processes

Article

Efficient Multi-Objective Optimization on Dynamic Flexible Job Shop Scheduling Using Deep Reinforcement Learning Approach

Zufa Wu [1], Hongbo Fan [1,2,*], Yimeng Sun [1] and Manyu Peng [1]

1 Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650504, China; wuzufa0603@stu.kust.edu.cn (Z.W.); 15836501728@163.com (Y.S.); 20202204327@stu.kust.edu.cn (M.P.)
2 Faculty of Modern Agricultural Engineering, Kunming University of Science and Technology, Kunming 650500, China
* Correspondence: 20110258@kust.edu.cn

Abstract: Previous research focuses on approaches of deep reinforcement learning (DRL) to optimize diverse types of the single-objective dynamic flexible job shop scheduling problem (DFJSP), e.g., energy consumption, earliness and tardiness penalty and machine utilization rate, which gain many improvements in terms of objective metrics in comparison with metaheuristic algorithms such as GA (genetic algorithm) and dispatching rules such as MRT (most remaining time first). However, single-objective optimization in the job shop floor cannot satisfy the requirements of modern smart manufacturing systems, and the multiple-objective DFJSP has become mainstream and the core of intelligent workshops. A complex production environment in a real-world factory causes scheduling entities to have sophisticated characteristics, e.g., a job's non-uniform processing time, uncertainty of the operation number and restraint of the due time, avoidance of the single machine's prolonged slack time as well as overweight load, which make a method of the combination of dispatching rules in DRL brought up to adapt to the manufacturing environment at different rescheduling points and accumulate maximum rewards for a global optimum. In our work, we apply the structure of a dual layer DDQN (DLDDQN) to solve the DFJSP in real time with new job arrivals, and two objectives are optimized simultaneously, i.e., the minimization of the delay time sum and makespan. The framework includes two layers (agents): the higher one is named as a goal selector, which utilizes DDQN as a function approximator for selecting one reward form from six proposed ones that embody the two optimization objectives, while the lower one, called an actuator, utilizes DDQN to decide on an optimal rule that has a maximum Q value. The generated benchmark instances trained in our framework converged perfectly, and the comparative experiments validated the superiority and generality of the proposed DLDDQN.

Keywords: deep reinforcement learning; multi-objective optimization; delay time sum; makespan; dual layer deep Q-network; global optimum; dynamic flexible job shop scheduling

check for updates

Citation: Wu, Z.; Fan, H.; Sun, Y.; Peng, M. Efficient Multi-Objective Optimization on Dynamic Flexible Job Shop Scheduling Using Deep Reinforcement Learning Approach. *Processes* **2023**, *11*, 2018. https://doi.org/10.3390/pr11072018

Academic Editors: Ming-Jong Tsai and Ricky Min-Fan Lee

Received: 24 May 2023
Revised: 19 June 2023
Accepted: 26 June 2023
Published: 6 July 2023

1. Introduction

With the development of artificial intelligence (AI) technologies becoming increasingly mature and data from industrial production growing substantially, production requirements in Industry 4.0 demand a higher level of task arrangement precision in job shop scheduling. However, the demand is unable to be achieved if applied with the traditional dispatching rules because the same rule is unchangeable during the whole production process and makes the workshop unlikely to utilize as many machines as possible according to the temporary scheduling situation, thus increasing the makespan and tardiness and causing other adverse metrics [1]. In a modern manufacturing system, uncertainties and complexities of factory shop floor production, for example, unexpected events such as random job arrivals, the modification or cancellation of jobs, the breakdown and recovery of machines, and the inconsistent processing time of operations, boost the advancement of intelligent dynamic scheduling schemes [2]. In order to develop a smart factory and make real-time scheduling intelligent, it is crucial to propose feasible solutions to the dynamic flexible job shop problem (DFJSP).

The DFJSP evolved from simpler scheduling management problems, i.e., the job shop scheduling problem (JSP) and the flexible job shop scheduling problem (FJSP), to meet the more complex and efficient productivity requirements of the contemporary shop floor. Scheduling optimization is a control process for making an optimal current decision for a global optimum with a limited resource allocation [3]. For variant production scenarios, it is indispensable to independently propose the corresponding appropriate multi-objective optimization schemes. In the JSP [4–7], the product processing of each operation can only be appointed one machine, while the addition of the flexible characteristic in the FJSP [8–11] can make it more conformed to a real-world production circumstance in which all operations of a product can be processed using more than one available machine. The dynamic JSP or FJSP emphasizes the adaptability of the trained model to handle interruptive incidents and still obtain a complete scheduling solution that is approximately optimal.

The DFJSP is known as an NP-hard problem in the computer algorithm [12]. The mainstream method used to obtain the best solution in recent years consists in applying biology-based algorithms, i.e., meta-heuristic approaches, such as GA series, simulated annealing algorithms, particle swarm optimization (PSO), ant colony optimization (ACO), etc. [13]. However, even though biology-based algorithms perform better in globally optimizing objectives than simple dispatching rules, they are not time-efficient, they do not provide real-time responses to schedules encountered with random dynamic incidents [14,15], and they are still unable to perform better than DRL-based methods. DRL is a promising method used to automatically control objects to fulfill tasks in the most intelligent way efficiently. For example, in order to resolve the optimal machine sequences, DRL has been commonly used in combination with the optimization of traditional industrial production. In a production system, DRL can be utilized in enormous manufacturing scenarios, including process control, production scheduling and dispatching, logistics, assembly, robotics, energy management, and so forth [16]. The prevalence of DRL-related applications highlights the reliance of running algorithms on DRL's intra-mechanism—self-adaptability.

1.1. Related Works

From the methodology of the deep reinforcement learning concept, DRL can be classified into two types, including value-based approaches such as SARSA or DQN and policy-based approaches such as Actor-Critic [17] or Proximal Policy Optimization (PPO) [18]. According to the application experiences of industrial manufacturing, there are no specific judgments or reasonings about which one performs better than the other, and both have weaknesses and advantages. For example, in a continuous control problem, the PPO method can be more promising than DQN or improved DQN, while in combinatorial optimization, most research prioritizes value-based methods instead of policy-based ones. Policy methods are sensitive to the newest experiences and to strengthen the interaction with the environment. Policy-based methods make decisions based on the probability distribution of actions. In cases where there is a requirement of continuous action, there are too many discrete actions and there is an environment with stochasticity, policy-based methods are more advisable. Value-based methods rely on experience samples from replay memory to learn the policy, and their work may increase the computation overhead.

Liu et al. (2020) [15] utilized actor-critic deep reinforcement learning to deal with the job shop scheduling problem (JSSP). A parallel training method is proposed combining an asynchronous update with the deep deterministic policy gradient (DDPG) in order to train the model. Luo et al. (2021) [19] devised a framework of a two-hierarchy deep Q network (THDQN) for the dynamic multi-objective flexible job shop scheduling problem

(DMOFJSP) with the high-level DQN to determine the temporary goal to guide the behavior of low-level DQN and the low-level DQN acts to decide on the best dispatching rule that achieves the given goal. Hu et al. (2020) [20] exploited an adaptive deep-reinforcement-learning-based AGV real-time scheduling approach to the flexible job shop in which DQN is used to achieve the optimal mixed rule policy, and suitable dispatching rules and AGVs are selected to execute scheduling. Lei et al. (2022) [21] presented an end-to-end deep reinforcement framework to learn a policy that solves the FJSP with the use of a graph neural network in which multi-pointer graph networks (MPGNs) and a muti-PPO training algorithm are developed to learn two sub-policies, i.e., an operation action policy and a machine action policy. Abebaw et al. (2023) [22] considered the JSSP as an iterative decision-making problem, and a DDQN is utilized for training the model and learning an optimal policy in which six continuous state features are formulated to record the production environment; an epsilon-greedy strategy is used on the action selection; furthermore, the reward and the penalty of the evaluation metric are designed. Zhang et al. (2022) [23] used the PPO algorithm in the DRL framework to tackle the dynamic scheduling problem in a job shop manufacturing system with an unexpected event of the machine failure in which the transport agent is required to dispatch jobs/orders to machines then to sinks from machines after the task of jobs is completed. The proposed framework was validated based on the real-world job shop manufacturing system. Liu et al. (2022) [24] worked out a hierarchical and distributed scheduling framework to solve the DFJSP using the DDQN algorithm to train the scheduling agents with the sequencing agent responsible for selecting a job from the queue to be processed by the machine when the machine is idle or at least two jobs are queuing, while the routing agent fulfills to decide on which machine's queue in the work center the job should join upon when arriving by the predetermined order. Zhang et al. (2020) [25] developed an end-to-end deep reinforcement learning agent to automatically learn priority dispatching rules (PDRs) for solving the real-world JSSP, and a graph-neural-network-based scheme was proposed to embed the states encountered during solving with the disjunctive graph representation of the JSSP. Luo et al. (2021) [26] applied a double loop deep Q-network method with exploration and exploitation loops to solve the dynamic JSSP in which the single-agent system integrates the global search ability of the exploration loop and the local optimal convergence capability of the exploitation loop to facilitate DQN to find a globally optimum solution. The action space in the proposed framework is directly oriented to jobs instead of dispatching rules. Wang et al. (2021) [27] adopted the PPO to find an optimal scheduling policy in the JSSP of smart manufacturing and deal with dimension disaster existing in the state and action space when the problem scale becomes especially large in industrial production. The proposed framework has been verified with adaptability and generalizability to schedule dynamically encountered unexpected events. Table 1 shows the latest articles discriminated through component classification with more precise and superior solutions via DRL-based optimization algorithms.

Table 1. An overview of classifications of up-to-date scheduling literature from the process of DRL solving problems.

Study	Problem	State	Objective	DRL	Benchmark	Network	Random Event
Liu et al. (2020) [15]	JSP	Discrete	Makespan	Actor-Critic	OR-Library	DNN	Machine breakdown; sudden additional order
Luo et al. (2021) [19]	FJSP	Continuous	Tardiness; machine utilization rate	Hierarchy DQN	Generation	DNN	Random job insertion

Table 1. *Cont.*

Study	Problem	State	Objective	DRL	Benchmark	Network	Random Event
Hu et al. (2020) [20]	FJSP	Continuous	Delay ratio; makespan	DQN	Generation	DNN	None
Lei et al. (2022) [21]	FJSP	Discrete	Makespan	Multi-PPO, MPGN	Generation; instances of other papers	GNN	None
Abebaw et al. (2023) [22]	JSP	Continuous	Makespan	DDQN	OR-Library	DNN	Machine breakdown; job rework
Zhang et al. (2022) [23]	JSP	Continuous	Average machine utilization; order wait time	PPO	Generation	DNN	Machine failure
Liu et al. (2022) [24]	FJSP	Continuous	Makespan	DDQN	Generation	DNN	Machine breakdown
Zhang et al. (2020) [25]	JSP	Discrete	Makespan	PPO	Generation; instances of other papers	GNN	Job arriving on-the-fly; random machine breakdown
Luo et al. (2021) [26]	JSP	Continuous	Makespan	Double Loop DQN	OR-Library	DNN	Random job insertion
Wang et al. (2021) [27]	JSP	Discrete	Makespan	PPO	OR-Library; generation	DNN	Machine breakdown; processing time change
This paper	FJSP	Continuous	Makespan; delay time sum	Dual Layer DDQN	Generation	DNN	Random job incoming

Markov decision process is the core of self-adaptability in DRL. Markov formula is defined as a five-element tuple, i.e., $MDP = (S_t, A_t, R_t, S_{t+1}, Info)$ [28], in which S_t denotes the problem's state set; A_t is the action that the agent takes to resolve the problem, such as dispatching rules used to schedule in the DFJSP; R_t is the reward that the environment feeds back to the agent for the current result of the action A_t, and causes to a dynamical readjustment to the deep neural network, a decision function approximator [29]; S_{t+1} is the obtained next state after the environment is updated; $Info$ is the other auxiliary fragments used to help training. The complex problem of the job shop scheduling can be decomposed into a representation of the MDP by extracting critical features as a state space, designing multiple dispatching rules as an action space that are available when the agent is situated in all possible environments, and generating a reward function that aims to direct the neural network to choose the proper action with consequent high results under the current environment. One of the obvious characteristics of DRL is that all elements in the MDP tuple should be formulated in order to optimize the objectives of the specified application [30], such as minimizing energy consumption, the makespan or job delay rate, and maximizing product processing profit in the workshop scheduling scene.

Tabular Q-learning is the earliest form of the value-based DRL method [31]. It requires a table to map (S_t, A_t) with the Q value calculated by Bellman approximation [32], through which the action with the maximum Q value will be chosen to execute the task, and the value will be updated by the same iterative approximation. The shortcomings of the table-based Q-learning are evident. First, the value iteration method has to store all of the (S_t, A_t) pairs in the memory table and perform a complete loop on each item of pairs to find the best action and execute updating, which will be a disaster if states at a large scale in amount, demanding too high memory usage(usually >32 G) and causing redundant iterations over unused (S_t, A_t) pairs that are a waste of time. Second, the primitive method is constrained to be used under the discreet state and action space, which will not be a proper method to handle problems with a continuous state or action space.

