mathematics

MDPI

Article

Multi-AGV Dynamic Scheduling in an Automated Container Terminal: A Deep Reinforcement Learning Approach

Xiyan Zheng [1], Chengji Liang [1], Yu Wang [1,*], Jian Shi [2] and Gino Lim [3]

[1] Institute of Logistics Science and Engineering, Shanghai Maritime University, Shanghai 201306, China
[2] Department of Engineering Technology, University of Houston, Houston, TX 77004, USA
[3] Department of Industrial Engineering, University of Houston, Houston, TX 77004, USA
* Correspondence: wangyu@shmtu.edu.cn

Abstract: With the rapid development of global trade, ports and terminals are playing an increasingly important role, and automatic guided vehicles (AGVs) have been used as the main carriers performing the loading/unloading operations in automated container terminals. In this paper, we investigate a multi-AGV dynamic scheduling problem to improve the terminal operational efficiency, considering the sophisticated complexity and uncertainty involved in the port terminal operation. We propose to model the dynamic scheduling of AGVs as a Markov decision process (MDP) with mixed decision rules. Then, we develop a novel adaptive learning algorithm based on a deep Q-network (DQN) to generate the optimal policy. The proposed algorithm is trained based on data obtained from interactions with a simulation environment that reflects the real-world operation of an automated in Shanghai, China. The simulation studies show that, compared with conventional scheduling methods using a heuristic algorithm, i.e., genetic algorithm (GA) and rule-based scheduling, terminal the proposed approach performs better in terms of effectiveness and efficiency.

Keywords: Multi-AGV scheduling; automated container terminal; mixed decision rules; deep reinforcement learning; simulation-based algorithm analysis

MSC: 90B06

Citation: Zheng, X.; Liang, C.; Wang, Y.; Shi, J.; Lim, G. Multi-AGV Dynamic Scheduling in an Automated Container Terminal: A Deep Reinforcement Learning Approach. *Mathematics* **2022**, *10*, 4575. https://doi.org/10.3390/math10234575

Academic Editor: Víctor Yepes

Received: 12 November 2022
Accepted: 28 November 2022
Published: 2 December 2022

1. Introduction

Ports and terminals play an important role in cargo transshipment and loading/unloading operations to support the advancement of global trade. As the global throughputs of containers have continued to grow for decades and are expected to continue increasing in the future [1], it is crucial to improve the efficiency and competitiveness of container terminals. Driven by the recent Industry 4.0 trends and the development of information technology, automated container terminals have become an attractive concept that has been highly valued by terminal operators around the world.

Figure 1 displays a representative process of container loading/unloading operations in an automated terminal that involves vertical and horizontal transportation by multiple equipment. By taking the import of containers as an example, quay cranes (QCs) are first assigned to berthing ships to unload the containers and transfer them to the AGVs, and the AGVs will transport the containers to a designated block of the container yard. When AGVs reach the designated block, AGV mates are used as a buffer to unload the containers from the AGVs and hold them until yard cranes (YCs) pick up the containers and transfer them to the corresponding location. The coordination of QCs and AGVs directly determines the efficiency of the loading and unloading operations on the sea side.

In order to make full use of scarce resources such as the QCs, it is important to optimize the scheduling of AGVs such that the efficiency of horizontal transportation at the automated terminal can be improved [2]. However, reaching this objective can be very challenging due to the complex and dynamic operating environment of a terminal.

Situations such as early or late arrival of ships and conflicts and failures of the equipment may occur. Moreover, AGVs may stop or delay due to unforeseen situations linked to other equipment and facilities. It is thus evident that the complex and changing nature of the automated terminal should be taken into account while making scheduling decisions for the AGVs.

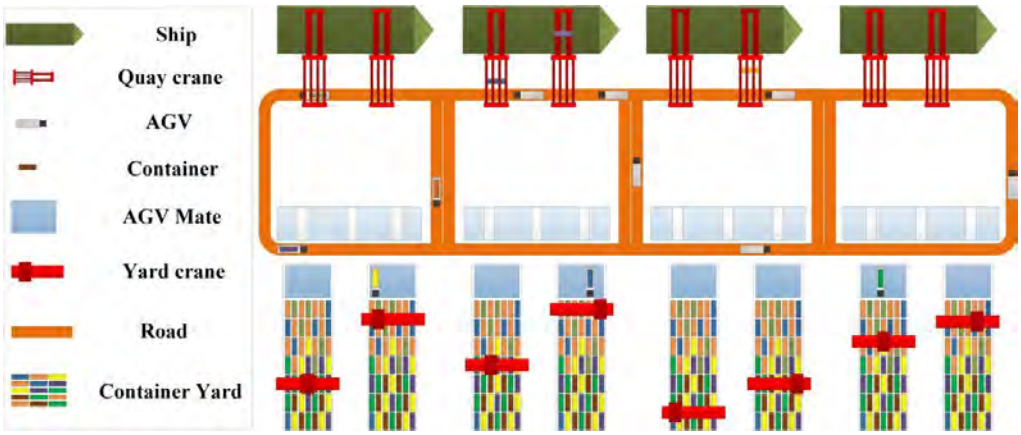


Figure 1. Automated terminal seaside scheduling scenario.

Traditional static methods are hard to adapt to the varying environment of the automated terminals. In order to improve operational efficiency, it is necessary to develop data-driven approaches to solve the dynamic scheduling problem of AGVs. In this paper, we propose a dynamic decision-making approach based on deep Q-network (DQN) for the multi-AGV dynamic scheduling problem in an automated terminal. As shown in Figure 2, key dynamic features in the terminal system are first recognized, and a deep Q-network is defined by the information on container tasks, the AGVs and several operation rules. Then, related data on the system status are sent to train the deep Q-network such that scheduling decisions can be made adaptively. A trained deep Q-network will be acquired after sufficient learning iterations, which can cope with most uncertainty factors in ports and could be applied to the automated terminal AGV dynamic scheduling efficiently.

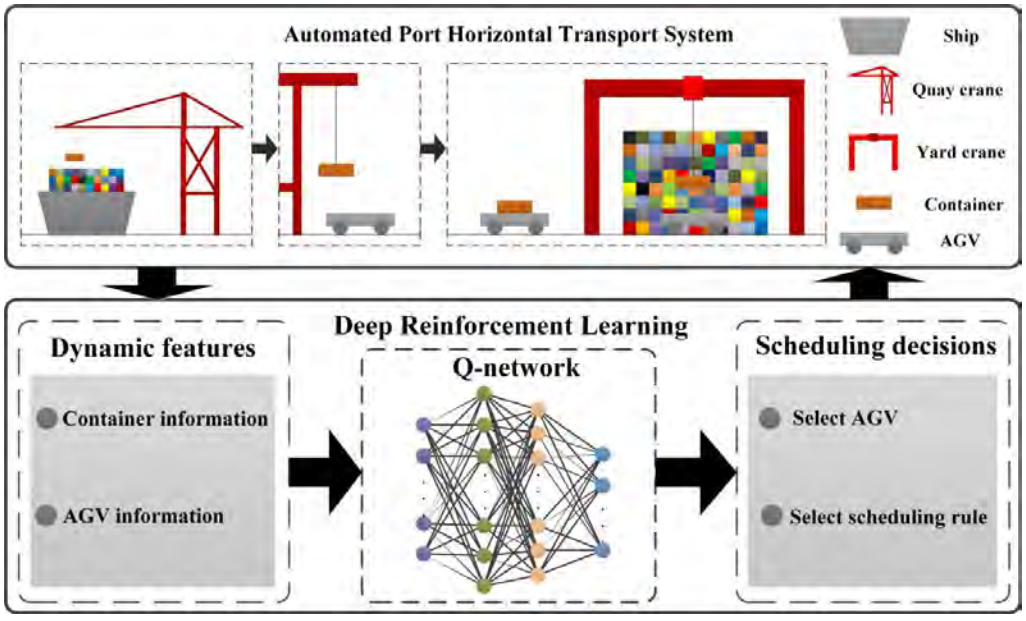


Figure 2. DQN-based dynamic scheduling method architecture of the automated terminal AGVs.

