is decreasing, and if the immediate reward is $r_t > 0$ according to Equation (19), this indicates that the cumulative reward is increasing. Moreover, the more the optimization objective is reduced, the greater the immediate reward in the iteration. If $P_a(t+1) = P_a(t)$, the scheduling optimization objective changes to 0 and the immediate reward is also 0. If $P_a(t+1) > P_a(t)$, the optimization goal is increasing and if the immediate reward is $r_t < 0$, the cumulative reward is decreasing. There is a negative correlation between the optimization goal and the cumulative reward. Therefore, through the definition of the immediate reward function, not only is the minimization objective of the scheduling problem transformed into the maximization objective of the cumulative reward, but also the selected action a_t of each decision point t is accurately evaluated, which improves the learning ability of the agent regarding a complex control strategy.

3.6. Action Selection Strategy

In deep reinforcement learning, exploration means that every action has the same probability of being randomly selected and exploitation involves selecting the action with the largest Q value. Due to the limited learning time, exploration and exploitation are contradictory. In order to maximize the cumulative reward, a compromise must be made between exploration and exploitation.

The ε-greedy policy, with ε being annealed linearly, is one of the most commonly used behavior policies. For example, ε with an initial value of 1, anneals linearly by 0.001 at each step and is fixed at 0.1. Thereafter, the probability of exploration is 0.1 and that of exploitation is 0.9 at each step. However, a fixed linear annealing rate is not reasonable for all flexible scheduling problems. In order to improve the learning speed of the agent, there is less exploration for scheduling problems with a small solution space, while exploration should be strengthened for scheduling problems with a large solution space. Therefore, in this study a soft ε-greedy behavior policy, which is calculated by Equation (20), was designed to adapt to flexible scheduling problems with different scales, and the linear annealing rate is $\frac{1}{(OP\_num)^{\mu}}$. The larger the total operation number OP_num and the larger the solution space of the scheduling problem, the smaller the value of $\frac{1}{(OP\_num)^{\mu}}$, which means that the linear decline of the exploration rate ε is slower, thus enhancing exploration and weakening exploitation.

$$\varepsilon_{soft} = max\{0.1\ ,\ 1 - \frac{step}{(OP\_num)^{\mu}}\}\ (\mu > 0) \tag{20}$$

3.7. Procedure of DDQN

By defining three key elements (state, action and reward), the DFJSP problem is transformed into an RL problem. According to the algorithm model architecture described in Section 3.2, the four production environment state characteristics in Section 3.3, the four action scheduling rules in Section 3.4, the immediate reward in Section 3.5 and the behavior policy in Section 3.6, the scheduling agent is trained to realize adaptive scheduling. Algorithm 4 is the training method of the scheduling agent, where L is the training time, t is the rescheduling time when an operation is completed or a new job arrives and T is the sum of all the current operations.

Algorithm 4 The DDQN-based training method

1: Initialize replay memory D to capacity N
2: Initialize online network action-value Q with random weights θ
3: Initialize target network action-value $\hat{Q}$ with weights $\hat{\theta} = \theta$
4: **for** episode = 1: L **do**
5: Initialize production state $s_1 = \{\overline{U}_m(1), ET_e(1), ET_a(1), P_a(1)\} = \{0, 0, 0, 0\}$
6: **for** t = 1: T **do**
7: With probability ε_{soft}, select a random action a_t
8: Otherwise, select action $a_t = argmax_a Q(s_t, a; \theta)$
9: Execute action a_t, calculate the immediate reward r_t by Equation (19) and observe the next state s_{t+1}
10: Set production state $s_{t+1} = \{\overline{U}_m(t+1), ET_e(t+1), ET_a(t+1), P_a(t+1)\}$
11: Store transition (s_t, a_t, r_t, s_{t+1}) in D
12: Sample a random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from D
13: Set target$y_j = \begin{cases} r_j & if\ episode\ terminates\ at\ step\ j+1 \\ r_j + \gamma \hat{Q}(s_{t+1}, argmax_{a'} Q(s_{t+1}, a'; \theta); \hat{\theta}) & otherwise \end{cases}$
14: Calculate the loss function $\left(y_j - Q(s_j, a_j; \theta)\right)^2$ and perform a stochastic gradient descent step with respect to the parameters θ of online network Q
15: Every C steps, reset $\hat{\theta} = \theta$
16: **end for**
17: **end for**

4. Numerical Experiments

In this section, a correlation analysis between the state features and the process of training the scheduling agent are provided, followed by a sensitivity study on the control parameter μ of the soft ε-greedy action selection policy. To confirm reasonable exploration of the soft ε-greedy strategy, the learning rate between the flexible ε-greedy strategy and the fixed linearly decreasing ε-greedy strategy was compared. To show the superiority and generality of the DDQN, we compared it with DQN; SARSA; a well-known heuristic algorithm, first in first out (FIFO); a traditional metaheuristic algorithm, genetic algorithm (GA); and a random action strategy (RA) with different production configurations. The training and test results, and the video of solving the DFJSP using the trained DDQN are uploaded as Supplementary Materials.