Research spots of the DFJSP of the past are apt to the optimization of a single objective applied with only one agent of the different network architecture, and innovations published about the multi-objective scheduling emerged in limited fashion. Therefore, solving the DFJSP with several optimization objectives will advance the research pace of workshop scheduling for realistic requirements to reduce the production cost of plants and increase product profit. Moreover, the single DQN method orients itself with one neural network in only one agent, which merely corresponds to one optimization objective instead of several ones since different objectives have different reward functions and behavioral strategies [19]. State-of-art studies have recently developed an HRL (hierarchical reinforcement learning) framework to make a compromise between multiple objectives through the cooperation of bi-layer agents [33,34]. Based on HRL, a dual-layer DDQN framework is proposed in this paper: the higher layer, called the controller, takes responsibility for learning policy to decide on the higher goal as the input of the lower neural network; the lower layer, called the actuator, learns a policy to select an action that best suits the current scheduling environment and gains the most rewards. Multi-objective optimization means better adaptability to input states and more precise decisions to acquire more rewards, which can be unsatisfied with only a single neural network. Thus, this work demands two neural networks, with the higher one for a dynamic tune-up of the optimization goal and the lower one for selecting a compositive dispatching rule under the guidance of the given goal, which finally achieves a good compromise between objectives.

1.2. Contributions

As mentioned above, with drawbacks of heuristic rules (FIFO, first in first out; EDD, earliest due date first; and LWT, the longest waiting time first) and metaheuristic algorithms that are unable to schedule in real time and unfit to dynamic environments changes [20], this article applies value-based DRL, i.e., a two-layer Double Deep Q Network, to tackle the multi-objective DFJSP with a random dynamic incident of new job incomings. Experimental results show that the proposed DRL framework performs better than other non-DRL-based implementations, regardless of the generalization ability to output near-optimal results or the solution quality [35]. Contributions of this work can be listed as follows.

(1) A framework of two layers of DDQN is proposed to optimize the DFJSP of multiple objectives, corresponding to two connected agents with respective task divisions. The goal selector that is the higher DDQN intends to output an optimization goal with a five-element state vector as the feature input. The output of the higher network is one specific reward form, which is applied as one input state of the lower DDQN, along with the goal selector's five input states. With six states input, the actuator (the lower DDQN) outputs a specific dispatching rule maximizing reward scores.

(2) This work specifies optimization objectives, i.e., minimization of the job's delay time sum and the machine's completion time (makespan). Seven compositive dispatching rules and six continuous states are proposed to outline the DFJSP environment. The six states are represented as formulas, and their correlations are erased. The reward function is realized by the goal space with six reward forms, covering the higher DDQN's goal output.

(3) The simulation experiments are carried out to appear the production environment and realize the dynamic scheduling of flexible workshops based on the proposed mathematical model in Section 2 and DRL architecture in Section 3. A large number of datasets of different production configurations considerably generalize the trained DLDDQN model, enabling it to fit raw test examples and achieve expected objective optimizations.

The remaining paper composition can be summarized as follows. Section 2 details the establishment of the mathematical modeling of the DFJSP. Section 3 illustrates the construction and realization of the DLDDQN architecture. Section 4 interprets the generation of numerical examples and undertakes experiments to testify to the generality and superiority of the proposed DLDDQN on two optimization objectives through comparisons with proposed dispatching and heuristic rules, meta-heuristic algorithms, and value-based DRL methods. Finally, in Section 5, conclusions of the research and innovative directions for future works are demonstrated.

2. Problem Formulation

The DFJSP is a typical multi-machine scheduling problem in which the proposed DLDDQN framework is required to decide on a selected job and a machine to execute the following operation at each rescheduling point t. Jobs of the DFJSP have a predetermined sequence of operations, and the solution to the DFJSP is to allocate operations of each job to the most suited machine's waiting queue for objective optimizations. Several premises of the commonsense level should be considered in the process of finding a solution in a precision optimization as the followings:

(1) Only an operation of one job can be processed at the arranged machine at a time.
(2) The machine's operation cannot be stopped without completing the running operation, following the atomicity principle.
(3) Jobs are processed following the operation sequence without being skipped or randomly chosen.
(4) The travel time between two consecutive operations and the machine startup time are both negligible.
(5) A job's unprocessed operations cannot be canceled, and the job processing quality caters to standard requirements.
(6) The buffer size is assumed infinite.

Mathematical Representation

J_i: index of a job, $i \in \{1, 2, \ldots\ldots, n\}$, $n =$ the total number of all jobs;

M_k: index of a machine, $k \in \{1, 2, \ldots\ldots, m\}$, $m =$ the total number of all machines;

$O_{i,j}$: the jth operation of job J_i, $j \in \{1, 2, \ldots\ldots, n_i\}$, $n_i =$ the total number of operations of job J_i;

$S_{i,j,k}$: equal to 1 if the kth machine is available to the $O_{i,j}$, otherwise equal to 0;

$R_{i,j,k}$: equal to 1 if the kth machine is allocated to run the $O_{i,j}$, otherwise equal to 0;

$t_{i,j,k}$: the kth machine's processing time of the $O_{i,j}$, $t_{i,j,k} \in \{-1\ or\ N^*\}$, in which $t_{i,j,k} = -1$ if the kth machine is not applicable to the $O_{i,j}$;

$t_{i,j}$: the expected processing time of the $O_{i,j}$, $t_{i,j} = \frac{\sum_{k=1}^{m} t_{i,j,k} \times S_{i,j,k}}{\sum_{k=1}^{m} S_{i,j,k}}$;

$ST_{i,j}$: the start time of the $O_{i,j}$;

PT_i: the expected processing time of all operations of job J_i, $PT_i = \sum_{j=0}^{n_i} t_{i,j}$;

$C_{i,j}$: the completion time of the $O_{i,j}$;

C_i: the completion time of job J_i;

D_i: the due time of job J_i;

A_i: the arrival time of job J_i;

$m_{i,j}$: the available machine set $\{m_1, m_2, \ldots\ldots m_m\}$ for the $O_{i,j}$.

In our proposed framework, we aim to realize multi-objective optimization, including minimizing the delay time sum of all jobs and maximumly shortening the completion time of the task problem C_{max} by improving the machine uptime utilization. An objective mathematical function that the DLDDQN is trained to achieve with the random affair of new job arrivals at uncertain times is presented below.

The delay time of job J_i (DT_i) is calculated as follows:

$$DT_i = max(C_i - D_i,\ 0)$$

$$Y_{i,j,u,v} = \begin{cases} 1 & if\ the\ O_{i,j}\ processed\ precede\ the\ O_{u,v} \\ 0 & if\ the\ O_{i,j}\ runs\ behind\ the\ O_{u,v} \end{cases}$$

$$Makespan = min \max_{1 \le i \le n} C_{i,n_i} \tag{1}$$

$$DTS = \sum_{i=1}^{n} DT_i \tag{2}$$

$$ST_{i1} \ge 0, C_{i,j} > 0\ i = 1 \ldots n;\ j = 1, 2 \ldots n_i \tag{3}$$

$$\sum_{k=1}^{m_{i,j}} S_{i,j,k} R_{i,j,k} = 1\ i = 1 \ldots n;\ j = 1 \ldots n_i;\ k = 1 \ldots m \tag{4}$$

$$\begin{gathered}(ST_{u,v} - ST_{i,j} - t_{i,j,k}) S_{i,j,k} S_{u,v,k} Y_{i,j,u,v} + (ST_{i,j} - ST_{u,v} - t_{u,v,k}) S_{i,j,k} S_{u,v,k} Y_{i,j,u,v} \\ \ge 0\ i, u = 1 \ldots n;\ j, v = 1 \ldots n_i;\ k = 1 \ldots m\end{gathered} \tag{5}$$

$$C_{i,j} \le ST_{i,j+1},\ A_i \le ST_{i,j}\ i = 1 \ldots n;\ j = 1, \ldots n_i \tag{6}$$

To optimize targeted objectives is to minimize Formulas (1) and (2). Equation (3) represents that the start time of the first operation of each job should be a positive integer or zero; Equation (4) symbolizes that one of the operations of a job can have more than one available machines to process, but only one machine can be used among them; Equation (5) indicates that the interval between two successive operations assigned on one machine must be equal to or greater than zero, namely a machine can only process a job at a time; and Equation (6) points out that an operation can be schedule, only when the closest front operation has been completed or the first operation of the job has arrived.

3. Construction of DRL Components

3.1. DRL Preliminaries

According to the definition of Markov decision processes [36], a problem satisfied with MDPs can be broken into a five-element tuple: state, action, reward, next state, and done. In the standard DRL training, the agent first observes the environment state and applies a tactic such as the epsilon-greedy strategy to select an action. Then, the environment feedbacks a reward as a result of the execution of the (state, action) pair. Finally, the environment enters a new situation and observes the next state again. Added together, a five-element tuple is stored in the replay memory to be prepared to be sampled by the neural network for parameter updating. Repeat the above procedures until iterations arrive to stop, and then a deep neural network is trained well. The target of the two agents is to accumulate maximum reward $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ at time t with discount factor $\gamma \in (0, 1]$.

The tabular learning algorithm has been proved a method for an action-value function to converge to Q^*, the optimal value function proposed by Bellman. $Q^*(s, a) = max_\pi Q^\pi(s, a)$ returns the maximum action value if selecting an action in state s and following policy π. The optimal policy $\pi^*(s) = arg\underset{a}{max} Q^*(s, a)$ can be formed if the algorithm can obtain Q^*. The Bellman optimality equation can be formularized as Equation (7), as follows:

$$Q^*(s, a) = E_{s'}[r + \gamma \underset{a}{max} Q^*(s', a') | s, a] \tag{7}$$

The tabular learning does not suit the presentation of the large-volume state space due to the high memory size required to record state transition and the number and unit size of state discretization that involve precise issues kept known. Therefore, using a function approximator to represent the action-value function emerged, forming the typical DRL-based methods, including deep Q-learning, DQN, and DDQN.

Deep Q-learning [37] uses a neural network to approximate the optimal action-value function, $Q^*(s,\ a) \approx Q(s,\ a;\ \theta)$, in which the parameter θ is updated iteratively by minimizing the loss between the expected Q function value and the current Q function value. The loss value for updating the parameter θ in the neural network can be calculated by the following Equation (8) at step t.

$$L_t(\theta_t) = E(r + \gamma \underset{a_{t+1}}{max} Q(s_{t+1,}\ a_{t+1}; \theta_{t-1}) - Q(s, a; \theta_t))^2 \tag{8}$$

In Equation (8), s_{t+1} and a_{t+1} symbolize the state and action at the next step $t+1$, respectively.

A replay buffer is introduced into the DQN training to satisfy the requirement of SGD optimization, which asks to get independent and identically distributed training data (i.i.d). With a certain size of experiences stored in the replay buffer, independent data generated by the recent policy will still be fresh enough to be sampled for training. To obliterate the correlation between two consecutive transitions, make the training stable, and avoid chasing one's own tail, a copy of the online network ($Q(\theta)$), called a target network ($\hat{Q}(\theta^-)$), is introduced into the DQN and is used for the calculation of the maximum $Q(s_{t+1}, a_{t+1})$ value in the Bellman equation. The backup network's parameter θ^- is periodically synchronized by the online network's parameter θ, and the period is usually defined as a pretty large hyperparameter (e.g., 1k or 10k training episodes) [38]. The target value y_t at step t with parameter θ^- is calculated on the target network by Equation (9), as follows:

$$y_t = r_t + \gamma \underset{a'}{max} \hat{Q}(s_{t+1,}\ a'; {\theta_t}^-) \tag{9}$$

The DQN, in practical use, can have a poor performance in Q value estimation, which tends to be high beyond the real value. The problem is caused by the same value used to select and evaluate an action in the max operation. The DDQN solves the problem by decoupling action evaluation from selection, which utilizes the target network to evaluate the value function with the action obtained from the max operation on the online network [30]. The DDQN's target value y_t calculated by target network $\hat{Q}(\theta^-)$ is formulized as the following, Equation (10).

$$y_t = r_t + \gamma \hat{Q}\left(s_{t+1,}\ \underset{a'}{argmax} Q(s_{t+1}, a'; \theta_t); {\theta_t}^-\right) \tag{10}$$

3.2. Model Architecture

Both DDQNs of the framework use four sheets of the fully connected neural network with 200 nodes and Relu as an activation function. The size of the input sheet of the higher DDQN is set to five, corresponding to five elements of the state feature vector. The size is six in the input sheet of the lower DDQN, in which one additional feature is from the higher DDQN's outcome as a reward form, and the other five ones are from the higher DDQN. The output size of the DLDDQN is seven, each indicating the probability of the symbolized rule being chosen. Two DDQNs take mse as the loss function and Adam of learning rate of 0.000001 as the optimizer.