The main research contributions of this paper are as follows:

(1) The dynamic scheduling problem of AGVs at automated terminals is formulated as a Markov decision process, in which the dynamic information of the terminal system, such as the number of tasks, task waiting time, task transportation distance, the working/idle status of the AGVs and the position of the AGVs, are modeled as input states.
(2) A novel dynamic AGV scheduling approach based on DQN is proposed, where the optimal mixed rule and AGV assignments can be selected efficiently to minimize the total completion time of AGVs and the total waiting time of QCs.
(3) A simulation model is built on Tecnomatix Plant Simulation software according to an automated terminal in Shanghai, China. Simulation-based experiments are conducted to evaluate the performance of the proposed approach. The experimental results show that the proposed approach outperforms the conventional heuristic approach based on GA and rule-based scheduling.

The rest of this paper is organized as follows. Section 2 is the literature review. Section 3 formally describes the dynamic scheduling AGV problem as an MDP. In Section 4, the DQN-based dynamic scheduling approach of automated terminal AGVs is proposed. Section 5 introduces the simulation-based training process, and Section 6 discusses the experimental results. Section 7 offers conclusions and future directions.

2. Literature Review

In this section, we reviewed the related literature on the static scheduling problems of AGVs, the traditional methods for dynamic scheduling of AGVs and the application of machine learning-based approaches in scheduling optimization.

Most of the traditional approaches for AGV scheduling in automated terminals are conducted based on analytical models that capture the configuration and operation status of the terminal. For instance, Yang et al. [3] proposed an integrated problem for equipment coordination and AGV routing. They set up a bilevel programming model to minimize the makespan and used the congestion prevention rule-based bilevel genetic algorithm (CPR-BGA) to solve the problem. Xu et al. [4] designed a load-in-load-out AGV route planning model with the help of a buffer zone, where an AGV can carry at most two containers, and used a simulated annealing algorithm to solve it. Zhong et al. [5] combined the two problems of AGV conflict-free path planning with quay cranes (QCs) and rail-mounted gantry (RMG) cranes to implement the integrated scheduling of multi-AGVs with conflict-free path planning and a series of small-scale and large-scale experiments were performed through the hybrid genetic algorithm-particle swarm optimization (HGA-PSO). Zhang et al. [6] developed a collaborative scheduling model of AGVs and ASCs in automatic terminal relay operation mode based on the genetic algorithm by considering the buffer capacity constraint and twin ASC operation interference. Even though the above approaches have shown good performance in a static and non-varying environment, they cannot sufficiently handle the complexity and dynamics involved in real-world terminal operations.

In order to address this issue, a handful of the literature has studied the dynamic AGV scheduling problems to be more adaptive in automated terminals. In the early days, various scheduling rules were used to dispatch AGVs in the dynamic scheduling problem [7,8], such as the first-come-first-serve (FCFS) rule, shortest travel distance (STD) rule, longest waiting time (LWT) rule, etc. Studies on dynamic scheduling problems indicated that using multiple dispatching rules could enhance the scheduling performance to a greater extent than using a single rule [9]. Angeloudis and Bell [10] studied job assignments for AGVs in container terminals with various conditions of uncertainty. They developed a new AGV dispatching approach that is capable of operating within a certain container terminal model. Gawrilow et al. [11] enhanced a static approach to meet the requirements of the dynamic problem. They developed a dynamic online routing algorithm that computes collision-free routes for AGVs by considering implicit time-expanded networks. Cai et al. [12] investigated replanning strategies for container-transportation task allocation of autonomous

Straddle Carriers (SC) at automated container terminals. By focusing on the uncertainty of the arrival of new jobs, this paper proposes two rescheduling policies, the newly arrived job Rescheduling (RNJ) policy and the newly unexecuted job rescheduling (RCJ) policy. The previous research on the dynamic scheduling of AGVs usually depends on specific assumptions of uncertain tasks or environmental factors, thus can only solve the problem within a given scenario. The related approaches are hard to generalize to other scenarios. Moreover, the complex environment of real terminals is hard to be defined by simple assumptions. Traditional optimization approaches cannot solve the dynamic scheduling of AGVs efficiently in real-world applications.

In order to overcome the aforementioned drawbacks in previous studies, machine learning techniques have been recently introduced [13–15]. Among the related studies, reinforcement learning (RL) methods are the most used ones, which can constantly adjust the agent's behavior through trial and error such that dynamic and uncertain environmental data can be fully considered. At present, most of the research on scheduling using RL is focused on manufacturing shop floor production [16–18]. Several research has focused on AGV scheduling in automated terminals [19,20]. Jeon et al. [21] studied an AGV path optimization problem in automated container terminals and proposed a method for estimating for each vehicle the waiting time that results from the interferences among vehicles during traveling by using the Q-learning technique and by constructing the shortest time routing matrix for each given set of positions of quay cranes. Choe et al. [22] proposed an online preference learning algorithm named OnPL that can dynamically adapt the policy for dispatching AGVs to change situations in an automated container terminal. The policy is based on a pairwise preference function that can be repeatedly applied to multiple candidate jobs to sort out the best one and to reduce waiting times for outside container trucks at container terminals.

Despite the insights offered by the above RL-based studies, one limitation they share is their performance and learning efficiency with the increase in environmental complexity. As RL relies on storing all possible states and actions in policy tables, when the number of state variables becomes larger, the performance of the algorithm may degrade significantly due to the expansion of the state space. In some cases, the state space becomes so large that the learning problem becomes intractable [23]. Deep reinforcement learning (DRL) was then introduced and has achieved impressive successes over the last several years. DRL methods use the deep neural network to capture the complex state features instead of the policy table, such that the loss of state information is greatly reduced [24]. The application of DQN in electric vehicle charging scheduling [23] and energy management [25] has also demonstrated its strong superiority and effectiveness. However, it has not been applied to the dynamic scheduling of AGVs in an automated terminal.

3. Problem Formulation

3.1. MDP Model the Dynamic Scheduling of Multi-AGVs

The dynamic scheduling of AGVs at a seaside terminal can be described and formulated as a Markov decision process (MDP) as follows: After a ship arrives at the terminal, the assigned QCs unload the imported containers one by one from the ship to the AGVs. Then, the AGVs transport each of the containers to a prespecified location at the container yard and then return for the next container. The dynamic scheduling of AGVs can be modeled as a sequential decision-making process in which multiple rules can be used to determine how to assign each of the containers to the available AGVs. Specifically, we propose an MDP model to formulate the process, denoted by $(S, a, P, R, \gamma, \pi)$. S is the set of states, which contains all possible states of the AGV scheduling process; a represents the actions that can be performed; P is the probability of transitioning from the current state $s \in S$ to the next state $s' \in S$ when action a is taken; γ is the discount factor; $R(s,\ a,\ s')$ is the reward function that evaluates the reward obtained by performing action a in state s; π is the policy that maps from the state set s to the set of actions a; and $\pi(s,\ a)$ represents

the probability of taking action a in a given state s. The detailed definitions of the states, actions, reward function and optimal mixed rule policy, are as follows:

3.2. State Definition

The state is used to represent the status information of the system. After a container ship ports in a specific berth, the terminal schedules an AGV according to the required ship total loading/unloading time and ship loading information. In the real-world scheduling process, the scale of container tasks and their priorities, the operation efficiency and occupied rates of the equipment, and the features of the infrastructure are usually considered to be important while making scheduling decisions for the AGVs. In the proposed MDP model, we define the state at time t as a five-tuple vector $s_t = (N_t, T_{awt}, D_{adt}, A_{st}, A_{iloc})$.

(1) N_t is the number of container tasks current on QCs which waiting for AGV transport, indicating the current workload in the terminal.

(2) T_{awt} represents the average waiting time of the container task, which is waiting for AGV transport currently on QCs. It indicates the average urgency of the current task. The T_{awt} is defined as follows:

$$T_{awt} = \frac{\sum_{k=1}^{N_t} t_k}{N_t} \tag{1}$$

where t_k is the waiting time of the k-th container task that awaits transportation by the AGVs.

(3) D_{adt} represents the average transportation distance of the container tasks on the QCs. This metric reflects the average workload per task and is defined as follows:

$$D_{adt} = \frac{\sum_{k=1}^{N_t} d_k}{N_t} \tag{2}$$

where d_k is the travel distance of the k-th task from the related QC to its destination location at the yard.