The problem instances were generated by simulating a dynamic production environment of a flexible job shop. A new job arrival or an operation completion is defined as the system event that triggers rescheduling. It is assumed that several jobs exist on the flexible shop floor at the very beginning. The arrival of subsequent new jobs follows a Poisson distribution, whereas the arrival interval obeys a negative exponential distribution with an average rate E_{ave}. For J_i, the delivery relaxation factor f_i, the operation number h_i, the process time t_{ij} of the jth operation O_{ij}, and ${w_i}^e$ and ${w_i}^t$ are satisfied with a uniform distribution [29]. The parameter settings are shown in Table 2.

Table 2. Parameter settings of different production configurations.

Parameter	Value
Number of machines (m)	{5, 10, 30}
Number of initial jobs (n_{ini})	10
Number of newly added jobs (n_{add})	{20, 30, 50, 100}
Delivery relaxation factor (f_i)	U [0.5, 2]
Average value of exponential distribution between two successive job arrivals (E_{ave})	{30, 50, 100}
Number of operations in a job (h_i)	U [1, 30]
Processing time of an operation on a machine (t_{ij})	U [0, 100]
Unit (per day) of earliness cost(${w_i}^e$)	U [1, 1.5]
Unit (per day) of tardiness cost(${w_i}^t$)	U [1, 2]

The algorithm proposed in this study and the flexible workshop production environments were coded with Python 3.8.3. The training and test experiments were performed on a PC with an Intel(R) Core(TM) i7-6700 CPU and a 3.40 GHz CPU and 16 GB RAM.

4.1. Training Details

The values of all the hyperparameters were selected by performing an informal search on the instances that were generated for the DFJSP, with random job arrival using different parameter settings of E_{ave}, n_{add} and m. In line with the literature [24], a systematic grid search was not performed owing to the high computational cost, although it is conceivable that even better results could be obtained by systematically tuning the hyperparameter values. The list of hyperparameters and their values are shown in Table 3.

Table 3. List of hyperparameters and their values.

Hyperparameter	Value
Replay memory size (N)	2000
Minibatch size	32
Behavior policy (ε_{soft})	Decreasing linearly from 1 to 0.1
Discount factor (γ)	0.95
Learning rate (η)	0.00025
Update step of the target network (C)	100
Replay start size	100

4.1.1. Correlations between States

In order to collect a set of states completely and fully, multiple problem instances were generated according to different production configurations. In order to avoid sequences of highly correlated states, a set of states was collected from different instances by running a random policy before the training started [23], so as to objectively evaluate the correlation between the state features. The correlation coefficient $\rho_{X,Y}$ between the state features is calculated according to the following equation: $\rho_{X,Y} = \frac{\sum_{i=1}^{n}(X_i-\overline{X})(Y_i-\overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i-\overline{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i-\overline{Y})^2}}(X:\ x_1, x_2, \ldots, x_n;\ Y:\ y_1, y_2, \ldots, y_n)$. The experimental results are shown in Table 4 below. ET_e is moderately correlated with ET_a, $\overline{U}$ is less correlated with ET_e and $\overline{U}$ is less correlated with ET_a as well, while the other state characteristics have extremely low correlations.

Table 4. Correlation coefficients between state features.

	$\overline{U}_m$	ET_e	ET_a	P_a
$\overline{U}_m$	1	−0.27118903	0.20070019	−0.03637265
ET_e	-	1	−0.54936787	0.06982738
ET_a	-	-	1	0.1090457
P_a	-	-	-	1

4.1.2. Training and Stability

The DDQN was trained for a simulated flexible job shop with 10 machines and 20 dynamic new job arrivals, and the average value of exponential distribution between two successive job arrivals (E_{ave}) was 30. The earliness and tardiness penalties of the first 4000 epochs calculated by the proposed DDQN algorithm are shown in Figure 2. It can be seen from the curve that the target value drops smoothly and that the volatility decreases gradually with an increase in the training steps. The learning curve remains relatively stable after the 2500th epoch. This shows that the scheduling agent learns the appropriate dispatching rules according to the changes in the production states and this self-learning ability improves the adaptability for solving the DFJSP.

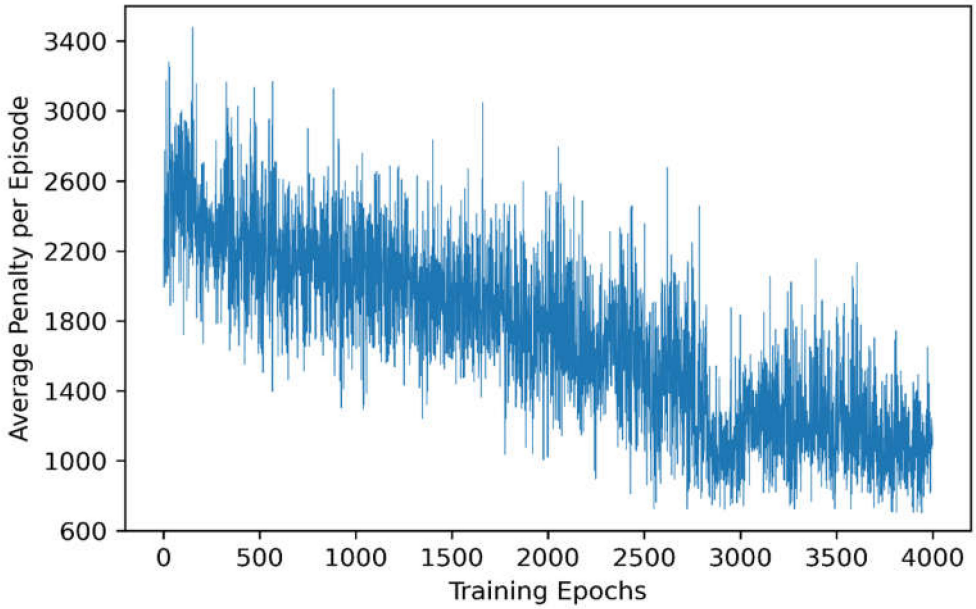

Figure 2. Average earliness and tardiness penalty at each training epoch.