Figure 1 depicts the DLDDQN architecture of the experimental process. Figure 2 is the realization of the higher DDQN module of the model architecture as Figure 1, and it elaborates on interfaces that communicate with the environment and the lower DDQN. Likewise, Figure 3 realizes the lower DDQN module of the model architecture as Figure 1, and it elaborates interfaces that communicate with the environment and the higher DDQN. Network structures of both agents in Figures 2 and 3 define the input and output vector of the DDQN and the process of gradient descent to update the parameters of the online network with the help of the target network to fix overestimation.

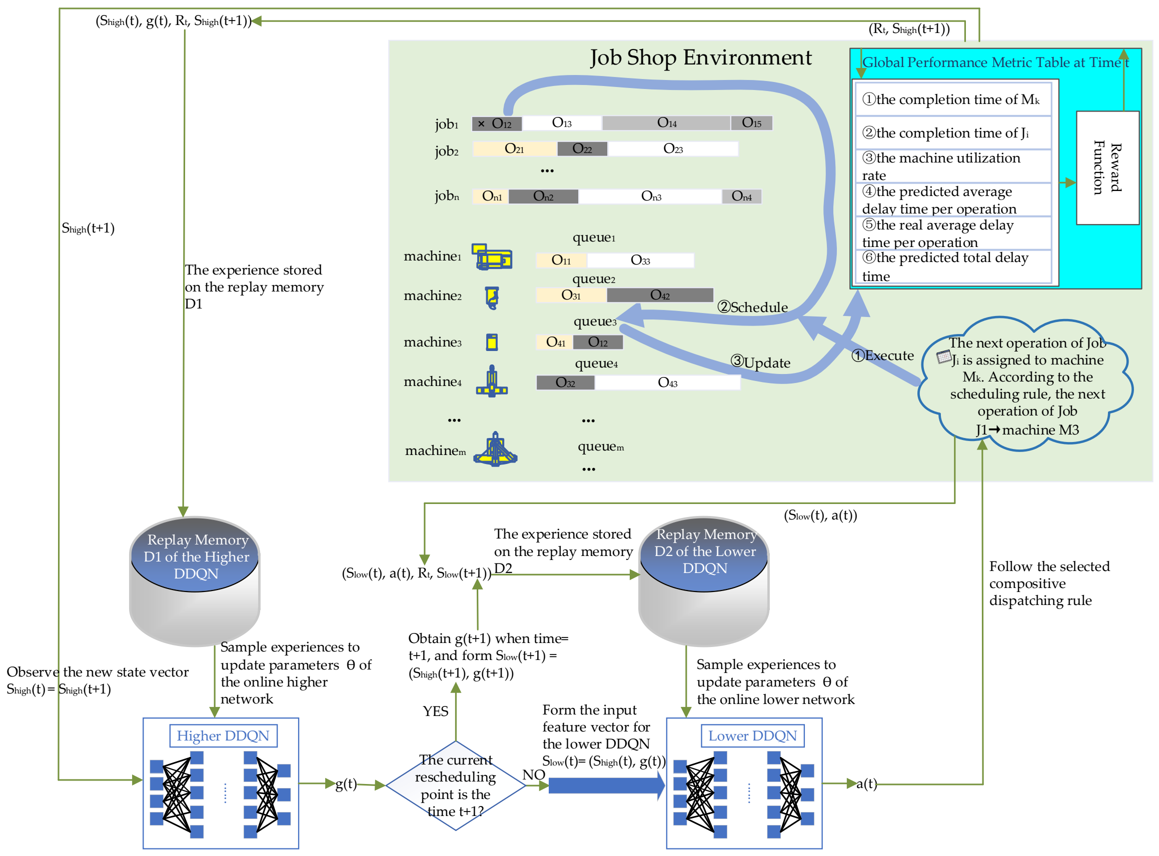

Job Shop Environment
Global Performance Metric Table at Time t
①the completion time of Mk
②the completion time of Ji
③the machine utilization rate
④the predicted average delay time per operation
⑤the real average delay time per operation
⑥the predicted total delay time
Reward Function
②Schedule
③Update
①Execute
The next operation of Job Ji is assigned to machine Mk. According to the scheduling rule, the next operation of Job J1→machine M3
The experience stored on the replay memory D1
Replay Memory D1 of the Higher DDQN
Replay Memory D2 of the Lower DDQN
The experience stored on the replay memory D2
Sample experiences to update parameters θ of the online higher network
Sample experiences to update parameters θ of the online lower network
Observe the new state vector Shigh(t) = Shigh(t+1)
Obtain g(t+1) when time=t+1, and form Slow(t+1) = (Shigh(t+1), g(t+1))
Form the input feature vector for the lower DDQN Slow(t)= (Shigh(t), g(t))
The current rescheduling point is the time t+1?
YES
NO
Higher DDQN
Lower DDQN
Follow the selected compositive dispatching rule

Figure 1. The architecture of the DLDDQN training process on solving the muti-objective DFJSP.

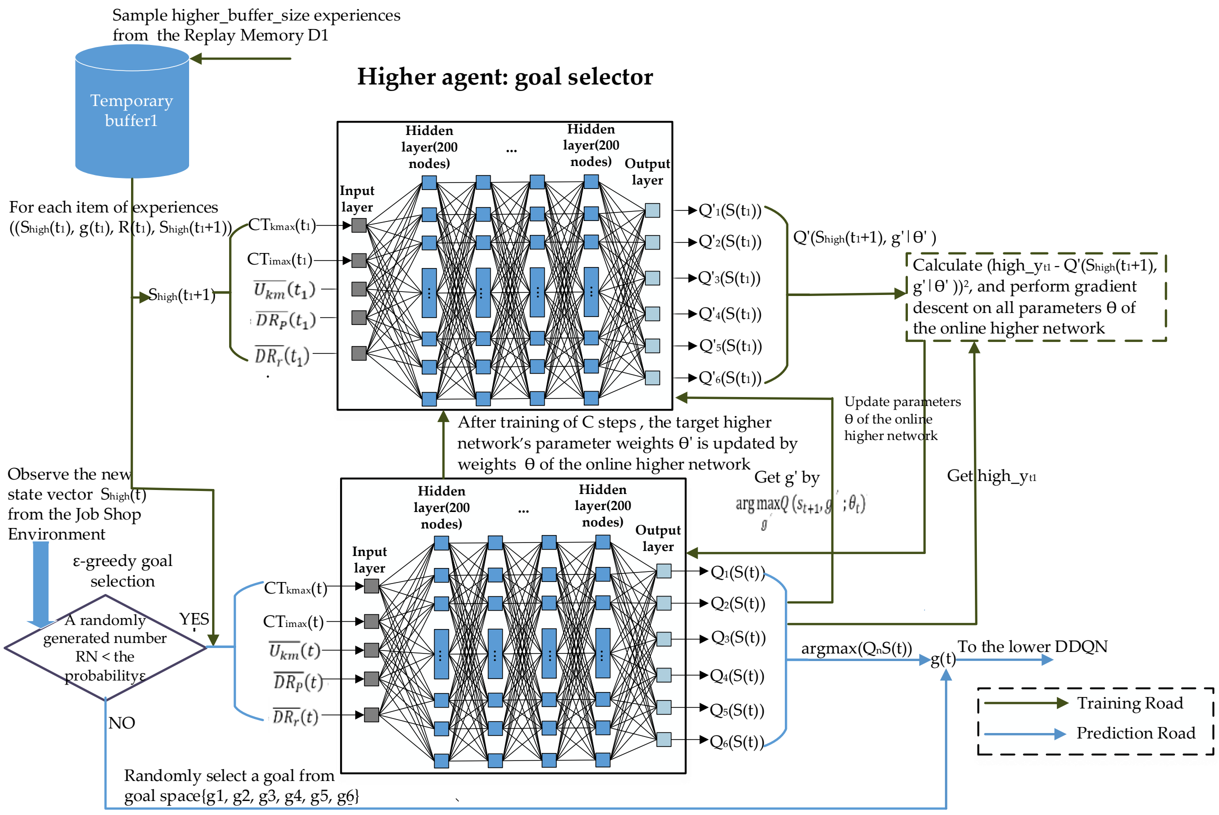

Sample higher_buffer_size experiences from the Replay Memory D1
Temporary buffer1
Higher agent: goal selector
For each item of experiences ((Shigh(t1), g(t1), R(t1), Shigh(t1+1))
Calculate (high_yt1 - Q'(Shigh(t1+1), g'|θ'))², and perform gradient descent on all parameters θ of the online higher network
After training of C steps, the target higher network's parameter weights θ' is updated by weights θ of the online higher network
Update parameters θ of the online higher network
Get g' by
Get high_yt1
Observe the new state vector Shigh(t) from the Job Shop Environment
ε-greedy goal selection
A randomly generated number RN < the probabilityε
YES
NO
Randomly select a goal from goal space{g1, g2, g3, g4, g5, g6}
argmax(QnS(t))
To the lower DDQN
Training Road
Prediction Road

Figure 2. The network structure of the Higher DDQN.

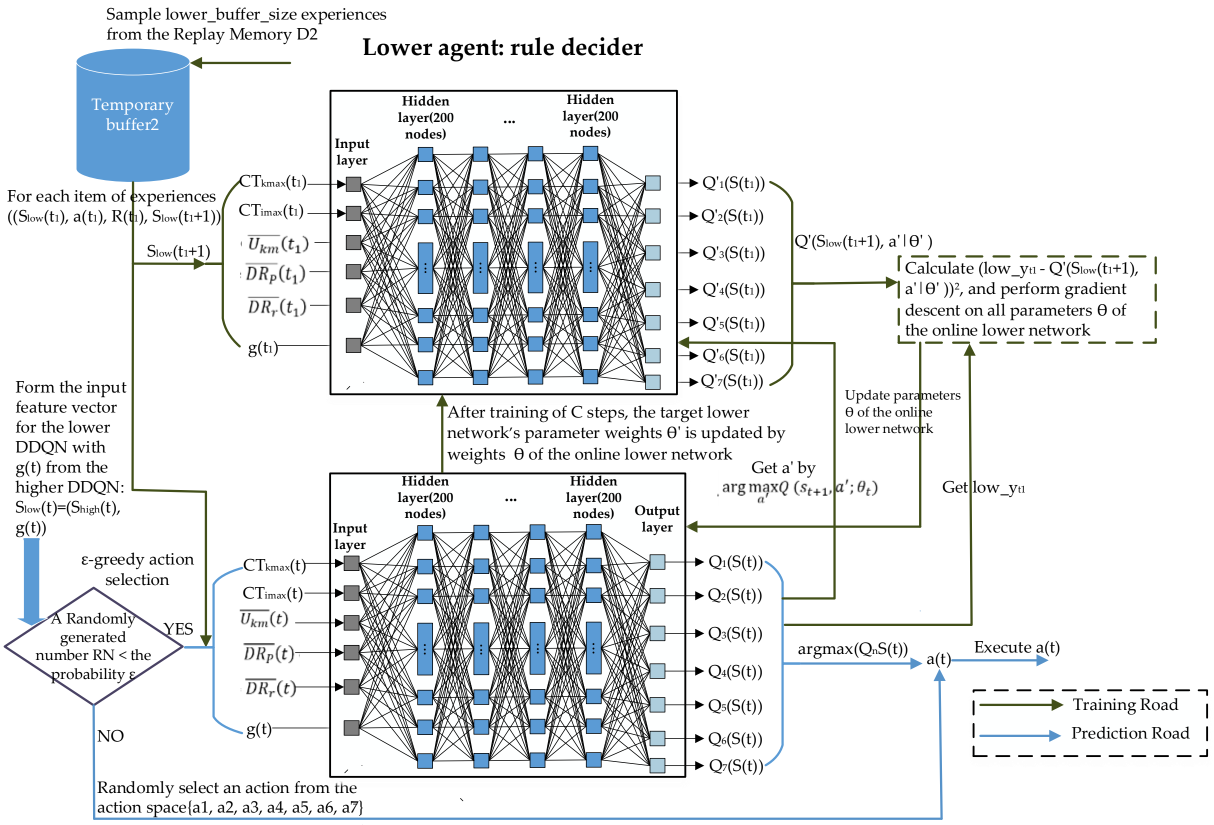

Sample lower_buffer_size experiences from the Replay Memory D2
Lower agent: rule decider
Temporary buffer2
For each item of experiences ((Slow(t1), a(t1), R(t1), Slow(t1+1))
Slow(t1+1)
Hidden layer(200 nodes)
Input layer
Output layer
Q'(Slow(t1+1), a'|θ')
Calculate (low_yt1 - Q'(Slow(t1+1), a'|θ'))², and perform gradient descent on all parameters θ of the online lower network
Update parameters θ of the online lower network
After training of C steps, the target lower network's parameter weights θ' is updated by weights θ of the online lower network
Get a' by arg max Q (s_{t+1}, a'; θ_t)
Get low_yt1
Form the input feature vector for the lower DDQN with g(t) from the higher DDQN: Slow(t)=(Shigh(t), g(t))
ε-greedy action selection
A Randomly generated number RN < the probability ε
YES
NO
argmax(QnS(t))
a(t)
Execute a(t)
Training Road
Prediction Road
Randomly select an action from the action space{a1, a2, a3, a4, a5, a6, a7}

Figure 3. The network structure of the Lower DDQN.