(4) A_{st} represents the working status of all the alternative AGVs, which can be represented by a binary vector in the form of:

$$A_{st} = n_1 n_2 \dots n_i \tag{3}$$

where $n_i = 1$ represents the working status of AGV i as "Working" and $n_i = 0$ represents the working status of AGV i as "Idle".

(5) A_{iloc} represents the dynamic position of AGV i in the port, determined by a given pair of coordinates (x_i, y_i).

3.3. Action Definition

The actions are all the possible scheduling decisions of all the AGVs. An action can be denoted as $a_t = (Ru_t, AGVt)$, where Ru_t is the scheduling rule that determines the order of the task assignments and $AGVt$ is the index of the AGV to be scheduled for the selected task. Three typical scheduling rules are considered: first come, first serve (FCFS); shortest task distance (STD); and longest wait task (LWT) [7,8]. The details of the rules are shown in Table 1. Based on these rules, we can define $Ru_t = \{1$ (if FCFS), 2 (if STD), 3 (if LWT)$\}$, $AGV_t = i$ (if AGV i is selected). By evaluating possible actions, the container tasks can be selected according to specific rules and assigned to the corresponding AGVs at time t.

Table 1. Scheduling rules.

Rule	Description	Advantage	Shortcoming
FCFS	Tasks are selected based on the order of arrival	FCFS can ensure the overall efficiency and smoothness of scheduling	FCF is not effective at meeting other metrics than efficiency, such as task priorities and travel costs
STD	Task with the shortest trip will be selected first	STD can improve overall efficiency to a certain extent	STD can cause longer wait times for tasks and longer trips
LWT	Task with the longest waiting time will be selected first	LWT can effectively reduce the waiting time for tasks and ensure production efficiency	LWT cannot effectively meet other metrics than waiting time, such as efficiency and travel costs

3.4. Reward Function

In order to evaluate the actions, we first define the cost functions with the objective of minimizing the total completion time of the AGVs and the total waiting time of the QCs as follows.

$$C_{ik}^{r} = \alpha(M - t_{ik}^{r}) \tag{4}$$

$$C_r = \alpha \sum_i \sum_k C_{ik}^{r} = \alpha \sum_i \sum_k (M - t_{ik}^{r}) \tag{5}$$

where C_{ik}^{r} is the cost determined by the completion time of AGV i after handling container task k, the completion time is denoted as t_{ik}^{r}, M is a sufficiently large constant, α is a given cost coefficient and C_r is the total completion time cost of all the AGVs over the entire scheduling process.

$$C_{ik}^{w} = \beta(N - t_{ik}^{w}) \tag{6}$$

$$C_w = \beta \sum_i \sum_k C_{ik}^{w} = \beta \sum_i \sum_k (N - t_{ik}^{w}) \tag{7}$$

In Equations (6) and (7), C_{ik}^{w} is the penalty cost of the QCs waiting for the AGV in the process of AGV i handling task k, where t_{ik}^{w} is the time that the QC waits for the AGV when AGV i performs task k, N is a sufficiently large constant, β is the waiting time cost coefficient of the QCs and C_w is the total waiting time cost of the QC in the whole scheduling process.

Based on the above cost functions, the reward function can be defined as follows:

$$R_t = \mu_1 C_{ik}^{r} + \mu_2 C_{ik}^{w} \tag{8}$$

$$R_f = \mu_1 C_r + \mu_2 C_w \tag{9}$$

where R_t is the current reward for evaluating the operation of each individual container task, R_f is the final reward for evaluating the overall performance of the schedule and μ_1 and μ_2 are the given weight parameters.

3.5. Optimal Mixed Rule Policy Based on Reinforcement Learning

In the RL framework, the learning process consists of repeated interactions between the system, usually known as the agent, and the environment, through continuous trial and error. At each stage of this iterative process, two steps need to be completed as described below:

(1) Prediction: given a policy and evaluation function, the value function corresponding to a state and action, denoted as $Q^{\pi}(s, a)$, is:

$$Q^{\pi}(s, a) = E_{\pi}\{R|s_t = s, a_t = a\} \tag{10}$$

where $E_{\pi}\{R|s_t = s, a_t = a\}$ is the expected value of reward when the state starts from s, takes action a and follows policy π [26–28]. For any pair of $(s,\ a)$, we can calculate its value function based on Equation (10).

(2) Control: to find the optimal policy based on the value function. As the goal of reinforcement learning is to obtain the optimal policy, i.e., the optimal selection of actions in different states, with multiple scheduling rules used as actions, the mixed

rule policy in our proposed model can be defined as the expected discounted future reward when the agent takes action a in state s as follows:

$$Q^{\pi}(s,a) = E_{\pi}\{R|s_t = s, a_t = a\} = E_{\pi}\{R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{*} R_{t+k+1}\} \quad (11)$$

where γ is the discount factor and R_{t+k+1} is the current reward at time $t+k+1$. Based on Equation (12), the multi-AGV dynamic scheduling problem is to find an optimal mixed rule policy π^{*} in each state s such that the reward obtained in the following form is maximized:

$$Q^{\pi^{*}}(s,a) = \max E_{\pi}\{R|s_t = s, a_t = a\} = \max Q^{\pi}(s,a), \forall s \in S, \forall a \in A \quad (12)$$

4. DQN-Based Scheduling Approach

In order to obtain the optimal mixed rule policy, a DQN-based approach is proposed in this paper. Based on the problem formulation described in Section 2, it is evident that the states of the automated terminal system contain uncertain values, so the state space will be very large. A large state space would not only need a huge amount of replay memory to store large tables but would also consume a long time to fill and search the tables accurately. In order to address this issue, we propose a deep Q-network (DQN) in which a neural network is used as a nonlinear approximation of the optimal action-value function; it is denoted as $Q(s,\ a,\ \theta) \rightarrow Q^{*}(s,\ a)$, where θ represents all the parameters of the neural network.

4.1. DQN Training Process

The goal of the training process is to reduce the estimation error between the DQN network and the optimal value function. The process is conducted by iteratively updating the parameters of the neural network [29–31]. DQN directly takes raw data (states) as input and generates the Q-value function of each state-action pair defined by Equation (10), which can handle a complex decision-making process with a large continuous state space.

Based on the proposed model, we designed two fully connected neural networks, namely, the main Q-network and the target Q-network. Each network consists of one input layer, two hidden layers and one output layer, and each layer contains a given number of neuron nodes. According to the definition of states and actions, the first three nodes of the input layer represent the task information N_t , T_{awt} and D_{adt}, and the remaining nodes correspond to the status information of each of the AGVs, including the working status n_i and the position coordinates $A_{iloc}(x_i, y_i)$. The number of nodes in the output layer is determined by a combination of the AGVs and the scheduling rules. Each node of the output layer represents a possible combination of the AGVs and the scheduling rules. The number of nodes in each of the hidden layers and the activation functions among the different layers are key parameters to be decided based on the input and output layers, which will impact the calculation accuracy of the neural network. There is no perfect theory to determine the number of hidden layer neurons or activation functions. The number of nodes in each hidden layer is decided by the simulations in our work, as in most of the literature. The ReLU function is used as the activation function between the input layer and the first hidden layer, as well as between the two hidden layers. The ReLU function is commonly used to increase the nonlinear relationship between the various layers of the neural network and to alleviate the occurrence of the overfitting problem [32]. The softmax function is adopted as the activation function between the second hidden layer and the output layer in our approach, which is also known as the normalized exponential function and is commonly used to show the results of multiple classifications in the form of a probability [33]. As shown in Figure 3, the major steps of the training algorithm can be described as follows:

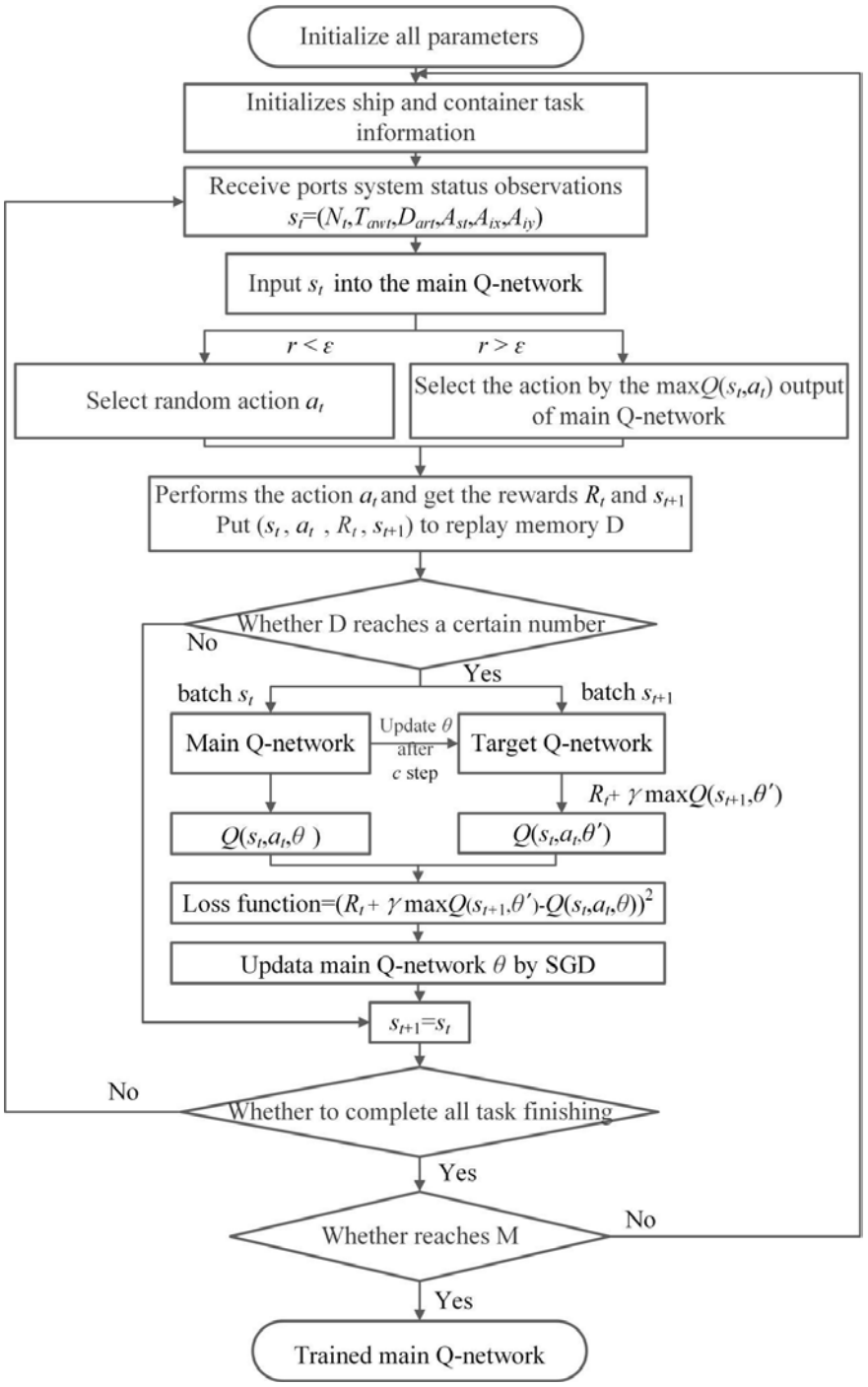


Figure 3. DQN training process.

Step 1: Initialization of parameters, including the parameters θ of the main Q-network and the target Q-network, discount factor γ, training times M and replay memory size D;

Step 2: When a QC completes the unloading of a container from ships or an AGV completes its current task, the scheduling process is triggered to calculate the current system state s_t and sent it to the main Q-network;

Step 3: Main Q-network outputs action value function $Q(s_t;\theta)$ for action selected by the ε-greedy policy. This policy is to produce a random number of r when $r < \varepsilon$, select one action a_t randomly in all actions, and when $r > \varepsilon$, select the action a_t corresponding to max$Q(s_t;\theta)$;

Step 4: The port environment parses the selected actions into AGV ID and scheduling rule and performs them, gaining the reward R_t and the next state s_{t+1}. A complete set of information denoted as the vector (s_t, a_t, R_t, s_{t+1}) is stored in the replay memory;

Step 5: When the samples in the replay memory reach a predetermined amount, the stored records are randomly sampled from the replay memory store to avoid an excessive correlation of the network. The sampled records are sent to the main Q-network and the target Q-network. These two Q-networks have the same neural network structure, and the parameters of the main Q-network and target Q-network are denoted as θ and θ', respectively. The value of the action-value function is predicted by the two networks, and the loss function is calculated according to the error between the two networks. The action value function $Q(s_t, a_t; \theta)$ is directly calculated by the main Q-network, and the action value function $Q(s_t, a_t; \theta')$ predicts the maximum $Q(s_{t+1}, a_{t+1}; \theta')$:

$$Q(s_t, a_t; \theta') = \begin{cases} R_t, & s_t = Terminal \\ R_t + \gamma \max Q(s_{t+1}; \theta'), else \end{cases} \tag{13}$$

Therefore, the loss function $L(\theta)$ is defined as:

$$L(\theta) = \left(Q(s_t, a_t; \theta') - Q(s_t, a_t; \theta)\right)^2 = \left(R_t + \gamma \max Q(s_{t+1}; \theta') - Q(s_t, a_t; \theta)\right)^2 \tag{14}$$

Step 6: The parameters θ of the main Q-network will be updated iteratively by a gradient descent algorithm to reduce the loss function, which is widely used in deep learning [34]. The parameter θ update formula of the gradient descent algorithm is shown in Equation (15):

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} L(\theta_t) \tag{15}$$

Step 7: The parameter θ' of the target Q-network will be updated every C step, and the update method is to completely assign the parameter θ of the main Q-network to the target Q-network. This event will continue to repeat until the reward function converges or the number of training iterations reaches a certain value;

Step 8: If all the tasks are being performed, conduct the next training time. Otherwise, make $s_t = s_{t+1}$, and jump to Step 2;

Step 9: If the training times episode reaches M, terminate the training;

The training Algorithm 1 can be summarized as the following pseudocode:

Algorithm 1: deep Q-learning with experience reply

Initialize reply memory D to capacity N
Initialize action − value function Q with random weights θ
Initialize target action − value function $\overline{Q}$ with random weights $\theta' = \theta$
For episode $= 1,\ M$ **do**
Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
For t $= 1,\ T$ **do**
With probability ε select a random action a_t
Otherwise select $a_t = \text{argmax} Q(\phi(s), a, \theta)$
Execute action a_t in emulator and observe reward r_t and image x_{t+1}
Set $s_{t+1} = s_t,\ a_t,\ s_{t+1}$ and preprocess $\overline{Q}(\phi_{t+1}) = \phi(s_{t+1})$
Store transition $(\phi_t, a_t, r_t,\ \phi_{t+1})$ in D
Sample random minibatch of transitions $\left(\phi_j, a_j, r_j,\ \phi_{j+1}\right)$ from D
Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \overline{Q}\left(\phi_j,\ a'; \theta'\right) & \text{otherwise} \end{cases}$
Perform a gradient descent step on $\left(y_i - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
Every C steps reset $\overline{Q} = Q$
End For
End For

4.2. Scheduling Procedure with Mixed Rules

Using the well-trained main Q-network as the optimal mixed rule policy to guide the dynamic scheduling of AGVs in automated terminals is shown in Figure 4. First, when the QC spreader grabs a new container task or one of the AGVs completes the transportation of a container task and becomes idle, a scheduling request is triggered. At this point, the current automated terminal system state will be sent to the main Q-network. Then, select the action corresponding to the maximum node output from the main Q-network, and convert it into the corresponding scheduling rule and AGV index i. Finally, when AGV i completes the corresponding container task, it recalculates the system state and sends it to the main Q-network for the next action selection. The whole process ends when all ships and containers are handled.