4.2. Comparison between the Soft ε-greedy and ε-greedy Behavior Policies

4.2.1. Sensitivity of the Control Parameter μ

The control parameter μ in Equation (20) affects the performance of the algorithm proposed in this article. The larger the μ, the slower the linear decline in ε, which enhances exploration. On the contrary, the smaller the μ, the faster the linear decline in ε, which weakens exploration. To determine the appropriate value of μ, it was increased from 0.4 to 2.4 in steps of 0.2. At each parameter level, the trained deep reinforcement learning model was independently tested 30 times on an instance with 10 machines, 20 new job arrivals and E_{ave} set to 30. Figure 3 shows the box plots of the earliness and tardiness penalties for 30 trials with different values of μ, with the mean values marked by green triangles. It can be observed in the figure that μ = 1.8 achieved the lowest degree in terms of both the distribution range and the mean value of earliness and tardiness penalties. Therefore, the recommended value for μ is 1.8.

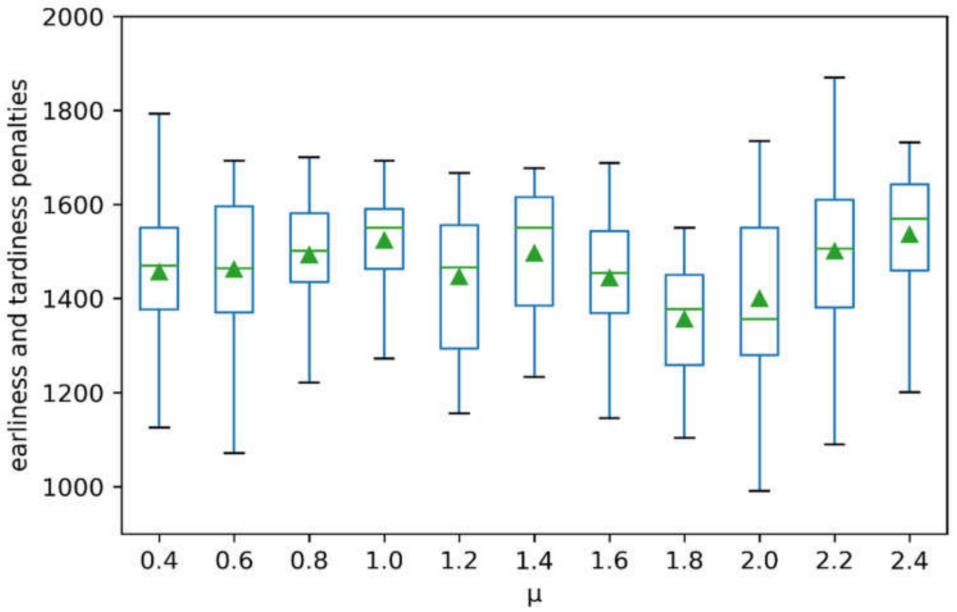

Figure 3. Box plots of earliness and tardiness penalties at different values of μ.

4.2.2. Comparison of Learning Rates

To demonstrate that the soft ε-greedy behavior policy can reasonably balance exploration and exploitation depending on the problem size, the DDQN was trained for an instance with 10 machines, 50 new job arrivals and E_{ave} set to 50. For the soft ε-greedy and ε-greedy behavior policy, ε anneals linearly from 1 to 0.1. The linear annealing rate of ε is $\frac{1}{(OP\_num)^{1.8}}$ for the soft ε-greedy policy, whereas the value is 0.001 for the ε-greedy policy. During training, the parameter μ was set to 1.8; the values of the other hyperparameters are listed in Table 3. Through a method from the DQN literature [23], a fixed set of states was collected by running a random policy before the training started and the average of the maximum predicted Q for these states was tracked. In Figure 4, the average predicted Q of the soft ε-greedy behavior policy is significantly higher than that of the ε-greedy policy, indicating that the linearly decreasing rate of ε was adjusted according to the problem size, thus achieving a better compromise between exploration and exploitation and maximizing the cumulative reward.