The process of the DLDDQN's cooperation of two agents to train networks for learning the optimal policy is as follows:

(1) A five-element state vector for the input of the higher DDQN is observed from the environment (zero vector when the training is initial) as $S_{high}(t)$

(2) The higher DDQN applies the ε-*greedy* goal selection policy to obtain the goal $g(t)$, as shown in Figure 1.

(3) Together with $S_{high}(t)$, $g(t)$ is included as the input of the lower DDQN to form $S_{low}(t) = (S_{high}(t),\ g(t))$.

(4) The lower DDQN applies the ε-*greedy* action selection policy to obtain the action $a(t)$, as shown in Figure 2.

(5) The environment executes the selected compositive dispatching rule that corresponds to the action $a(t)$ as shown in "① Execute" in Figure 1.

(6) The environment arranges the scheduled operation to its dispatched machine's queue and then updates the job shop situation, as shown in "② Schedule" and "③ Update" in Figure 1, respectively.

(7) The environment feedbacks $(R_t,\ S_{high}(t+1))$ to the higher DDQN. Together with $(S_{high}(t), g(t))$ to form an experience item $(S_{high}(t),\ g(t),\ R_t,\ S_{high}(t+1))$, the higher DDQN stores the item to the Replay Memory D1. Then, repeat procedures (1)–(3) with the input of the higher DDQN as $S_{high}(t+1)$ to obtain $g(t+1)$ and form $S_{low}(t+1) = (S_{high}(t+1),\ g(t+1))$.

(8) Together with $(S_{low}(t),\ a(t))$ to form an experience item $(S_{low}(t),\ a(t),\ R_t,\ S_{low}(t+1))$, the lower DDQN stores the item to the Replay Memory D2.

(9) $S_{low}(t) \leftarrow S_{low}(t+1)$, repeat procedures (4)–(9) until operations of all jobs are completely allocated.

(10) Repeat procedures (1)–(9) until the training epochs end and the trained model converges.

3.3. State Feature Extraction

The state representation should reflect the real-world production environment and satisfy the quantitative and qualitative requirements of the DDQN training. From the quantitative perspective, the number of states should not be excessive or too scarce because too many may increase the probability of the strong correlation between states, while too few cannot outline enough information about the workshop environment to complete ideal scheduling at each rescheduling point t. Thus, the more refined states are, the better the DLDDQN training results can be. From the qualitative perspective, the refined states should not have a drastic value change because it raises the likelihood of training volatility [25]. Generally speaking, the design of the state presentation should observe the following criteria:

- All states should directly or indirectly relate to optimization objectives and the reward function. Other redundant features can be excluded.
- Refined states should be quantitatively and qualitatively advantageous to the DDQN training and the most considerable probable reflection of the global and local scheduling environment.
- State features are numerical representations of the state vector on all dimensions and should be easy to calculate when running on a high-performance CPU or GPU. If necessary, state features are intended to be uniformly normalized to maintain training stability and avoid other issues.

Considering all of the above factors, the following state representations are proposed to characterize the status of the scheduling environment.

Some intervening variables should be specified first. $CT_k(t)$ is the completion time of the last operation assigned on machine M_k at rescheduling point t. $OP_i(t)$ is the number of job J_i's operations assigned with a machine for execution at time t. $CT_i(t)$ is the completion time of the last operation assigned on job J_i at rescheduling point t. $U_{km}(t) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{OP_i(t)} t_{i,j,k}S_{i,j,k}}{CT_k(t)}$ is the utilization rate of machine M_k at rescheduling point t.

(1) The maximum completion time on the last operations of all assigned machines at rescheduling point t, as defined in Equation (11):

$$CT_{kmax}(t) = Max(CT_k(t))\ k = 1,2\ldots m \tag{11}$$

(2) The maximum completion time on the last operations of all assigned jobs at rescheduling point t, as defined in Equation (12):

$$CT_{imax}(t) = Max(CT_i(t))\ i = 1,2\ldots n \tag{12}$$

(3) The average utilization rate of all machines in the system, $\overline{U_{km}}(t)$, can be formulized in Equation (13):

$$\overline{U_{km}}(t) = \sum_{k=1}^{m}\frac{U_{km}(t)}{m} \tag{13}$$

(4) The predicted average delay rate $\overline{DR_P}(t)$ over unfinished jobs in the system.

At rescheduling point t, some of a job's operations have been completed. Therefore, timely prediction by calculating the due-date runtime over the following operations of a job can be critical to deciding whether the global reward should be added or deducted after the environment executes a dispatching rule (or an action). $\overline{DR_P}(t)$ is the ratio of all delayed jobs' total expected processing time since $OP_i(t)$ that $C_{i,j}$ exceeds D_i to the number of timeout operations of all delayed jobs. The procedure to calculate $\overline{DR_P}(t)$ can be written as Algorithm 1.

Algorithm 1: Procedure to calculate the predicted average delay rate $\overline{DR_P}(t)$

```
input: $OP_i(t), D_i$
output: $\overline{DR_P}(t)$
1:$T_{delay}$←0
2:$NO_{uf}$←0
3:for $i = 1 : n$ do
4: if $OP_i(t) < n_i$ then
5:   $T_{irear}$←0
6:   $first$←0
7:     if $C_{i,OP_i(t)} < D_i$
8:       $first$←1
9:     end if
10: for j = $OP_i(t) + 1 : n_i$ do
11:     $T_{irear}$←$T_{irear} + t_{i,j}$
12:       if $C_{i,OP_i(t)} + T_{irear} > D_i$
13:         $NO_{uf}$←$NO_{uf} + 1$
14:           if $first = 1$
15:             $T_{delay}$←$T_{delay} + C_{i,OP_i(t)} + T_{irear} - D_i$
16:             $first$←0
17:         else
18:             $T_{delay}$←$T_{delay} + t_{i,j}$
19:           end if
20:       end if
21:     end for
22: end if
23: end for
24: $\overline{DR_P}(t)$← $\frac{T_{delay}}{NO_{uf}}$
25: return $\overline{DR_P}(t)$
```

(5) The real average delay rate $\overline{DR_r}(t_1)$ over unfinished jobs in the system.

The calculation of the real delay time is based on the last operation's time point $C_{i,OP_i(t)}$ of the current job J_i. If $C_{i,OP_i(t)}$ is greater than D_i, then J_i is considered a real delayed job, in which a delay duration is calculated by the equation: $DD = C_{i,OP_i(t)} - D_i + \sum_{j=OP_i(t)+1}^{n_i} t_{i,j}$. $\overline{DR_r}(t)$ is defined as the ratio of DD of all delayed jobs to the number of delayed operations of all delayed jobs in the system. The procedure to calculate $\overline{DR_r}(t)$ can be written as Algorithm 2.

Algorithm 2: Procedure to calculate the real average delay rate $\overline{DR_r}(t)$

```
Input: $OP_i(t), D_i$
Output: $\overline{DR_r}(t)$
1:$T_{delay}$←0
2:$NO_{uf}$←0
4:for $i = 1 : n$ do
5: if $OP_i(t) < n_i$ then
6:     if $C_{i,OP_i(t)} > D_i$ then
7:         $NO_{uf}$← $NO_{uf} + 1$
8:         $T_{delay}$← $T_{delay} + C_{i,OP_i(t)} - D_i$
9:         for j = $OP_i(t) + 1 : n_i$ do
10:            $NO_{uf}$ ← $NO_{uf} + 1$
11:            $T_{delay}$← $T_{delay} + t_{i,j}$
12:        end for
13:     end if
14: end if
15:end for
16: $\overline{DR_r}(t)$ ← $\frac{T_{delay}}{NO_{uf}}$
17: return $\overline{DR_r}(t)$
```

3.4. Action Space

Actions in the multi-objective DFJSP should be varied and objective-precise in order to adapt to various complicated scheduling situations with which the environment can be fed with the locally optimal dispatching rule to achieve global optimality. For example, in such a scene where there exist jobs with different expired durations and machines with different loads, the better policy decision for the agent is to select the job with the nearest expired duration and distribute the selected job to the machine with the lightest workloads. The action of a policy can be further detailed into two parts: a job selection for the next operation and a machine allocation to complete the selected job's operation. The job selection also can be encountered in two cases: the earlier stage when no delayed jobs exist and the later period when delayed jobs exist partially. In order to shrink the system completion time as small as possible and reduce most of the delay time sum, the seven compositive dispatching rules are developed, given here as Algorithms 3–9.

3.4.1. Compositive Dispatching Rule 1

In this rule, the first task is to find out the set of delayed jobs $Delay_{job}(t)$ and the set of uncompleted jobs $UC_{job}(t)$ at current time t. If there is no existence of delayed jobs, a job selection is converted to $UC_{job}(t)$, and the job with the minimum slack time ($D_i - C_{i,OP_i(t)}$) is selected. Otherwise, a job is selected from $Delay_{job}(t)$, and the job J_i with the largest delay time is selected ($argmax_{i \in Delay_{job}(t)}(C_{i,OP_i(t)} + \sum_{j=OP_i(t)+1}^{n_i} t_{i,j} - D_i)$). The selected job J_i's next operation is allocated to the earliest available machine $M_i = argmin_{k \in M_{i,j}}\{CT_k(t),\ k = 1,\ 2,\ \ldots,\ m\}$. Note that the start runtime of job J_i's next operation on machine M_i is not always $CT_{Mi}(t)$, and it should be $max\left\{\ CT_{Mi}(t),\ C_{J_i,j-1},\ A_{J_i}\right\}$.

Algorithm 3: Pseudo code of compositive dispatching rule 1

```
1:  $Delay_{job}(t) \leftarrow \left\{i|OP_i(t) < n_i \&\& C_{i,OP_i(t)} > D_i\right\}$
2:  $UC_{job}(t) \leftarrow \{i|OP_i(t) < n_i\}$
3:  if Count($Delay_{job}(t)$) = 0 then
4:    $J_i \leftarrow argmax_{i \in UC_{job}(t)}(D_i - C_{i,OP_i(t)})$
5:  else
6:    $J_i \leftarrow argmax_{i \in Delay_{job}(t)}(C_{i,OP_i(t)} + \sum_{j=OP_i(t)+1}^{n_i} t_{i,j} - D_i)$
7:  end if
8:    $j \leftarrow OP_{J_i}(t) + 1$
9:    $M_i \leftarrow argmin_{k \in M_{J_i,j}}\{CT_k(t)\}$
10: $ST_{J_ij} \leftarrow max\left\{\ CT_{Mi}(t),\ C_{J_i,j-1},\ A_{J_i}\right\}$
11: arrange $O_{J_ij}$ of job $J_i$ on $M_i$ to execute a task
```

3.4.2. Compositive Dispatching Rule 2

In the second dispatching rule, $Delay_{job}(t)$ and $UC_{job}(t)$ at the current rescheduling point t should be initialized first. Supposing $Delay_{job}(t)$ is empty, the strategy to select a job is the smallest sum of the remaining processing time of all its uncomplete operations from $UC_{job}(t)$. Otherwise, the top priority should be given to $Delay_{job}(t)$, and the job J_i with the largest delay time is selected ($argmax_{i \in Delay_{job}(t)}(C_{i,OP_i(t)} + \sum_{j=OP_i(t)+1}^{n_i} t_{i,j} - D_i)$). The next operation of the selected job J_i is $OP_{J_i}(t) + 1$. The selected job's next operation can be allocated to every available machine with its unique processing time, but the machine with the smallest product of $t_{J_i,j,k} \times CT_k(t)$ is selected.

Algorithm 4: Pseudo code of compositive dispatching rule 2

```
1:  $Delay_{job}(t) \leftarrow \left\{ i|OP_i(t) < n_i \&\& C_{i,OP_i(t)} > D_i \right\}$
2:  $UC_{job}(t) \leftarrow \{i|OP_i(t) < n_i\}$
3:  if Count($Delay_{job}(t)$) = 0 then
4:  $J_i \leftarrow argmin_{i \in UC_{job}(t)}(\sum_{j=OP_i(t)+1}^{n_i} t_{i,j})$
5:  else
6:  $J_i \leftarrow argmax_{i \in Delay_{job}(t)}(C_{i,OP_i(t)} + \sum_{j=OP_i(t)+1}^{n_i} t_{i,j} - D_i)$
7:  end if
8:  $j \leftarrow OP_{J_i}(t) + 1$
9:  $M_i \leftarrow argmin_{k \in M_{J_i,j}} \left\{ t_{J_i,j,k} \times CT_k(t) \right\}$
10: $ST_{J_ij} \leftarrow max\left\{ CT_{Mi}(t),\ C_{J_i,j-1},\ A_{J_i} \right\}$
11: arrange $O_{J_ij}$ of job $J_i$ on $M_i$ to execute a task
```

3.4.3. Compositive Dispatching Rule 3

In the third dispatching rule, if $Delay_{job}(t)$ is empty, select the job with the shortest predicted average processing time of the imminent operation, calculated based on available machines of the operation. Otherwise, the job with the longest overdue time should be prioritized from $Delay_{job}(t)$. The selected job's next operation is allocated to the machine with the lowest workload.