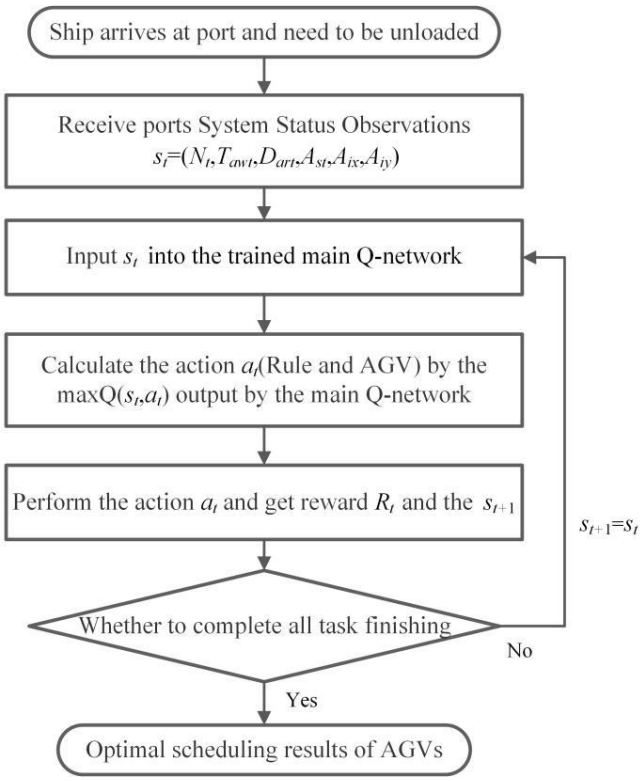


Figure 4. Optimal mixed rule scheduling of automated terminal AGVs.

5. Simulation-Based Experiments

In this section, we evaluate the performance of the proposed approach through a set of simulation-based case studies created based on an automated terminal in Shanghai, China.

5.1. Simulation Set Up

We used Siemens Tecnomatix Plant Simulation 15.0 to simulate the seaside horizontal transportation and vertical loading/unloading of the automated terminal. Through the built-in SimTalk language, a series of complex processes, such as unloading the container from the quayside crane to the AGV, transporting it from the AGV to the container yard and grabbing the container from the yard crane to the corresponding container position, can be simulated. As shown in Figure 5, we considered an automated terminal that consists of 4 berths, 8 QCs, 12 AGVs, 8 container yards and 8 YCs. There are four ships waiting to be unloaded in one training session, and each ship has 96 container tasks for a total of 384 container tasks.

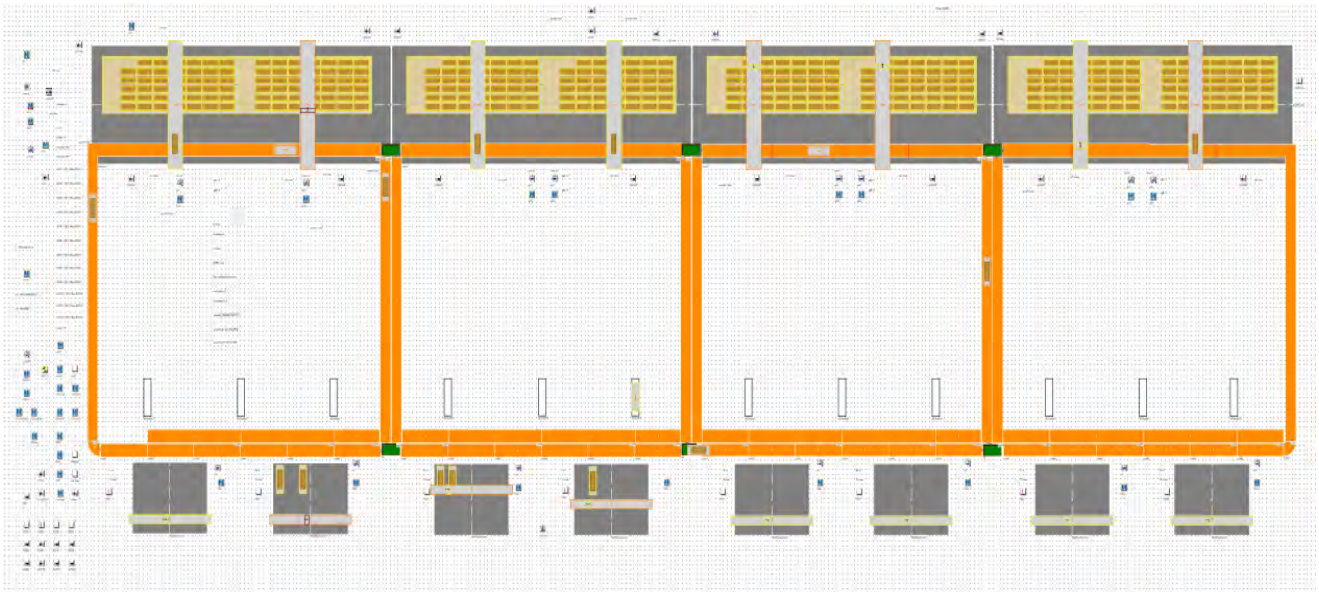


Figure 5. Schematic of the automated container terminal considered in the case studies.

In order to properly take into account the practical operation processes and dynamics involved in an automated container terminal, we adopted the following assumptions:

(1) All pending containers are standardized 20 ft TEUs;
(2) The problem of turning over the box is not considered;
(3) The information of the container task is randomly generated;
(4) The order of unloading of the QCs is predetermined;
(5) All containers are import containers;
(6) All AGV roads in the simulation scene are one-way streets;
(7) Each AGV can only transport one container at a time;
(8) When the AGV transports the container to the designated yard, it needs to wait for the YC to unload the container it carries for the task to be considered "completed";
(9) When the AGV completes its current task, and there are currently no incoming tasks, it will return to the AGV waiting area;
(10) All AGVs' navigation methods are the shortest path policy. Each container has its own key attributes, including container ID, the container from QC, container destination, container yard, the current waiting time for the AGV and the estimated distance to be transported.

5.2. Implementation of DQN-Based Multi-AGV Dynamic Scheduling

The proposed AGV dynamic scheduling algorithm based on DQN is programmed in TensorFlow, which is integrated with the simulation platform through a socket module, as shown in Figure 6. The simulation program includes several subprograms, namely, the terminal operation logic subprogram, the state calculation subprogram, the communication subprogram and the scheduling subprogram, to provide specific functions. The state calculation subroutine is used to calculate the current system state. The communication subroutine is responsible for transmitting the status information to the DQN module and receiving the action information transmitted by the DQN module, including the selected AGV and the scheduling rules. When the scheduling is triggered, the dynamic information of the current task and AGV will be sent to the state subroutine to calculate the current system state $s_t = (N_t, T_{awt}, D_{adt}, A_{st}, A_{ix}, A_{iy}, \ldots)$ and will be sent to the communicator after processing the subprograms. The communicator establishes a network connection with the DQN program through the TCP/IP protocol and sends the status record to the DQN program. The optimal combined action is calculated by the main Q-network, and the output is sent back to the scheduling subprograms to be parsed into the corresponding

scheduling rules and AGV indices. After the AGVs finish the tasks based on the outputs, the corresponding information of the tasks will be sent back to the DQN program and stored in the replay memory.