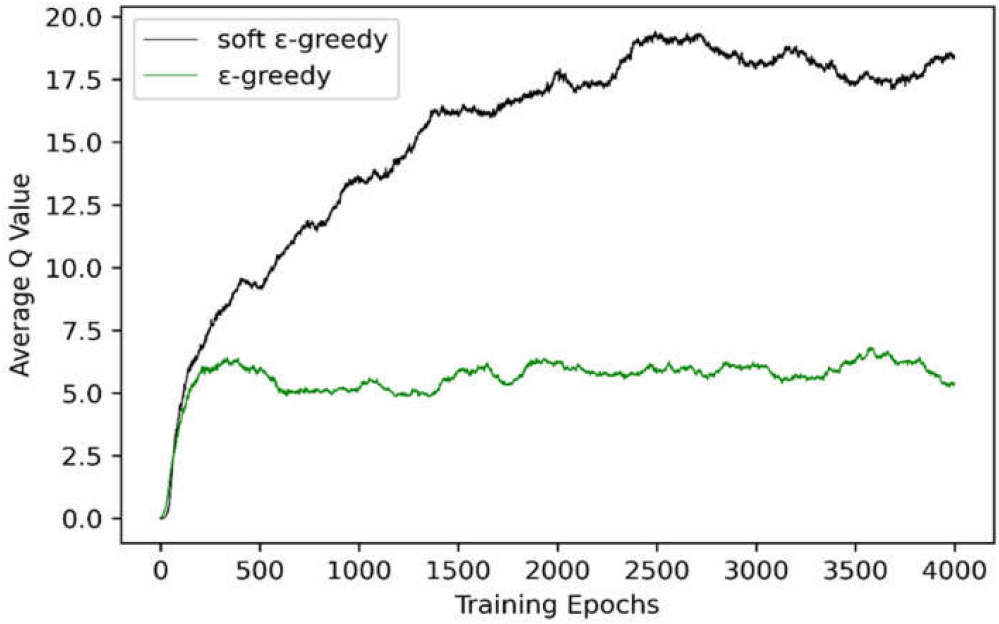

Figure 4. Average maximum predicted action value Q for each training epoch.

4.3. Comparison of DDQN with Other Methods

To verify the effectiveness and generalization of the proposed DDQN, the DFJSP was classified according to different parameter settings for E_{ave}, m and n_{add}. To simulate a real production environment, the number of operations, delivery factors and penalty coefficients were distributed randomly for each job, so 30 independent instances were generated for each type of DFJSP. The DDQN was compared with two other RL algorithms, DQN and SARSA; one of the most commonly used heuristics algorithms, FIFO; and a famous metaheuristic algorithm, GA. Moreover, the random action selection policy RA was designed to prove the learning ability of the agent. In each instance, the DDQN and the other algorithms were repeated independently 20 times. The mean values of the total earliness and tardiness penalty obtained by each method are shown in Table 5 and the best results are highlighted in bold font.

Table 5. Average total earliness and tardiness penalties of the different algorithms for each type of test (the best results are highlighted in bold font).

E_{ave}	m	n_{add}	DDQN	DQN	SARSA	FIFO	GA	RA
30	5	20	$\mathbf{5.17 \times 10^3}$	6.63×10^3	7.27×10^3	1.35×10^4	1.07×10^4	1.41×10^4
		30	1.38×10^4	1.93×10^4	2.02×10^4	3.10×10^4	$\mathbf{1.29 \times 10^4}$	2.18×10^4
		50	$\mathbf{3.68 \times 10^4}$	4.42×10^4	4.64×10^4	7.02×10^4	4.97×10^4	5.55×10^4
		100	1.62×10^5	$\mathbf{1.61 \times 10^5}$	1.98×10^5	2.57×10^5	3.15×10^5	2.18×10^5
	10	20	$\mathbf{1.48 \times 10^3}$	2.26×10^3	3.02×10^3	4.42×10^3	3.53×10^3	4.08×10^3
		30	$\mathbf{3.43 \times 10^3}$	4.20×10^3	5.06×10^3	7.14×10^3	6.98×10^3	8.24×10^3
		50	$\mathbf{9.26 \times 10^3}$	1.12×10^4	1.15×10^4	1.78×10^4	1.38×10^4	1.90×10^4
		100	$\mathbf{4.57 \times 10^4}$	4.95×10^4	8.22×10^4	7.81×10^4	6.60×10^4	8.74×10^4
	30	20	$\mathbf{5.31 \times 10^3}$	6.10×10^3	6.29×10^3	6.11×10^3	6.41×10^3	6.40×10^3
		30	8.18×10^3	1.08×10^4	1.13×10^4	1.09×10^4	$\mathbf{8.07 \times 10^3}$	1.14×10^4
		50	$\mathbf{8.03 \times 10^3}$	1.31×10^4	1.43×10^4	1.35×10^4	1.09×10^4	1.52×10^4
		100	$\mathbf{1.63 \times 10^4}$	1.90×10^4	2.08×10^4	2.12×10^4	2.21×10^4	2.17×10^4
50	5	20	1.06×10^4	1.38×10^4	$\mathbf{9.78 \times 10^3}$	2.04×10^4	1.66×10^4	1.81×10^4
		30	$\mathbf{1.94 \times 10^4}$	2.34×10^4	2.91×10^4	3.69×10^4	2.65×10^4	3.07×10^4
		50	5.50×10^4	$\mathbf{5.00 \times 10^4}$	7.02×10^4	8.33×10^4	6.92×10^4	8.00×10^4
		100	$\mathbf{1.61 \times 10^5}$	1.86×10^5	2.26×10^5	2.88×10^5	2.64×10^5	2.27×10^5
	10	20	$\mathbf{1.67 \times 10^3}$	2.38×10^3	3.96×10^3	4.76×10^3	3.99×10^3	4.46×10^3
		30	$\mathbf{3.29 \times 10^3}$	4.22×10^3	8.13×10^3	7.61×10^3	7.46×10^3	9.18×10^3
		50	1.40×10^4	$\mathbf{1.35 \times 10^4}$	2.25×10^4	2.33×10^4	1.79×10^4	2.47×10^4
		100	$\mathbf{5.88 \times 10^4}$	6.46×10^4	7.24×10^4	9.64×10^4	7.72×10^4	9.18×10^4
	30	20	7.42×10^3	9.82×10^3	1.03×10^4	$\mathbf{7.30 \times 10^3}$	1.05×10^4	1.04×10^4
		30	$\mathbf{9.06 \times 10^3}$	1.20×10^4	1.25×10^4	1.23×10^4	1.20×10^4	1.26×10^4
		50	$\mathbf{1.27 \times 10^4}$	1.52×10^4	1.63×10^4	1.66×10^4	1.64×10^4	1.67×10^4
		100	$\mathbf{1.85 \times 10^4}$	2.22×10^4	2.29×10^4	2.33×10^4	2.15×10^4	2.47×10^4
100	5	20	$\mathbf{9.10 \times 10^3}$	1.20×10^4	1.38×10^4	2.12×10^4	1.47×10^4	1.95×10^4
		30	2.21×10^4	$\mathbf{2.09 \times 10^4}$	2.59×10^4	3.35×10^4	2.60×10^4	2.94×10^4
		50	$\mathbf{5.11 \times 10^4}$	5.50×10^4	5.80×10^4	9.16×10^4	6.72×10^4	7.46×10^4
		100	$\mathbf{1.88 \times 10^5}$	2.14×10^5	2.17×10^5	3.61×10^5	2.64×10^5	2.77×10^5
	10	20	$\mathbf{3.68 \times 10^3}$	4.42×10^3	4.98×10^3	6.67×10^3	6.35×10^3	6.95×10^3
		30	$\mathbf{4.14 \times 10^3}$	5.28×10^3	8.79×10^3	9.40×10^3	6.53×10^3	1.05×10^4
		50	1.19×10^4	1.62×10^4	$\mathbf{1.17 \times 10^4}$	2.35×10^4	1.84×10^4	2.71×10^4
		100	$\mathbf{6.03 \times 10^4}$	6.58×10^4	6.74×10^4	9.72×10^4	7.55×10^4	9.50×10^4
	30	20	8.58×10^3	$\mathbf{8.49 \times 10^3}$	1.12×10^4	1.18×10^4	1.14×10^4	1.17×10^4
		30	$\mathbf{1.10 \times 10^4}$	1.40×10^4	1.43×10^4	1.47×10^4	1.50×10^4	1.44×10^4
		50	$\mathbf{1.34 \times 10^4}$	1.83×10^4	1.87×10^4	2.02×10^4	2.02×10^4	1.98×10^4
		100	$\mathbf{2.24 \times 10^4}$	2.66×10^4	2.71×10^4	2.91×10^4	2.84×10^4	3.07×10^4