Algorithm 5: Pseudo code of compositive dispatching rule 3

```
1:  $Delay_{job}(t) \leftarrow \left\{ i|OP_i(t) < n_i \&\& C_{i,OP_i(t)} > D_i \right\}$
2:  $UC_{job}(t) \leftarrow \{i|OP_i(t) < n_i\}$
3:  if Count($Delay_{job}(t)$) = 0 then
4:  $J_i \leftarrow argmin_{i \in UC_{job}(t)}(\frac{\sum_{k=1}^{m} t_{i,OP_i(t)+1,k} \times S_{i,j,k}}{\sum_{k=1}^{m} S_{i,j,k}})$
5:  else
6:  $J_i \leftarrow argmax_{i \in Delay_{job}(t)}(C_{i,OP_i(t)} + \sum_{j=OP_i(t)+1}^{n_i} t_{i,j} - D_i)$
7:  end if
8:  $j \leftarrow OP_{J_i}(t) + 1$
9:  $M_i = argmin_{k \in M_{J_i,j-1}} \left\{ \sum_{i=1}^{n} \sum_{j=1}^{OP_i(t)} t_{i,j,k} S_{i,j,k} \right\}$
10: $ST_{J_ij} \leftarrow \max\left\{ CT_{Mi}(t), C_{J_i,j-1}, A_{J_i} \right\}$
11: arrange $O_{J_ij}$ of job $J_i$ on $M_i$ to execute a task
```

3.4.4. Compositive Dispatching Rule 4

In the fourth dispatching rule, after all front $OP_i(t)$ operations are completed, among jobs that are predicted to delay, calculate their predicted delay sum ($C_{i,OP_i(t)} + \sum_{j=OP_i(t)+1}^{n_i} t_{i,j} - D_i$) and the job with the largest value is selected, and jobs that do not delay by prediction are ignored. If no jobs are predicted to delay, the job with the smallest predicted due distance ($D_i - C_{i,OP_i(t)} - \sum_{j=OP_i(t)+1}^{n_i} t_{i,j}$) is selected. The next operation of the selected job J_i, j $= (OP_{J_i}(t) + 1)$ is allocated to the machine with the smallest product of $t_{J_i,j,k} \times CT_k(t)$.

Algorithm 6: Pseudo code of compositive dispatching rule 4

```
1:  $UC_{job}(t) \leftarrow \{i|OP_i(t) < n_i\}$
2:  $J_i \leftarrow argmax_{i \in UC_{job}(t)}(C_{i,OP_i(t)} + \sum_{j=OP_i(t)+1}^{n_i} t_{i,j} - D_i)$
3:  $j \leftarrow OP_{J_i}(t) + 1$
4:  $M_i \leftarrow argmin_{k \in M_{J_i,j}} \left\{ t_{J_i,j,k} \times CT_k(t) \right\}$
5:  $ST_{J_ij} \leftarrow \max\left\{ CT_{Mi}(t), C_{J_i,j-1}, A_{J_i} \right\}$
6:  arrange $O_{J_ij}$ of job $J_i$ on to execute a task
```

3.4.5. Compositive Dispatching Rule 5

In the fifth dispatching rule, the operation completion rate of job J_i should be considered. If no delayed jobs exist, the job with a low operation completion rate and short slack time should be selected. Otherwise, the job with a low operation completion rate and large delay time from $Delay_{job}(t)$ should be selected. The operation $j = (OP_{J_i}(t) + 1)$ of selected job J_i is allocated to the machine with the smallest product of $t_{J_i,j,k} \times CT_k(t)$.

Algorithm 7: Pseudo code of compositive dispatching rule 5

```
1:  $Delay_{job}(t) \leftarrow \left\{ i | OP_i(t) < n_i \&\& C_{i,OP_i(t)} > D_i \right\}$
2:  $UC_{job}(t) \leftarrow \{ i | OP_i(t) < n_i \}$
3:  if Count($Delay_{job}(t)$) = 0 then
4:  $J_i \leftarrow argmin_{i \in UC_{job}(t)}((D_i - C_{i,OP_i(t)}) \times \frac{OP_i(t)}{n_i})$
5:  else
6:  $J_i \leftarrow argmax_{i \in Delay_{job}(t)}((C_{i,OP_i(t)} + \sum_{j=OP_i(t)+1}^{n_i} t_{i,j} - D_i) \times \frac{n_i}{OP_i(t)})$
7:  end if
8:  $j \leftarrow OP_{J_i}(t) + 1$
9:  $M_i \leftarrow argmin_{k \in M_{J_i,j}} \left\{ t_{J_i,j,k} \times CT_k(t) \right\}$
10: $ST_{J_ij} \leftarrow \max\left\{ CT_{Mi}(t), C_{J_i,j-1}, A_{J_i} \right\}$
11: arrange $O_{J_ij}$ of job $J_i$ on $M_i$ to execute a task
```

3.4.6. Compositive Dispatching Rule 6

The sixth dispatching rule considers the critical ratio of the slack time to the remaining processing time. If no delayed jobs exist, a smaller cr is expected. Otherwise, the delayed job J_i with a larger delay time and a shorter processing time of remaining operations is selected. The next operation of the selected job J_i, $j = \left(OP_{J_i}(t) + 1 \right)$, is allocated to the machine with the smallest processing time $argmin\ _{k \in M_{J_i, OP_{J_i}(t)+1}} \left\{ t_{J_i, OP_{J_i}(t)+1,k} \right\}$.

Algorithm 8: Pseudo code of compositive dispatching rule 6

```
1: $UC_{job}(t) \leftarrow \{ i | OP_i(t) < n_i \}$
2: $J_i \leftarrow argmin_{i \in UC_{job}(t)}(\frac{(D_i - C_{i,OP_i(t)})}{\sum_{j=OP_i(t)+1}^{n_i} t_{i,j}})$
3: $j \leftarrow OP_{J_i}(t) + 1$
4: $M_i \leftarrow argmin\ _{k \in M_{J_i,j}} \left\{ t_{J_i,j,k} \right\}$
5: $ST_{J_ij} \leftarrow \max\left\{ CT_{Mi}(t), C_{J_i,j-1}, A_{J_i} \right\}$
6: arrange $O_{J_ij}$ of job $J_i$ on $M_i$ to execute a task
```

3.4.7. Compositive Dispatching Rule 7

In the last dispatching rule, if $Delay_{job}(t)$ is empty, select the job with the earliest due date. Otherwise, select the job with the largest expected delay time from $Delay_{job}(t)$. The next operation of the selected job J_i is allocated to the machine earliest to finish the operation $(OP_{J_i}(t) + 1)$.

Algorithm 9: Pseudo code of compositive dispatching rule 7

```
1:  $Delay_{job}(t) \leftarrow \left\{ i | OP_i(t) < n_i \&\& C_{i,OP_i(t)} > D_i \right\}$
2:  $UC_{job}(t) \leftarrow \{ i | OP_i(t) < n_i \}$
3:  if Count($Delay_{job}(t)$) = 0 then
4:  $J_i \leftarrow argmin_{i \in UC_{job}(t)}(D_i)$
5:  else
6:  $J_i \leftarrow argmax_{i \in Delay_{job}(t)}(C_{i,OP_i(t)} + \sum_{j=OP_i(t)+1}^{n_i} t_{i,j} - D_i)$
7:  end if
8:  $j \leftarrow OP_{J_i}(t) + 1$
9:  $M_i \leftarrow argmin_{k \in M_{J_i,j}} \left\{ max\left\{ CT_k(t),\ C_{J_i,j-1},\ A_{J_i} \right\} + t_{J_i,j,k} \right\}$
10: $ST_{J_i j} \leftarrow \max\left\{ CT_{Mi}(t), C_{J_i,j-1}, A_{J_i} \right\}$
11: arrange $O_{J_i j}$ of $J_i$ on $M_i$ to execute a task
```

3.5. Goal Formations and Reward Functions

Six different higher goals are developed to serve as directions for the DDQN to optimize two objectives. Among the six higher goals, five of which sources are the state presentations, while the last one is a brand-new notation $PTDT(t)$ (predicted total delay time) as defined in Algorithm 10. It is known that the more detailed and specific goals are to describe the two optimization objectives, the more possible to direct the DDQN to orientate objectives for optimization from various aspects.

Due to the impossibility of two-objective parallel optimization at a particular training iteration, a higher DDQN is needed to decide on a preferred higher goal that corresponds to one optimization objective (makespan or total delay time), and a lower DDQN is required to decide on the preferred dispatching rule that suits the current scheduling situation with the selected goal and other five states as input. Through the cooperation between the two DDQNs, different goals adaptive to the current environment can take turns to direct the two DDQNs to train with more rewards. To accumulate maximum rewards from the current state S_t to the next state S_{t+1}, ways to evaluate whether the current chosen action is effective with the guidance of the goal selected from the higher DDQN are shown in Table 2.

Algorithm 10: Procedure to calculate the predicted total delay time $PTDT(t)$

```
input: $OP_i(t), D_i,\ C_{i,OP_i(t)}$
output: $PTDT(t)$
1:$PTDT(t) \leftarrow 0$
2:for $i = 1 : n$ do
3: if $OP_i(t) < n_i$ then
4:     if $C_{i,OP_i(t)} + \sum_{j=OP_i(t)+1}^{n_i} \frac{\sum_{k=1}^{m} S_{i,j,k} t_{i,j,k}}{m} - D_i > 0$ then
5:         $PTDT(t) \leftarrow PTDT(t) + C_{i,OP_i(t)} + \sum_{j=OP_i(t)+1}^{n_i} \frac{\sum_{k=1}^{m} S_{i,j,k} t_{i,j,k}}{m} - D_i$
6:     end if
7: end if
8:end for
9:return $PTDT(t)$
```

Table 2. Six ways to increase cumulative reward guided by a goal under its one special objective.

Number	Goal	Objective Property	Way of Cumulative Reward
1	$CT_{kmax}(t)$	makespan	$CT_{kmax}(t+1) \leq 1.1 \times CT_{kmax}(t)$
2	$CT_{imax}(t)$	makespan	$CT_{imax}(t+1) \leq 1.1 \times CT_{imax}(t)$
3	$\overline{U_{km}}(t)$	makespan	$\overline{U_{km}}(t+1) > \overline{U_{km}}(t)$
4	$\overline{DR_P}(t)$	total delay time	$\overline{DR_P}(t+1) < \overline{DR_P}(t)$
5	$\overline{DR_r}(t)$	total delay time	$\overline{DR_r}(t+1) < \overline{DR_r}(t)$
6	$PTDT(t)$	total delay time	$\mathrm{PTDT(t+1) < PTDT(t)}$

Six higher goals {1, 2, 3, 4, 5, 6} towards the two optimization objectives are set as the goal space to calculate the reward, as presented in Table 2. An increment or decrement of the current reward at rescheduling point t is through a comparison of values of the selected goal at scheduling time t and scheduling time $t+1$ after the environment follows the chosen action that has the maximum Q value in the lower DDQN and executes the dispatching rule. The reward implementation of Table 2 is shown in Algorithm 11.