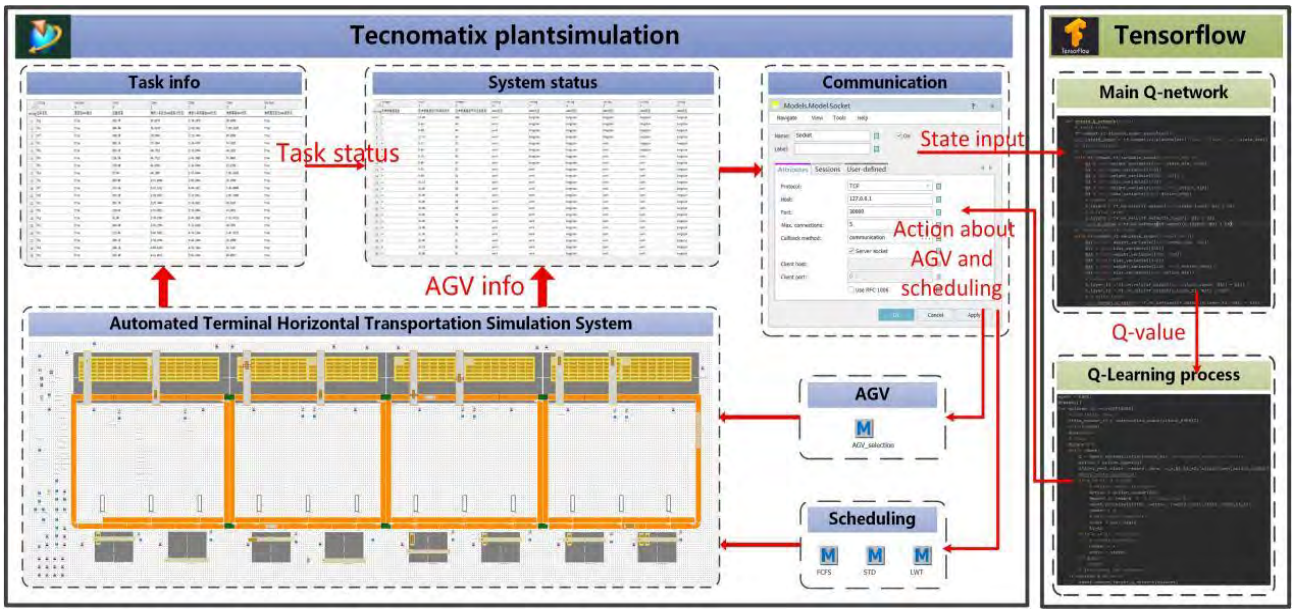


Figure 6. Dynamic scheduling of automated terminal AGVs based on DQN.

As described in Section 3.1, two fully connected neural networks are generated based on the simulation scenario established in Section 4.1. For each of the Q-networks, there are two hidden layers, with 350 nodes in the first layer and 140 nodes in the second layer, which are determined by previous simulation analysis. As there are 12 AGVs and 3 scheduling rules in the simulation scenario, according to the definitions of states and actions, there are 28 nodes in the input layer and 36 nodes in the output layer. The specific settings of the main Q-network and target Q-network are shown in Table 2.

Table 2. Network parameter settings.

Layer	Number of Nodes	Activation Function	Description
Input layer	28	None	Used to accept system state s_t
hidden layer 1	350	ReLU	None
hidden layer 2	140	ReLU	None
Output layer	36	Softmax	Value function $Q(s, a)$ for output action

By taking a process in one episode of training as an example, the training process is intuitively illustrated in Figure 6. Suppose the current task is as shown in Table 3, and the scheduling is triggered at this time. AGV1, AGV2, AGV4, AGV8 and AGV11 are in the working state, and AGV3, AGV5, AGV6, AGV7, AGV9, AGV10 and AGV12 are in the idle state at the same time. The positions of some AGVs, such as AGV1, AGV2 and AGV3, are AGV1 (20.0, 34.0), AGV2 (53.0, 41.0) and AGV3 (39.0, 13.0), respectively. Therefore, according to the definition of the system state in Section 2, the current state s_t = (5, 194.0, 258.68, 1920.0, 20.0, 34.0, 53.0, 41.0, 39.0, 13.0 . . .).

After inputting this state into the main Q-network, the action value function value Q-value is obtained, and the action is determined to be (2, 9) according to the selection policy; that is, the 9th AGV is used to perform STD and use transport container No. 004. When the action is completed, the reward function value R_t = 2.41, and the next state s_{t+1} = (3, 191.0, 129.0, 3872.0, 66.0, 39.0, 53.0, 41.0, 15.0, 33.0...); therefore, the complete record of the scheduling process Re can be obtained, Re = ((5, 194.0, 258.68, 1920.0, 20.0,

34.0, 53.0, 41.0, 39.0, 13.0 . . .), (2, 9), 2.41, (5, 194.0, 258.68, 1920.0, 20.0, 34.0, 53.0, 41.0, 39.0, 13.0 . . .)), and then stored in the replay memory for the main Q-network to perform training. The training of DQN takes 5000 times to learn the optimal mixed rule policy for the dynamic scheduling of the AGVs. The training process is conducted on a server with an i7-6700 CPU @ 3.40 GHz and with 16 G RAM. The training result is shown in Figure 7:

Table 3. Container task information.

ID	From	Destination	Waiting Time	Transport Distance
001	QC1	container yard6	55 s	284.54
002	QC4	container yard2	186 s	247.0
003	QC3	container yard7	255 s	278.9
004	QC5	container yard4	78 s	203.6
005	QC8	container yard3	396 s	279.4

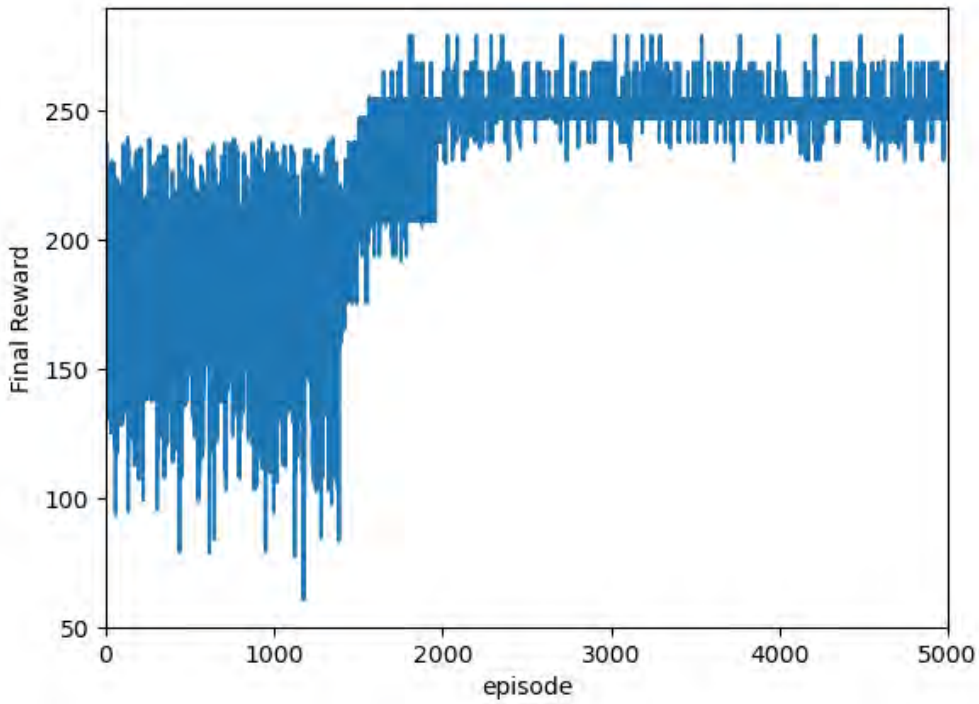


Figure 7. Final reward changing process during training.