In this study, the only difference between the DQN and the DDQN was the target y_i. Moreover, during training, ε decreased linearly from 1 to 0.1 in the DQN, while ε decreased from 1 to 0.01 in the DDQN. During testing, ε was fixed at 0.001.

In SARSA, in order to discretize the production state space reasonably, the neural network with a self-organizing mapping layer (SOM) from [19] was used to divide the state features into nine discrete states. The SARSA agent had the same action set of dispatching rules used in this study at each discrete state. A Q table was maintained that contained 9×6 Q-values for the state–action pairs and the SARSA agent was trained to learn the policies linearly.

In GA [30], the chromosomes of the FJSP were encoded in the form of operation sequence (OS) and machine assignment (MS). In order to improve the response speed of production events, every individual of the initial population was randomly generated. The selection operation adopted a combination of the roulette wheel and the elite retention

strategy. In the crossover operation, a uniform crossover was applied for MS and a precedence preserving order-based crossover (POX) was used for OS. The MS of the mutation operator adopted a multi-round single point exchange mutation and the OS part adopted a neighborhood search mutation. The hyperparameter settings were the population size $N = 100$, the number of iterations $I = 200$, the crossover probability $p_c = 0.8$ and the mutation probability $p_m = 0.01$.

FIFO choses the next operation of the earliest arriving job from among the unfinished jobs and the selected operation was assigned to the machine with the smallest sum of available time and processing time among the suitable machines. The RA was used to randomly select a dispatching rule at each rescheduling point.

In order to show the solution quality of the DDQN designed in this study, the average earliness and tardiness penalties of all the algorithms compared for all test instances were calculated according to Table 5, as shown in Figure 5. As can be seen in Figure 5, the DDQN outperformed the competing methods. The performance in terms of solution quality was normalized with respect to the average penalty of RA (that is, 100%). Note that the normalized performance of other algorithms, expressed as a percentage, was calculated as 100% * (algorithm penalty—RA penalty)/RA penalty. The normalized performance of the other algorithms was 33.80% (DDQN), 25.75% (DQN), 16.60% (SARSA), 1.68% (GA) and −13.32% (FIFO). It can also be seen that reinforcement learning (DDQN, DQN and SARSA) outperformed the competing methods (GA, RA and FIFO) in almost all instances, whereas deep reinforcement learning (DDQN and DQN) obtained a better solution quality than standard reinforcement learning with linear function approximation (SARSA).