Algorithm 11: Procedure to calculate the reward with the specified goal of higher DDQN in the DLDDQN architecture

```
Input: The chosen goal's value at rescheduling point $t$ and $t+1$, the total reward sum $TRS(t)$
Output: The total rewards after the action is executed at the current rescheduling point $t$
1:if $goal_t = 1$ then
2:     if $CT_{kmax}(t+1) \leq 1.1 \times CT_{kmax}(t)$ then
3:            $TRS(t) \leftarrow TRS(t) + 1$
4:    else if $1.1 \times CT_{kmax}(t) < CT_{kmax}(t+1) \leq 1.5 \times CT_{kmax}(t)$ then
5:            $TRS(t) \leftarrow TRS(t) + 0$
6:    else
7:            $TRS(t) \leftarrow TRS(t) - 1$
8:    end if
9: else if $goal_t = 2$ then
10:    if $CT_{imax}(t+1) \leq CT_{imax}(t)$ then
11:        $TRS(t) \leftarrow TRS(t) + 1$
12:    else if $1.1 \times CT_{imax}(t) < CT_{imax}(t+1) \leq 1.5 \times CT_{imax}(t)$ then
13:        $TRS(t) \leftarrow TRS(t) + 0$
14:    else
15:        $TRS(t) \leftarrow TRS(t) - 1$
16:    end if
17:else if $goal_t = 3$ then
18:    if $\overline{U_{km}}(t+1) > \overline{U_{km}}(t)$ then
19:        $TRS(t) \leftarrow TRS(t) + 1$
20:     else if $\overline{U_{km}}(t+1) < \overline{U_{km}}(t)$ then
21:        $TRS(t) \leftarrow TRS(t) - 1$
22:else if $goal_t = 4$ then
24:     if $\overline{DR_P}(t+1) > \overline{DR_P}(t)$ then
25:        $TRS(t) \leftarrow TRS(t) - 1$
26:    else if $\overline{DR_P}(t+1) < \overline{DR_P}(t)$then
27:        $TRS(t) \leftarrow TRS(t) + 1$
28:    end if
29:else if $goal_t = 5$ then
30:     if $\overline{DR_r}(t+1) > \overline{DR_r}(t)$ then
31:            $TRS(t) \leftarrow TRS(t) - 1$
32:    else if $\overline{DR_r}(t+1) < \overline{DR_r}(t)$ then
33:            $TRS(t) \leftarrow TRS(t) + 1$
34:    end if
35:else if $goal_t = 6$ then
36:     if $PTDT(t+1) > PTDT(t)$ then
37:            $TRS(t) \leftarrow TRS(t) - 1$
38:    else if $PTDT(t+1) < PTDT(t)$ then
39:            $TRS(t) \leftarrow TRS(t) + 1$
40:    end if
41:end if
42:return $TRS(t)$
```

4. Numerical Experiments

In this section, the first introduces the generation of random benchmark examples, which specifies the time consumption on each available machine for operations of each job. The second is the parameter tuning of the DLDDQN to perform the best result of two optimization objectives. The third is a case study on one of the training iterations, and the fourth is a trained model selection and the validation of the proposed framework. The fifth is an experimental result comparison between the DLDDQN and proposed compositive rules, and the last is a generality analysis of the DLDDQN with an experimental result comparison among DLDDQN, other two DRL-based methods, heuristic rules, and a metaheuristic algorithm (genetic algorithm).

4.1. Instance Generations

It is assumed that the number of initial jobs is already known beforehand and equal to a particular value. The following jobs come to the job shop obeying a Poisson distribution: an arriving interval between two jobs accords with the exponential distribution $-\lambda e^{-1/\lambda}$, in which the parameter λ (> 0) corresponds to the average rate E_ave. The arrival time of initial jobs is set to 0. The number of operations of each job, n_i, is determined by a random value of [1–20]. The value of integer k is generated within [1–$(m-2)$], where m is the total number of all machines. The available machine set $m_{i,j}$ for an operation of each job $O_{i,j}$ is built with $k+2$ randomly chosen machine indexes, in each of which the program generates the processing time $t_{i,j,k}$. The due time of each job D_i equals the sum of the arrival time A_i, the product of the expected average processing time PT_i, and a factor DDT.

The higher buffer size for storing experiences of the higher DDQN differs from the lower buffer due to the update problem of the two networks [39]. Consistency between the two-layer networks should be maintained, which requests to avoid a circumstance where different transitions of the lower network have the same goal generated from the higher network. Therefore, it is a must to control the higher layer to adapt to the changing of the lower layer. The efficient way is to set the higher layer's buffer size to the same amount as the training sample size to discard timeout experiences.

The proposed framework's training, testing, and validating are implemented in the remote Linux ubuntu18.04 server platform with a configuration of 12 vCPU Intel(R) Xeon(R) Platinum 8255C CPU @ 2.50 GHz, RTX 3080 (10 GB), RAM 40 GB, and tensorflow1.15. Benchmark instances are generated according to the parameter setting of Table 3, in which randxxx symbolizes that the instance generation randomly picks one of the values within the given range. The DRL framework aims to learn an optimal policy that accumulates the maximum long-term rewards during training epochs. Intelligent agents should have the ability of generalization to adapt to the raw input and output near-optimal optimization objectives [40]. The generalization is achieved through a new instance put into the model training since an epoch restarts.

Table 3. The value range of parameters for the generation of different instances.

Number	Parameter	Value
1	Total number of machines (m)	randint [10–30]
2	Arrival interval of new coming jobs λ (E_ave)	randint [50–200]
3	Number of operations for each job (n_i)	randint [1–20]
4	Variable for available machines of each operation (k)	randint [1–$(m-2)$]
5	Processing time of $O_{i,j}$ in machine k ($t_{i,j,k}$)	randint [1–50]
6	Due date tightness (DDT)	randfloat [0.5–2]
7	Number of initial jobs (n_ij)	randint [10–20]
8	Number of new arrival jobs (n_naj)	randint [10–200]

4.2. Sensitivity of Hyperparameters That Influence Performance of the Proposed Framework DLDDQN

The training performance of Deep Q-network can be affected by different value combinations of hyperparameters. A possible way to configure the DLDDQN to maximize metric performances is to search the value space of all hyperparameters for the best one or some. Nevertheless, the method is infeasible because the search space of one hyperparameter is vast, let alone a large number of hyperparameters, which causes intense computational cost and training time for comparisons. By random search, enough limited trials, and verifications [39], we found the approximately optimal value of hyperparameters for the proposed framework. Production examples instantiated as Table 4 are used to test the performance sensitivity of optimization objectives under different hyperparameter settings.

Table 4. Parameter specification of instance generations.

Number	Parameter	Value
1	Total number of machines (m)	10
2	Number of initial jobs (n_ij)	5
3	Number of new arrival jobs (n_naj)	10
4	Arrival interval of new jobs λ (E_ave)	50
5	Due date tightness (DDT)	0.5
6	Number of operations for each job (n_i)	randint (1–5)
7	Variable for available machines of each operation (k)	randint (1–8)
8	Processing time in selected ($k+2$) available machines	randint (1–50)
9	Arrival time of new coming jobs (A_i)	exponential (50)
10	Due date of jobs (D_i)	$Ai + \sum_{j=1}^{n_i} t_{i,j} \times 0.5$

Unlike JSP benchmark examples in which operations are specified on a fixed machine, the generated ones with parameters initialized by Table 4 enable an operation of job J_i to be processed by all available machines in the production shop.

Using numerical examples of benchmark tests from Table 4, we can find hyperparameters' superior combinatorial value for the proposed DLDDQN model by observing and analyzing obtained convergent training curves.

From Figure 4, we can find that the bigger the batch size of the model, the faster the training converges. Meanwhile, it should be noted that a much larger batch size means a slower training speed due to larger sampled experiences to be learned by the model and more time spent on each step. Therefore, considering the training time and the faster convergence speed, the advisable batch size setting is 64 units.

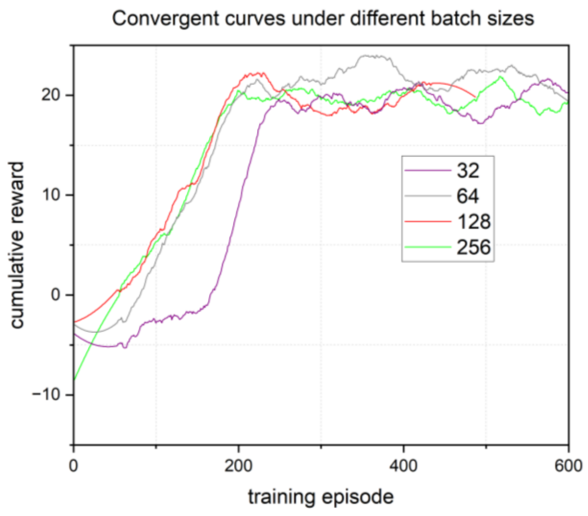


Figure 4. Convergence comparison of different batch sizes.

With analogous comparative experiments, the convergent speed, stability, and the possibly achieved lowest convergent value of objectives are analyzed through experimental result comparisons on different value settings of hyperparameters. One of the most likely best hyperparameter combinations is found, which is used in all of the following comparative experiments to train the desired model: (1) discount factor $\gamma = 0.9$; (2) frequency of updating the target network on two DDQNs = 20 steps; (3) initial probability of exploration in the action selection policy ε-greedy = 0.95; (4) decrement each episode on the probability of exploration in action selection policy ε-greedy = 0.00001; (5) buffer size of higher DDQN and lower DDQN is 64 and 1000, respectively; (6) batch size to sample experiences = 64; and (7) training episodes = 7000.

4.3. Analysis of a Case Study

Our multi-objective optimization research is to achieve a compromise between the job's delay time and the machine's makespan and to minimize the two objectives simultaneously with two DDQNs as function approximators. The training process contains thousands of steps to converge on a near-optimal solution. Thus, to clarify the training process of the proposed scheduling agents, an illustration of the episode's training at an intermediate stage of the entire training loop is exemplified in the benchmark numerical examples of Table 4.

It can be found from Table 4 that numerical examples need 10 machines to complete 15 jobs, each of which has a different number of operations, aggregated at least 15 operations. According to the settings of the 9th and 10th items in Table 4, the arrival time and due date time in one of generated numerical examples can be seen in Tables 5 and 6.

Table 5. The arrival time of initial jobs and new coming jobs.

Job	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
arrival time	0	0	0	0	0	22	11	5	58	32	1	38	31	98	178

Table 6. The due time of initial jobs and new coming jobs.

Job	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
due time	56	68	33	57	65	60	51	44	126	53	12	83	71	103	247

A complete rule dispatching starts at rescheduling point t, when a five-element state feature vector $S_{high}(t) = (CT_{kmax}(t), CT_{imax}(t), \overline{U_{km}}(t), \overline{DR_P}(t), \overline{DR_r}(t))$ is updated from the execution of the last dispatched rule. Observing the current production environment after several operations of each job have been assigned on machines, it can be found that $S_{high}(t) =$ [4.604725, 3.9096076, 5.646244, 5.6403065, 4.417972, 5.2875075]. According to the goal selection strategy, the parameter *high_ε_greedy* will gradually decrease to 0.1 with the increment of assigned operations. If a randomly generated number is smaller than the current *high_ε_greedy*, a random goal will be generated to satisfy the exploration requirement of the higher DDQN at the early stage of training; otherwise, it can turn to the trained higher DDQN for Q value prediction of goals and the goal with the maximum Q value will be selected. In the case where the randomly generated value is greater than the compared parameter *high_ε_greedy* at the later training stage, the Q value vector obtained from the higher online network can be presented as [4.604725, 3.9096076, 5.646244, 5.6403065, 4.417972, 5.2875075] and *goal* 2 with the maximum Q value of 5.646244 is selected. After the specific reward form and the corresponding optimization objective are confirmed, the lower DDQN's input state vector $S_{low}(t) = (CT_{kmax}(t), CT_{imax}(t), \overline{U_{km}}(t), \overline{DR_P}(t), \overline{DR_r}(t), goal)$ can be assigned the value vector [4.604725, 3.9096076, 5.646244, 5.6403065, 4.417972, 5.2875075, 2]. According to the action selection policy, a bigger value of *low_ε_greedy* at the early training stage will cause the selection of a random action (0–6) for exploration in the lower DDQN. In the later training stage, the action is determined by the lower online network from the predicted Q value vector [4.9402447, 5.679509, 4.9360456, 7.033271, 5.8471146, 7.110085, 5.3056755] and the policy 5 with the maximum Q value of 7.110085 is selected. Following the selected rule 6, the uncomplete job with the smallest ratio of slack time to the expected time sum of successive operations is selected, and job 12 is the eligible one; the selected machine is the one on which the imminent operation of the selected job has the shortest processing time, and machine 6 is the wanted one. Therefore, the subsequent operation of job 12 is arranged for machine 6.

After 7000 training iterations of the all-operation allocation, the convergent curve of total rewards can be depicted in Figure 5.

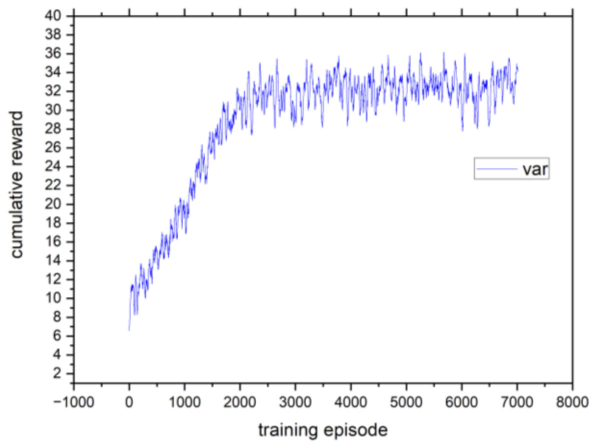


Figure 5. Convergent training curve of total rewards.

4.4. Model Selection and Validation of the Proposed DLDDQN

The DLDDQN is trained under problem instances in Table 4. Training curves of two objectives in Figure 6 illustrate that the training process changes to become steady and flat within a small-variation range since 3100 steps and has no clues for decreasing to a smaller objective value. In the later stage of training, objective curves do not converge to an exact value for the sake of the action selection strategy and the neural network mechanism: there still exists a low probability of exploration for learning even approaching the training end; the framework of two neural networks is an approximator of the reward function with stochasticity, which means that being in different environments with a distinct state feature vector, the agents can only output the objective value nearby an optimal solution.