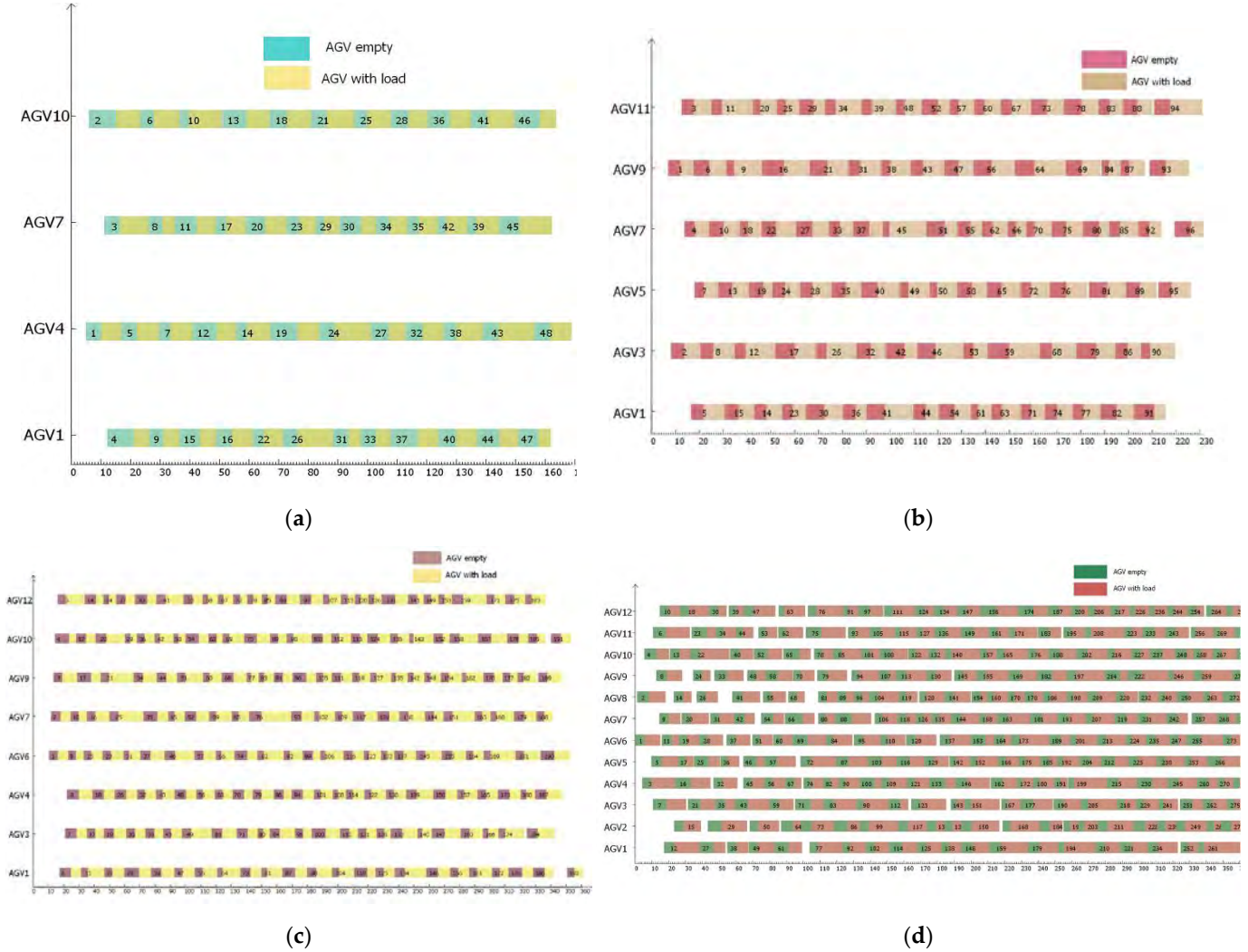
Figure 7 shows that the total reward value obtained during the entire scheduling process in the early stage of training is relatively random, the total reward value begins to converge after approximately 1500 training iterations, and the training results tend to stabilize after approximately 2000 iterations.

6. Results and Discussions

6.1. Testing Case Description

After the training is completed, the performance of the neural network trained by DQN is tested through the scheduling procedure described in Section 3.2. First, we consider the cases of four different problem sizes that are determined by the number of container tasks N, and the number of AGVs A: (1) 4 AGVs and 48 container tasks; (2) 6 AGVs and 96 container tasks; (3) 8 AGVs and 192 container tasks; and (4) 12 AGVs and 384 container tasks to analyze the experimental results. The Gantt of the four case scheduling by DQN is shown in Figure 8. Then, the total waiting time of the QCs, the total completion time of the AGVs and the computational efficiency are used as the performance indicators. In order to verify the superiority of the DQN-based mixed-rule scheduling approach, this section first designs

a genetic algorithm (GA) for the automated terminal AGV scheduling global optimization solution, comparing the results with the proposed DQN-based dynamic scheduling method in different task sizes and the AGV numbers, as detailed in Section 5.1. Finally, the results of the DQN-based mixed-rule scheduling approach and the three scheduling rules of FCFS, STD and LWT were compared, as described in Section 5.2.



(**a**) (**b**)

(**c**) (**d**)

Figure 8. AGV scheduling Gantt chart. (**a**) 48 × 4, (**b**) 96 × 6, (**c**) 192 × 8, (**d**) 384 × 12.

6.2. Results Analysis

6.2.1. Comparison of Results Based on DQN and Genetic Algorithm (GA)

In order to show the effectiveness of the proposed DQN-based approach in solving dynamic scheduling of AGVs in automated terminals, we designed a genetic algorithm (GA) to solve the static version of the scheduling problem using the testing data. We solved the dynamic scheduling problem by taking random features, such as the arrival time of ships and the travel speed of the AGVs, into consideration while using the DQN-based approach. The random features that are accomplished in the simulation environment and the relevant data are considered deterministic instances while solving the scheduling problem of AGVs by GA. We compared the results of the GA to those of the DQN-based approach, including the completion time of AGVs, the waiting time of QCs and the makespan of the system, respectively.

In order to integrate with the simulation case in 4.1, the implementation of the present-designed GA chromosome code in Python and the fitness values were calculated from the simulation program. We designed the GA chromosome in the form of real code; the

chromosome length is equal to m (total number of container tasks), the position of the chromosome in the container task number and the gene value is the AGV number, which is the container for which the AGV performs. The chromosome form is shown in Figure 9, where the sequential gene values of 4,5,8,1,2,4 indicate that container tasks numbered 1 to 6 are performed by the AGVs numbered 4,5,8,1,2,4, respectively.

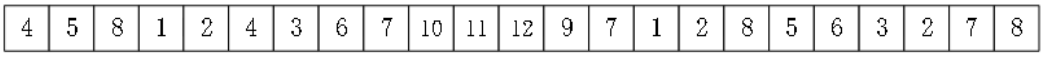


Figure 9. Chromosomal form.

Detailed steps of GA:

Step 1: Random numbers of 1 to 12 were generated at each gene locus;

Step 2: Python sends the generated chromosomes to the simulation program via the TCP/IP protocol;

Step 3: The simulation program schedules the AGV according to the chromosome information. The scheduling process is when the QC spreader grabs a new container task and finds the corresponding AGV number in the chromosome. According to the container task number, it is sent to the corresponding AGV task sequence. The AGV performs the corresponding schedules by its own sequence of tasks. When all container tasks have been performed, the objective function is calculated and converted to fitness values.

Step 4: The simulation program sends the calculated fitness values to the GA of Python through the TCP/IP protocol;

Step 5: When the fitness of all chromosomes is calculated, the selection, crossover and variation are processed;

Step 6: After generating a new population, we switch to Step 2;

Step 7: The optimal individual fitness is calculated after completing the iteration.

In the following analysis, the scheduling results of the DQN algorithm are compared with those generated by the traditional genetic algorithm (GA), and more comparisons are made based on the instances of four different scales described at the beginning of this section. As the results show in Table 4, the total completion time of the AGVs and the total waiting time of the QCs are compared, as well as the maximum completion time of the system, i.e., the makespan.

Table 4. Comparison of the results of DQN and GA at different instance scales.

N × A	Total Waiting Time of QCs (min)		Total Completion Time of AGVs (min)		Makespan (min)	
	DQN	GA	DQN	GA	DQN	GA
48 × 4	142.89	144.84	103.74	96.93	28.14	27.29
96 × 6	181.54	217.58	210.43	198.57	38.64	41.56
192 × 8	294.54	425.63	439.78	405.20	60.21	73.45
384 × 12	397.40	801.61	937.09	829.30	93.08	143.93

From the results of Table 4, for instances with different numbers of container tasks and AGVs, the proposed DQN algorithm could always sharply reduce the waiting time of the QCs compared to the GA. Even though the total completion time of the AGVs generated by the dynamic scheduling approach based on DQN is slightly worse than those generated by the GA, the final makespan appears to be improved for each of the instances.

Computational efficiency is extremely important for the AGV dynamic scheduling to meet the stringent operational requirements for an automated container terminal, especially when the size of the problem increases. In the following analysis, we documented the computational time for the instances with the four different problem sizes mentioned. The computational performance of the GA and proposed DQN-based approach, along with that of the benchmark models, are given in Table 5 below:

Table 5. Comparison of the computational efficiency.

N × A	DQN	GA
48 × 4	3.24 s	20 min
96 × 6	7.95 s	42 min
192 × 8	10.95 s	69 min
384 × 12	21.02 s	135 min

As shown in Table 5, at the same scale, when compared to the DQN, the GA takes significantly longer to calculate. However, DQN has a good speed advantage compared with the response time of genetic algorithms in the simulation environment.