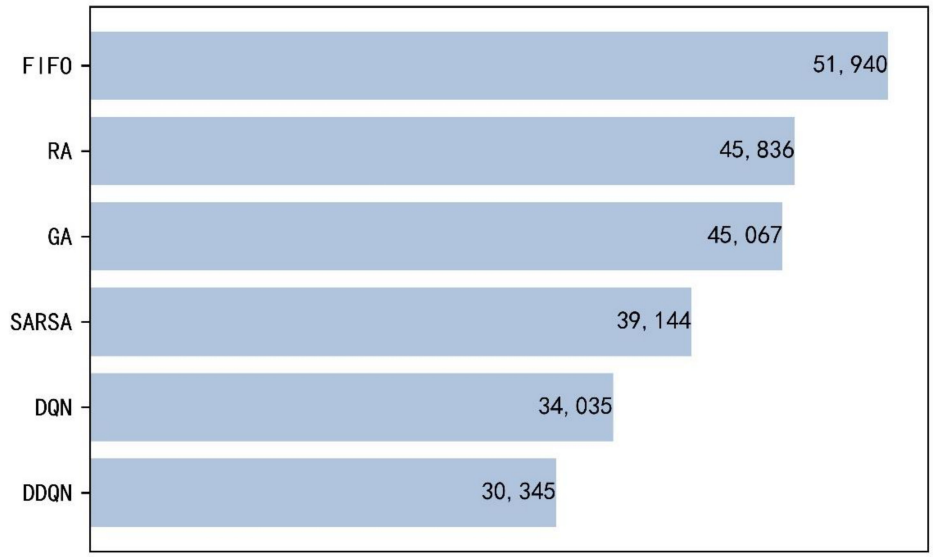

Figure 5. Average total earliness and tardiness penalties of the algorithms compared for all test instances.

To verify the generalization ability of DRL, the winning rate was defined, which was calculated as the number of instances in which the method achieved the best result divided by the number of all instances. Figure 6 presents the winning rate of all the algorithms calculated according to Table 4. Of the 36 test instances, the DDQN had the best results in 28 kinds of instances and the winning rate was 72.22%. The DQN has the smallest penalty for five scheduling problems and the winning rate was 13.89%. Both SARSA and GA had the minimum target value for two instances and the winning rate was 5.56%. The target value of FIFO was the smallest for one scheduling problem, with a winning rate of 2.78%. The winning rate of the random action strategy was 0%. The DDQN proposed in this study had the highest winning rate and performed at a level that was superior to the compared algorithms in general. Compared with RA, the DDQN obtained a lower total penalty in all

test instances, demonstrating its ability to master difficult control policies for solving the DFJSP, to determine the proper dispatching rule at each rescheduling point.

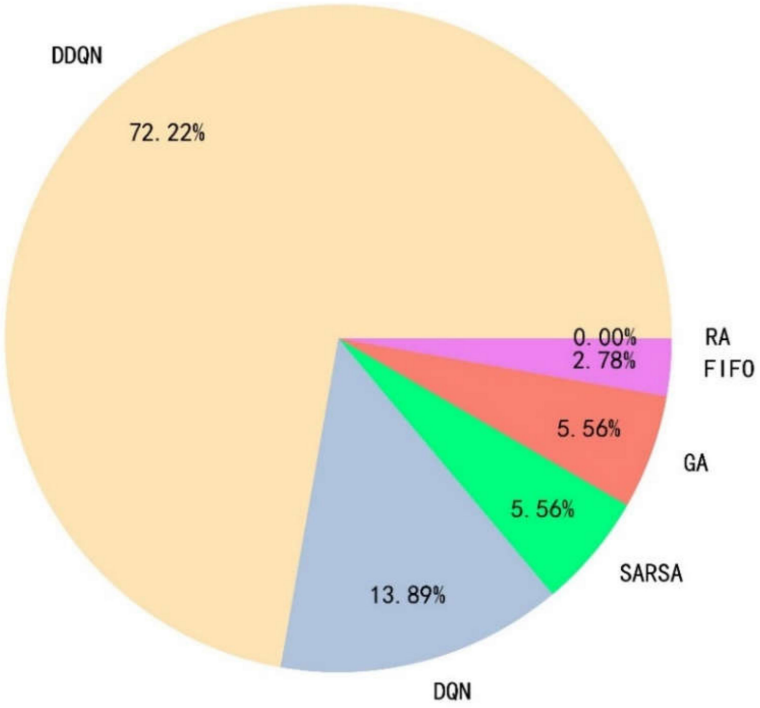

Figure 6. Winning rate of the DDQN and other algorithms, calculated according to Table 5.

It can be seen that on the whole, the DDQN clearly outperformed the other five methods in terms of solution quality and generalization. It is competent for solving the DFJSP with random job arrival to minimize penalties for earliness and tardiness. Reinforcement learning solves the scheduling problem as a Markov decision-making process and determines the ongoing action according to the production state to optimize the scheduling objectives.

5. Discussion

In this study, the DDQN was developed for the dynamic flexible job shop scheduling problem with random job arrival, aiming at optimizing the penalties for earliness and tardiness. In contrast to previous work [15–20], our approach provides DRL for the DFJSP in handling ongoing and weak correlation production states. Moreover, the soft ε-greedy strategy is designed to balance exploration and exploitation according to the problem scale, which improves the self-learning speed of the scheduling agent.