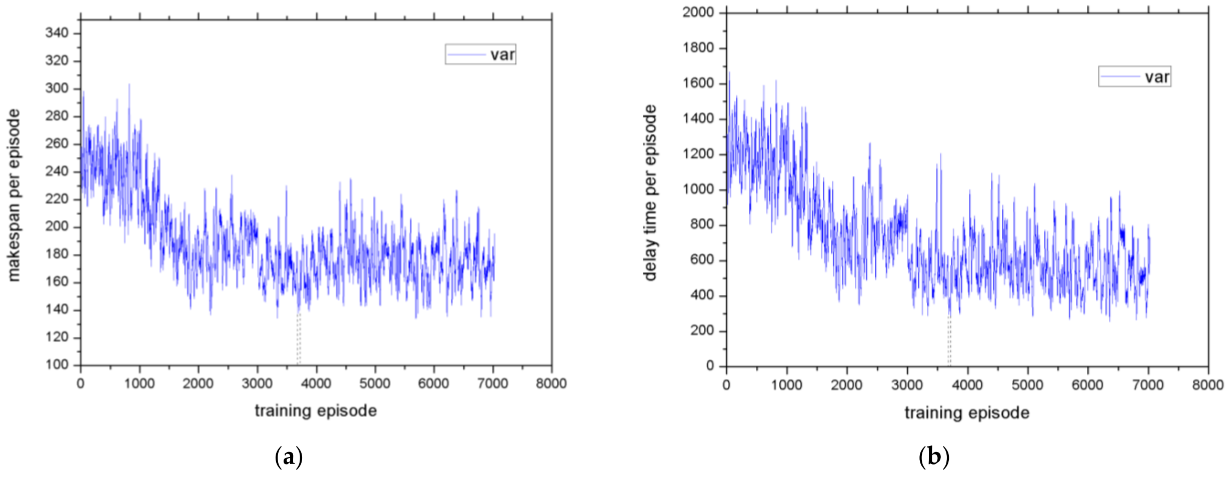
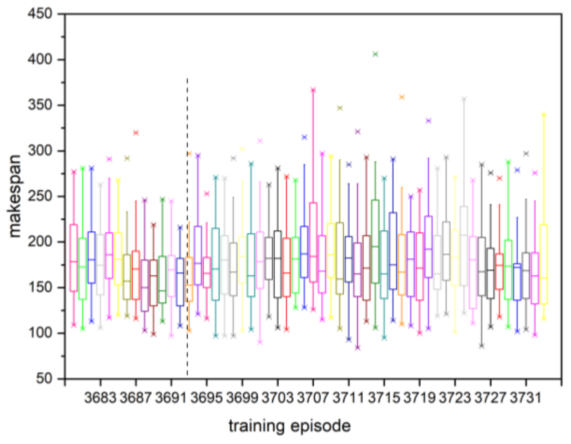


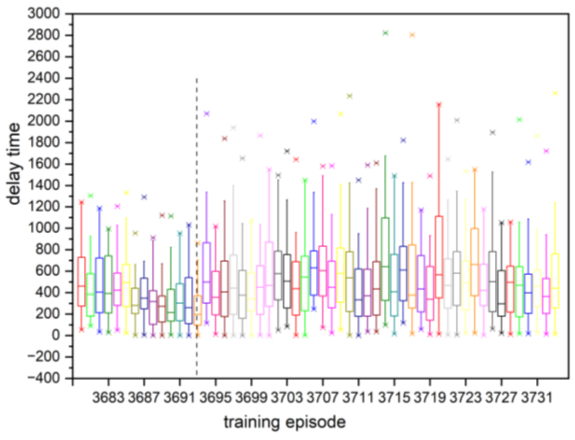
Figure 6. Training curve of the two optimization objectives: models from 3680 to 3733 are selected for validation: (**a**) makespan; (**b**) delay time sum.

After each training step, the trained higher and lower online network are saved for validation. The distribution scope and the lowest objective level in box plots should be as small as possible in order to obtain a perfect convergent model with good generalization. Therefore, 54 models from the 3680th to the 3733rd episode (the part between two black dot lines in Figure 6a,b) are selected for validation on generated 30 problem instances from the initialization of Table 4. It can be seen from Figure 7a,b that the model on episode

3693 has a shorter length in the box of a rectangle shape and smaller average objective values, which indicates that the model can learn effective rules to adapt to a new job shop environment at any time step t and obtain relatively more stable and smaller objective values with validation of 30 instances on it. After that, the model trained in episode 3693 is selected as the final standard DLDDQN model for the experimental tests of Section 4.5.



(**a**)

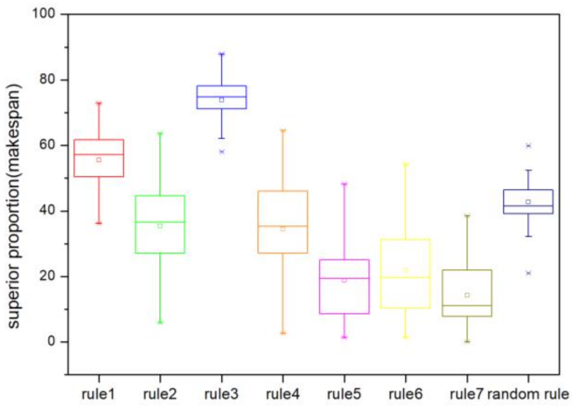


(**b**)

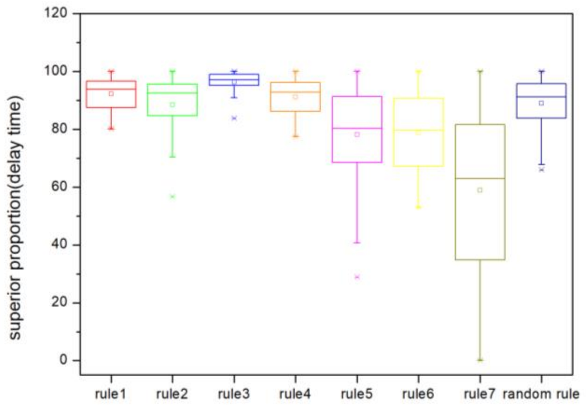
Figure 7. Performance of 54 selected models on two objectives tested from generated 30 problem instances. (**a**) makespan (**b**) delay time sum.

4.5. Comparison with Proposed Compositive Dispatching Rules

In order to verify that the proposed framework has better improvements on objectives of makespan and delay time, seven proposed compositive dispatching rules and a random rule, i.e., arbitrary selection of a designed compositive dispatching rule at each rescheduling point, are compared with the trained DLDDQN model on tested objective results. A total of 30 test instances are generated from production configuration set as $m = 10$, $n\_naj = 10$, $E\_ave = 50$, $DDT = 0.5$. The random rule is run on benchmark instances 20 times, and the mean value is obtained to compare results from other rules run only once. The superior proportion is introduced as $\frac{y-x}{y}$, where y has the inferior performance with a larger objective value, while x is the opposite [41]. Figure 8a,b show the superior proportion (each compositive rule to the trained DLDDQN model) on objectives in box plots.



(**a**)



(**b**)

Figure 8. The superior proportion of the DLDDQN's performance to other proposed compositive dispatching rules on two objectives tested on generated 30 problem instances: (**a**) makespan; (**b**) delay time sum.

In Figure 8a, the compositive dispatching rule 2 and 4–7 have the lowest superior proportion boundary near to 0, which illustrates that these compositive rules are well designed to optimize makespan, enabling the objective value to shrink to the virtually identical low as the proposed DLDDQN in some instances. In contrast, rules 1 and 3 and the random rule have a relatively higher superior proportion boundary, which reveals that these applied to the DLDDQN's action space can gain a better result. In Figure 8b, compositive dispatching rules and the random rule all have the highest superior proportion boundary, which tells that proposed compositive rules applied to the action space earn a better result. The comparison with compositive dispatching rules shows that the trained DLDDQN model outperforms any handmade dispatching rules orientated to objective optimization and proves the effectiveness and superiority of the proposed framework to optimize objectives.

4.6. Comparison with Other Methods

To expand the generalization of the proposed framework, datasets whose instances are generated with various sizes of hyperparameters (m, n_naj, E_ave, DDT) in order to adapt to the complex production environment are provided. A new instance is given to create a novel training environment and train neural networks at each training step. Comparative methods include heuristic rules (FIFO, MRT), a metaheuristic rule (GA), and DRL-based methods (DQN, DDQN). Throughout the whole process of the generalization experiment, n_naj is set at a constant 100. For DRL-based comparative methods, since there is a lack of a decision neural network on reward forms, six reward form calculate their own obtained reward r_i, and the final reward is the sum of the product of the average weighted factor and the reward r_i. In running DRL-based methods, 20 times are repeated, and the objective test result is the mean value of the 20 runs. Without decision randomness, heuristic rules are just run one time. The DQN and the DDQN use the same action space (seven compositive dispatching rules) as the DLDDQN for comparative fairness.

In comparative method GA [42], firstly, jobs, machines, and other scheduling information of the DFJSP are encoded into a chromosome structure, followed by a representation of individuals and populations. Secondly, after initialization, considering the multiple optimization objectives, i.e., makespan and delay time sum, defining a fitness function for evaluation is required. Next, the algorithm employs a selection mechanism, i.e., the roulette wheel strategy, to choose parents for reproduction. Then, genetic operators, including crossover and mutation, are applied to create offspring solutions. Meanwhile, local search techniques are incorporated to refine solutions and exploit the local neighborhood. The dynamic update of the population realizes the dynamic adaptation to the changing problem instances. Hyperparameters of GA are set as follows: population size $pop\_size = 200$; crossover probability $p\_c = 0.8$; mutation probability $p\_m = 0.3$; selection of a way to crossover $p\_v = 0.5$; selection of a way to mutation $p\_w = 0.99$; and the number of iterations $max\_i = 100$.

From the comparative results of Table 7, it can be observed that for most test benchmarks, the proposed framework outperforms any other scheduling methods regarding the makespan objective. Although a part of the test benchmarks cannot derive the best metric result on the objective, i.e., delay time sum, using the DLDDQN method, these results are relatively lower than most of the comparative methods and near to the optimal solution of the other comparative method, which can owe to a good compromise on the two objectives that the proposed framework achieves. It also can be found that apart from the DLDDQN, other methods that have the best comparative result (marked in bold) in test benchmarks are optimal in one certain objective and tend to have the worst result in another, which illustrates that these methods cannot realize a compromise between objectives and result in sacrificing other objectives in order to promote one certain objective. Based on the analysis above, it can be concluded that the DLDDQN has the predominant advantage in handling multi-objective optimization over the other methods. In addition, the generalization ability and practicality of the proposed framework are validated by comparing the metric results of two objectives in various production configurations.

Table 7. Comparison of metric results on objectives between DLDDQN and other methods (makespan/delay time). The bold highlights the best comparative result of two optimization objectives in each row.

DDT	m	E_ave	DLDDQN	DQN	DDQN	FIFO	MRT	GA
0.5	10	50	**443.7**/**22,579.45**	507.85/22,819.38	482.29/22,967.39	907/34,367	842/30,217	523.25/29,486.96
		100	**388.6**/**17,008.4**	438.24/18,645.23	428.51/18,022.21	690/21,203	755/23,733	459.63/20,330.54
		200	**518.7**/**15,851.8**	538.63/17,160.44	533.27/16,155.25	860/20,374	919/26,907	555.39/20,330.54
	20	50	240.75/**2756.15**	200.14/3458.58	**196.88**/3431.4	299/2937	319/8212	205.82/7067.16
		100	**154.5**/**1764.5**	205.52/2423.27	204.15/2393.77	362/1831	362/4995	234.74/5278.68
		200	**150.6**/133.1	183.26/470.16	172.3/394.66	297/**22**	269/1095	194.95/886.46
	30	50	**95.18**/442	112.21/795.8	116.83/505.42	195/**64**	194/1964	113.63/2377.63
		100	**120.6**/233.2	140.56/486.37	132.26/334.0	249/**0**	269/1581	143.94/908.44
		200	**103.24**/37.85	137.41/102.59	138.31/134.42	215/**0**	268/1461	124.22/129.57
1	10	50	**396.4**/**1527.95**	497.04/4478.87	490.7/3349.26	883/20,094	893/33,496	531.87/30,495.17
		100	**343.78**/**9135.8**	417.44/10,434.38	455.07/10,372.92	729/13,589	787/20,811	478.55/23,818.5
		200	**352.4**/**3386.2**	448.13/9912.2	472.53/8401.61	768/4200	847/18,567	493.84/18,260.04
	20	50	**121.5**/231.1	189.26/970	152.98/7291.1	354/**128**	358/5934	205.58/7144.79
		100	**183.76**/**76.75**	199.03/870.99	192.69/208.49	452/87	408/4688	212.75/4589.49
		200	**194**/585.25	235.06/992.59	222.03/539.58	440/**0**	439/2317	217.08/1794.76
	30	50	**101.22**/278.55	109.08/364.37	104.39/285.82	323/**2**	258/3495	125.26/2201.4
		100	**112.83**/449.55	125.63/559.36	127.53/484.35	271/**0**	212/1129	129.03/814.17
		200	157.35/145.9	172.72/119.29	**148.43**/155.65	370/**0**	318/1118	165.5/265.33
2	10	50	**325.78**/**18,573**	382.14/20,218.93	363.33/20,840.85	764/33,589	699/30,139	464.71/25,763.73
		100	**343.11**/**12,541.6**	385.74/14,875.95	370.25/13,807.0	788/30,588	738/28,654	490.46/23,826.31
		200	**351.54**/**9970.55**	382.82/11884.94	419.46/11,155.08	663/16,114	682/18,230	501.84/15,119.49
	20	50	**154.4**/**5052**	180.09/5569.82	178.96/5246.45	297/8110	321/7309	195.67/6984.51
		100	**174.98**/2595.6	194.13/**2263.94**	184.15/2834.0	317/4289	344/7321	216.3/3615.23
		200	**164.8**/833.45	199.35/797.08	188.45/**790.41**	311/1522	356/3366	205.79/1153.18
	30	50	**109.14**/727.25	139.23/730.05	128.46/769.17	197/**589**	182/2286	143.08/3524.76
		100	**102.23**/**158.5**	107.19/162.98	103.04/163.62	171/165	158/843	128.39/776.18
		200	**100.12**/106.3	137.47/137.89	107.78/132.74	193/**78**	201/428	126.18/172.81

5. Conclusions and Future Research Potentials

Different from previous research on algorithms of single objective optimization, in our work, we present a dual layer DDQN architecture to realize optimization of multiple objectives on the DFJSP, among which the machine's makespan on completing the system task and the job's total delay time sum are minimized simultaneously, performing better on a large proportion of instances compared with other algorithms with only one objective optimized under the same numerical example test. Through the structure of the two-agent DLDDQN, five features as the higher DDQN's state vector, reflecting the performance of objectives, are extracted; six forms of rewarding an executed action corresponding to a metric of two objectives are designed as the goal space, and one goal is selected from the higher DDQN to be an input feature of lower DDQN along with other five higher features; and seven high-quality dispatching rules as the action space are created. At each rescheduling point, the lower DDQN determines an eligible rule to be executed, with the most cumulated reward and the lowest metric results in optimizing objectives locally and globally.

The training process of the DDQN can be carried out with an exact number of machines and jobs, which is a simplified problem satisfying DRL. Although a simplified trained model does not apply to scenarios with different sizes of machines and jobs, variations such as the processing time, the number of a job's operations, available machines of an operation, and the randomness of job arrivals can be tolerant [22]. Another training way to generalize a model with the two variables included is to generate datasets of all possible production configurations and to use these benchmark examples during each training episode to train a convergent model. Given a test example within the production configuration, the trained model can feed back a near-optimal answer within its knowledge range. However, the cost for the two types of training way can be matched. Since the simplified trained model is limited to specified scenarios, users have to spend computational resources and time again in order to create a new model adaptive to new scenarios. A complex model trained with more datasets has a better generalization ability to handle diverse scenarios. However, resources and time costs in advance to train the model can be many times that of the simplified model, which can be a drawback. To sum up, both types are feasible, and they should be chosen for application with the requirements of reality.

One of the limitations of this study is that it cannot always be ensured to obtain the optimal solution on optimization objectives, even though it is enabled to reach approximal optimum when given instances within its knowledge base, because our action space is limited and it may have some other more excellent compositive dispatching rules that are not included in our action space. Due to the problem of action space explosion, we cannot embrace all possible dispatching rules. A feasible way to shrink action space is to attempt comparative experiments on possible compositive dispatching rules for the best several ones. Furthermore, with the increment of the problem size in instantiated examples, the possible policies become exponentially large, and the time to train a model for the optimal policy will be unacceptably long with invariant computational resources. Likewise, as the dataset grows larger, the time cost to converge the model will be more prolonged or unacceptable.

For future work, the dynamic mechanism can be intensified, making it possible for the trained model to tolerate more disruptive events. When an incident occurs, the trained model searches its knowledge base and finds if the new problem can be solved by itself, during which a scheduling training restart for learning can be motivated if the new problem is beyond its knowledge range. This way, trained model self-handling dynamic events can be formed, and generality is further widened. Moreover, in factory production scenes, optimization objectives can be vast and expansive; therefore, utilizing the proposed framework to solve new production scenes of reality can be feasible. In addition, it must be noted that if designed imperfectly, state space can be highly correlative, and it is a necessity to develop algorithms to eliminate the correlation between state features. In the application of reality, demand for schedule schemes can be urgent; if the DRL method cannot respond in time, desirable approaches such as meta-heuristic algorithms are recommended to be combined as backup scheduling methods. Finally, other policy-based DRL methods such as Actor-Critic and PPO are worth undergoing experiments on objective result comparisons for better solutions to solve the DFJSP with multiple optimization objectives.

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

Funding: This research received no external funding.

Data Availability Statement: Data of experiments are all included in this article.

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

References

1. Mohan, J.; Lanka, K.; Rao, A.N. A review of dynamic job shop scheduling techniques. *Procedia Manuf.* **2019**, *30*, 34–39. [CrossRef]
2. Xiong, H.; Shi, S.; Ren, D.; Hu, J. A survey of job shop scheduling problem: The types and models. *Comput. Oper. Res.* **2022**, *142*, 105731. [CrossRef]
3. Zhou, H.; Gu, B.; Jin, C. Reinforcement Learning Approach for Multi-Agent Flexible Scheduling Problems. *arXiv* **2022**, arXiv:2210.03674.
4. 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.
5. 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]
6. Monaci, M.; Agasucci, V.; Grani, G. An actor-critic algorithm with deep double recurrent agents to solve the job shop scheduling problem. *arXiv* **2021**, arXiv:2110.09076.
7. Ferreira, C.; Figueira, G.; Amorim, P. Effective and interpretable dispatching rules for dynamic job shops via guided empirical learning. *Omega* **2022**, *111*, 102643. [CrossRef]
8. Inal, A.F.; Sel, Ç.; Aktepe, A.; Türker, A.K.; Ersöz, S. A Multi-Agent Reinforcement Learning Approach to the Dynamic Job Shop Scheduling Problem. *Sustainability* **2023**, *15*, 8262. [CrossRef]
9. Chang, J.; Yu, D.; Zhou, Z.; He, W.; Zhang, L. Hierarchical Reinforcement Learning for Multi-Objective Real-Time Flexible Scheduling in a Smart Shop Floor. *Machines* **2022**, *10*, 1195. [CrossRef]
10. Ahmadi, E.; Zandieh, M.; Farrokh, M.; Emami, S.M. A multi objective optimization approach for flexible job shop scheduling problem under random machine breakdown by evolutionary algorithms. *Comput. Oper. Res.* **2016**, *73*, 56–66. [CrossRef]
11. Zhu, J.; Wang, H.; Zhang, T. A Deep Reinforcement Learning Approach to the Flexible Flowshop Scheduling Problem with Makespan Minimization. In Proceedings of the 2020 IEEE 9th Data Driven Control and Learning Systems Conference (DDCLS), Liuzhou, China, 19–21 June 2020; pp. 1220–1225. [CrossRef]
12. Garey, M.R.; Johnson, D.S.; Sethi, R. The complexity of flowshop and jobshop scheduling. *Math. Oper. Res.* **1976**, *1*, 117–129. [CrossRef]
13. Xie, J.; Gao, L.; Peng, K.; Li, X.; Li, H. Review on flexible job shop scheduling. *IET Collab. Intell. Manuf.* **2019**, *1*, 67–77. [CrossRef]
14. Luo, S. Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. *Appl. Soft Comput.* **2020**, *91*, 106208. [CrossRef]
15. 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]
16. Panzer, M.; Bender, B. Deep reinforcement learning in production systems: A systematic literature review. *Int. J. Prod. Res.* **2022**, *60*, 4316–4341. [CrossRef]
17. Sutton, R.S.; McAllester, D.; Singh, S.; Mansour, Y. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in Neural Information Processing Systems 12*; MIT Press: Cambridge, MA, USA, 2000.
18. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347.
19. 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]
20. Hu, H.; Jia, X.; He, Q.; Fu, S.; Liu, K. Deep reinforcement learning based AGVs real-time scheduling with mixed rule for flexible shop floor in industry 4.0. *Comput. Ind. Eng.* **2020**, *149*, 106749. [CrossRef]
21. Lei, K.; Guo, P.; Zhao, W.; Wang, Y.; Qian, L.; Meng, X.; Tang, L. A multi-action deep reinforcement learning framework for flexible Job-shop scheduling problem. *Expert Syst. Appl.* **2022**, *205*, 117796. [CrossRef]
22. Workneh, A.D.; Gmira, M. Learning to schedule (L2S): Adaptive job shop scheduling using double deep Q network. *Smart Sci.* **2023**. [CrossRef]
23. Zhang, M.; Lu, Y.; Hu, Y.; Amaitik, N.; Xu, Y. Dynamic Scheduling Method for Job-Shop Manufacturing Systems by Deep Reinforcement Learning with Proximal Policy Optimization. *Sustainability* **2022**, *14*, 5177. [CrossRef]
24. Liu, R.; Piplani, R.; Toro, C. Deep reinforcement learning for dynamic scheduling of a flexible job shop. *Int. J. Prod. Res.* **2022**, *60*, 4049–4069. [CrossRef]
25. Zhang, C.; Song, W.; Cao, Z.; Zhang, J.; Tan, P.S.; Xu, C. Learning to dispatch for job shop scheduling via deep reinforcement learning. In Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS'20), Vancouver, BC, Canada, 6–12 December 2020; Curran Associates Inc.: Red Hook, NY, USA, 2020; pp. 1621–1632.
26. Luo, B.; Wang, S.; Yang, B.; Yi, L. An improved deep reinforcement learning approach for the dynamic job shop scheduling problem with random job arrivals. *J. Phys. Conf. Ser.* **2021**, *1848*, 012029. [CrossRef]
27. Wang, L.; Hu, X.; Wang, Y.; Xu, S.; Ma, S.; Yang, K.; Liu, Z.; Wang, W. Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning. *Comput. Netw.* **2021**, *190*, 107969. [CrossRef]
28. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An introduction*; MIT Press: Cambridge, MA, USA, 2018.
29. 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]

30. Van Hasselt, H.; Guez, A.; Silver, D. Deep reinforcement learning with double q-learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016; Volume 30.
31. Lapan, M. *Deep Reinforcement Learning Hands-On: Apply Modern RL Methods, with Deep Q-Networks, Value Iteration, Policy Gradients, TRPO, AlphaGo Zero and More*; Packt Publishing Ltd.: Birmingham, UK, 2018.
32. Dolcetta, I.C.; Ishii, H. Approximate solutions of the Bellman equation of deterministic control theory. *Appl. Math. Optim.* **1984**, *11*, 161–181. [CrossRef]
33. Rafati, J.; Noelle, D.C. Learning representations in model-free Real-Time Flexible Scheduling. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27–28 January 2019; Volume 33, pp. 10009–10010.
34. Pateria, S.; Subagdja, B.; Tan, A.H.; Quek, C. Hierarchical reinforcement learning: A comprehensive survey. *ACM Comput. Surv. CSUR* **2021**, *54*, 1–35. [CrossRef]
35. 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]
36. Puterman, M.L. Markov decision processes. In *Handbooks in Operations Research and Management Science*; Elsevier: Amsterdam, The Netherlands, 1990; Volume 2, pp. 331–434.
37. Fan, J.; Wang, Z.; Xie, Y.; Yang, Z. A theoretical analysis of deep Q-learning. In Proceedings of the 2nd Conference on Learning for Dynamics and Control, Berkeley, CA, USA, 11–12 June 2020.
38. Lv, P.; Wang, X.; Cheng, Y.; Duan, Z. Stochastic double deep Q-network. *IEEE Access* **2019**, *7*, 79446–79454. [CrossRef]
39. Nachum, O.; Gu, S.S.; Lee, H.; Levine, S. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems 31*; Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2018; pp. 3303–3313.
40. Han, B.-A.; Yang, J.-J. Research on Adaptive Job Shop Scheduling Problems Based on Dueling Double DQN. *IEEE Access* **2020**, *8*, 186474–186495. [CrossRef]
41. Li, Y.; Gu, W.; Yuan, M.; Tang, Y. Real-time data-driven dynamic scheduling for flexible job shop with insufficient transportation resources using hybrid deep Q network. *Robot. Comput. Integr. Manuf.* **2022**, *74*, 102283. [CrossRef]
42. Katoch, S.; Chauhan, S.S.; Kumar, V. A review on genetic algorithm: Past, present, and future. *Multimed. Tools Appl.* **2021**, *80*, 8091–8126. [CrossRef] [PubMed]