Because of the weak correlation between state features, the suitable number of dispatching actions and the DDQN architecture, the scheduling agent can achieve stable learning and convergence through training. Therefore, the training curve shows that the average penalties for earliness and tardiness drops smoothly with increasing training epochs. The proposed DRL allows the scheduling agent to learn the action–value function efficiently, to learn the optimal scheduling rules from the different production states. These comparison experiments show the DDQN-based scheduling agent outperforms the five compared methods in terms of solution quality and generalization.

The real-time optimization and decision-making of the DFJSP makes rapid and scientific response to customer orders and production emergencies and realizes intelligent matching between dispersed resources such as manpower, materials and machines. It improves the on-time delivery rate and reduces inventory and costs of enterprises. The proposed DRL provides reliable and robust scheduling schemes and meets customized production requirements according to dynamic changes in the production process. Cost, inventory,

procurement, sales and transportation plans are automatically generated, which drives various management modules of enterprises around production. Therefore, the factory has the capabilities of self-learning and self-adaptation to realize intelligent decision-making.

For future work, more dynamic events like rush order insertions, stochastic processing time and machine breakdowns are worthy of investigation. Other objectives, such as machine utilization rate, energy consumption and makespan, will be considered to validate the generality of the proposed DDQN over different objectives. Meanwhile, there is not a single dispatching rule that performs well for all production environments [20], so the number of actions should be increased for a more general agent. However, a general state value that is shared across many similar actions is learned in many control tasks with large action spaces [31] and, consequently, introducing the dueling architecture [31] can be useful to improve the performance of the DDQN. In addition, since the DDQN is based on experience replay, which limits the methods to off-policy learning algorithms, we will apply state-of-the-art on-policy RL algorithms, such as an asynchronous advantage actor-critic algorithm (A3C) [32,33] and proximal policy optimization for solving the DFJSP.

6. Conclusions

This study introduced deep reinforcement learning for solving the DFJSP with random job arrival, to achieve the production goal of minimizing the penalties for earliness and tardiness. On the basis of constructing a mathematical model, the DDQN architecture of the DFJSP was constructed and the state features, the actions, the reward and the soft ε-greedy behavior policy were designed accordingly. Our approach indicated that the proposed DRL gave state-of-the-art results in 28 out of 36 production instances, compared with DQN, SARSA, FIFO, GA and RA, without changing the architecture or hyperparameters of the deep network.

Supplementary Materials: The training and test results, and the video of solving the DFJSP using the trained agent can be downloaded at: www.mdpi.com/2227-9717/10/4/760.

Author Contributions: Conceptualization, J.C. and D.Y.; methodology, J.C.; software, J.C.; validation, J.C., Y.H.; formal analysis, J.C. and D.Y.; investigation, W.H. and H.Y.; resources, J.C.; data curation, J.C.; writing—original draft preparation, J.C.; writing—review and editing, J.C., W.H. and H.Y.; visualization, J.C. and W.H.; supervision, J.C. and D.Y.; project administration, D.Y. and Y.H.; funding acquisition, D.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Science and Technology Special Project of China [2018ZX04032002], the Scientific Research Project of Fujian Province [JAT210946], and the General Scientific Research Project of Liaoning Province [LJKZ1414].

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All experiment datasets and the video of solving DFJSP using the trained agent are available at https://github.com/changjingru/DDQN-for-DFJSP (accessed on 11 March 2022).

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

References

1. Bouazza, W.; Sallez, Y.; Beldjilali, B. A distributed approach solving partially flexible job-shop scheduling problem with a Q-learning effect. *IFAC Pap.* **2017**, *50*, 15890–15895. [CrossRef]
2. Gao, K.Z.; Suganthan, P.; Chua, T.J.; Chong, C.S.; Cai, T.X.; Pan, Q.-K. A two-stage artificial bee colony algorithm scheduling flexible job-shop scheduling problem with new job insertion. *Expert Syst. Appl.* **2015**, *42*, 7652–7663. [CrossRef]
3. Wang, S.; Wan, J.; Li, D.; Zhang, C. Implementing Smart Factory of Industrie 4.0: An Outlook. *Int. J. Distrib. Sens. Netw.* **2016**, *12*, 3159805. [CrossRef]
4. Brucker, P.; Schlie, R. Job-shop scheduling with multi-purpose machines. *Computing* **1990**, *45*, 369–375. [CrossRef]
5. Garey, M.R.; Johnson, D.S.; Sethi, R. The Complexity of Flowshop and Jobshop Scheduling. *Math. Oper. Res.* **1976**, *1*, 117–129. [CrossRef]

6. Gao, K.; Yang, F.; Zhou, M.; Pan, Q.; Suganthan, P.N. Flexible Job-Shop Rescheduling for New Job Insertion by Using Discrete Jaya Algorithm. *IEEE Trans. Cybern.* **2018**, *49*, 1944–1955. [CrossRef] [PubMed]
7. 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.* **2020**, *59*, 5867–5883. [CrossRef]
8. Jain, V.; Raj, T. An adaptive neuro-fuzzy inference system for makespan estimation of flexible manufacturing system assembly shop: A case study. *Int. J. Syst. Assur. Eng. Manag.* **2018**, *9*, 1302–1314. [CrossRef]
9. Lawrence, S.R.; Sewell, E.C. Heuristic, optimal, static, and dynamic schedules when processing times are uncertain. *J. Oper. Manag.* **1997**, *15*, 71–82. [CrossRef]
10. Ning, T.; Jin, H.; Song, X.; Li, B. An improved quantum genetic algorithm based on MAGTD for dynamic FJSP. *J. Ambient Intell. Humaniz. Comput.* **2017**, *9*, 931–940. [CrossRef]
11. Nouiri, M.; Bekrar, A.; Trentesaux, D. Towards Energy Efficient Scheduling and Rescheduling for Dynamic Flexible Job Shop Problem. *IFAC Pap.* **2018**, *51*, 1275–1280. [CrossRef]
12. Wu, X.; Li, J.; Shen, X.; Zhao, N. NSGA-III for solving dynamic flexible job shop scheduling problem considering deterioration effect. *IET Collab. Intell. Manuf.* **2020**, *2*, 22–33. [CrossRef]
13. Cai, J.; Peng, Z.; Ding, S.; Sun, J. Problem-specific multi-objective invasive weed optimization algorithm for reconnaissance mission scheduling problem. *Comput. Ind. Eng.* **2021**, *157*, 107345. [CrossRef]
14. Staddon, J.E.R. The dynamics of behavior: Review of Sutton and Barto: Reinforcement Learning: An Introduction (2nd ed.). *J. Exp. Anal. Behav.* **2020**, *113*, 485–491. [CrossRef]
15. Wang, S.; Sun, S.; Zhou, B.; Xi, L.F. Q-Learning Based Dynamic Singe Machine Scheduling. *J. Shang Hai Jiao Tong Univ.* **2007**, *47*, 1227–1232.
16. Fonseca-Reyna, Y.C.; Martinez, Y.; Rodríguez-Sánchez, E.; Méndez-Hernández, B.; Coto-Palacio, L.J. An Improvement of Reinforcement Learning Approach to Permutational Flow Shop Scheduling Problem. In Proceedings of the 13th International Conference on Operations Research (ICOR 2018), Beijing, China, 7–9 July 2018.
17. Shahrabi, J.; Adibi, M.A.; Mahootchi, M. A reinforcement learning approach to parameter estimation in dynamic job shop scheduling. *Comput. Ind. Eng.* **2017**, *110*, 75–82. [CrossRef]
18. Wang, Y.-F. Adaptive job shop scheduling strategy based on weighted Q-learning algorithm. *J. Intell. Manuf.* **2018**, *31*, 417–432. [CrossRef]
19. Luo, S. Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. *Appl. Soft Comput.* **2020**, *91*, 106208. [CrossRef]
20. Luo, S.; Zhang, L.; Fan, Y. Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning. *Comput. Ind. Eng.* **2021**, *159*, 107489. [CrossRef]
21. Rosa, B.; Souza, M.; De Souza, S. Algorithms based on VNS for solving the Single Machine Scheduling Problem with Earliness and Tardiness Penalties. *Electron. Notes Discret. Math.* **2018**, *66*, 47–54. [CrossRef]
22. Jing, X.-L.; Pan, Q.-K.; Gao, L.; Wang, Y.-L. An effective Iterated Greedy algorithm for the distributed permutation flowshop scheduling with due windows. *Appl. Soft Comput.* **2020**, *96*, 106629. [CrossRef]
23. 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.
24. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
25. Hasselt, H.V.; Guez, A.; Silver, D. Deep Reinforcement Learning with Double Q-learning. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16), Phoenix, AZ, USA, 12–17 February 2016.
26. van Hasselt, H. Double Q-learning. In *Advances in Neural Information Processing Systems*; Curran Associates, Inc.: New York, NY, USA, 2010; Volume 23, pp. 2613–2621.
27. Vera, F. Performing Deep Recurrent Double Q-Learning for Atari Games. *arXiv* **2019**, arXiv:1908.06040.
28. Shi, D.; Fan, W.; Xiao, Y.; Lin, T.; Xing, C. Intelligent scheduling of discrete automated production line via deep reinforcement learning. *Int. J. Prod. Res.* **2020**, *58*, 3362–3380. [CrossRef]
29. Shiue, Y.-R.; Lee, K.-C.; Su, C.-T. Real-time scheduling for a smart factory using a reinforcement learning approach. *Comput. Ind. Eng.* **2018**, *125*, 604–614. [CrossRef]
30. Yang, S.; Xu, Z.; Wang, J. Intelligent Decision-Making of Scheduling for Dynamic Permutation Flowshop via Deep Reinforcement Learning. *Sensors* **2021**, *21*, 1019. [CrossRef]
31. Wang, Z.Y.; Schaul, T.; Hessel, M.; Hasselt, H.V.; Lanctot, M.; Freitas, N.D. Dueling Network Architectures for Deep Reinforcement Learning. *arXiv* **2016**, arXiv:1511.06581.
32. Chen, T.; Liu, J.-Q.; Li, H.; Wang, S.-R.; Niu, W.-J.; Tong, E.-D.; Chang, L.; Chen, Q.A.; Li, G. Robustness Assessment of Asynchronous Advantage Actor-Critic Based on Dynamic Skewness and Sparseness Computation: A Parallel Computing View. *J. Comput. Sci. Technol.* **2021**, *36*, 1002–1021. [CrossRef]
33. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous Methods for Deep Reinforcement Learning. *arXiv* **2016**, arXiv:1602.01783.