The above results imply that the GA algorithm tends to pursue global optimization based on the known information over the entire horizon, while the proposed dynamic scheduling approach based on DQN is capable of handling dynamic information and generating better solutions based on the waiting times of the QCs and the makespan. The decreases in the QC waiting time and the makespan could be the reason that the AGVs have to be occupied more during the dynamic decision-making process. It reveals that the DQN-based algorithm outperforms the GA in terms of making full use of scarce resources and improving the working efficiency of the automated terminal.

6.2.2. Dynamic Scheduling Comparison

Three dynamic scheduling rules, FCFS, STD and LWT, are considered in our DQN-based approach as the basis to generate an optimized mixed-rule policy for the dynamic scheduling of AGVs. In this section, the total waiting time of the QCs and the total completion time of the AGVs are compared while applying the proposed DQN-based approach and applying each of the three rules independently.

First, we considered a case with 384 container tasks and 12 AGVs with odd indices, in the above automated terminal simulation scenario, as an example. A total of 384 container tasks were randomly generated in the four different scenes and performed by 12 AGV sets. Twenty episodes for each scene were carried out, and the results of the average performance of all the methods are shown. The comparison between the total waiting time of the QCs and the total completion time of the AGVs is shown in Figure 10.

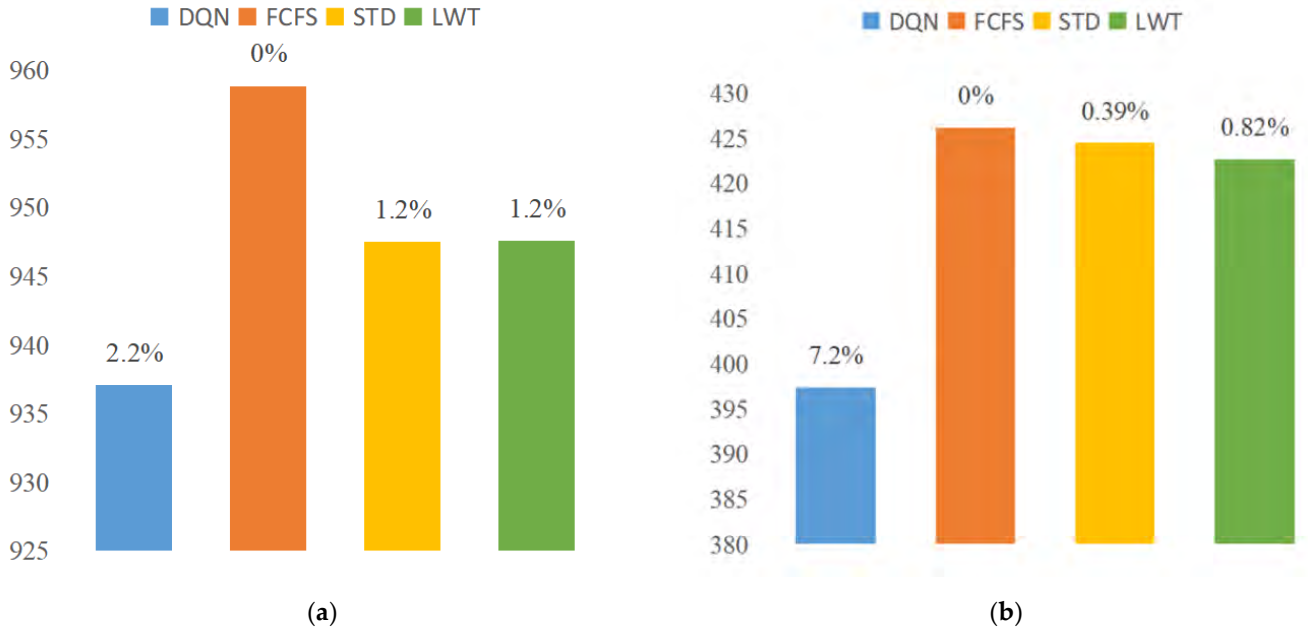


Figure 10. Results comparison: (**a**) improvements in total completion time of AGVs; (**b**) improvements in total waiting time of QCs.

As shown in Figure 10, the results generated by the proposed approach based on the DQN algorithm outperform those approaches applying the individual scheduling rule. The maximum improvements are 2.2% in terms of the total completion time of the AGVs and 7.2% in terms of the total waiting time of the QCs.

The scheduling results of the DQN algorithm are compared with those individual scheduling rules in 5.1 using the same four sets of instances with different scales. By solving the instances by the proposed DQN-based algorithm and the individual scheduling rules, the total waiting time of the QCs is shown in Table 6, and the total completion time of the AGV is shown in Table 7.

Table 6. Total waiting time of QCs (min) under different scales.

N × A	DQN	FCFS	STD	LWT
48 × 4	142.891	162.609	138.670	128.690
96 × 6	181.545	204.300	189.323	207.723
192 × 8	294.548	350.697	325.924	318.611
384 × 12	397.404	426.185	424.501	422.681

Table 7. Total completion time of AGVs (min) under different scales.

N × A	DQN	FCFS	STD	LWT
48 × 4	103.748	107.570	111.710	110.347
96 × 6	210.433	219.130	217.960	211.420
192 × 8	439.785	441.478	442.928	442.493
384 × 12	937.090	958.891	947.500	947.610

From the comparison of the results in Tables 6 and 7, it can be seen that the dynamic scheduling performance of our DQN-based approach with a mixed-rule policy can always outperform the scheduling results applying individual rules under different input scales. The results show that a better performance can be achieved only by selecting the most appropriate scheduling rules according to the different situations, and the computational performance of the proposed approaches, along with that of the benchmark models, are given in Table 8 below:

Table 8. Comparison of the computational efficiency.

N × A	DQN	FCFS	STD	LWT
48 × 4	3.24 s	1.11 s	1.15 s	1.10 s
96 × 6	7.95 s	2.08 s	2.29 s	2.05 s
192 × 8	10.95 s	4.05 s	4.16 s	4.03 s
384 × 12	21.02 s	8.06 s	8.28 s	7.91 s

As shown in Table 8, the CPU processing time of DQN is slightly higher than that of FCFS, STD and LWT in the different task sizes and AGV sizes, but it is still within a relatively acceptable range for dynamic scheduling because the mixed scheduling rules based on DQN have a high computing complexity. Therefore, the experimental results prove that the deep reinforcement learning DQN algorithm applied to horizontal transportation scheduling can greatly improve the horizontal transportation efficiency of the automated terminals.

Based on the above analysis, compared with the traditional global optimization algorithm, GA, although the GA can approximate the optimal results infinitely, due to the disadvantages of dynamic performance and the operational efficiency of the formal GA, DQN has relatively significant advantages in dynamic performance and operational efficiency. Compared to the three dynamic scheduling rules, the appropriate scheduling rules can be selected according to the different states, which can achieve better overall performance.

7. Conclusions and Future Directions

In this paper, a DQN-based dynamic scheduling approach is proposed to optimally schedule the horizontal transportation of AGVs in automated terminals. We first cast the scheduling problem as a Markov decision process (MDP) that is composed of the system state, action space and reward function. Then, we used the Tecnomatix simulation platform to generate the data for model training, based on which the optimal AGV scheduling policies can be determined under different situations. Finally, we compared the proposed approach with conventional optimization-based approaches, including GA and three scheduling rules. In four cases with different task numbers and AGV number sizes, DQN does not have a particularly obvious advantage in small-scale cases compared with the rule-based scheduling and GA; however, as the number of tasks and AGV numbers increases, the optimization performance of the DQN-based mixed scheduling rules becomes increasingly obvious. The comparison results show that, compared to rule-based scheduling and the GA, the DQN-based approach has a better optimization performance. In terms of computational efficiency, DQN runs slightly slower than rule-based scheduling due to the complexity of its own calculations but has a significant advantage over GA. With the increasing number of tasks and AGVs, the advantage of the DQN-based AGV dynamic scheduling in computational efficiency is more significant than that of the GA.

Future studies can be conducted in the following ways: The DQN-based dynamic decision-making method can be extended to include AGV collision avoidance and dynamic path planning to further improve the effectiveness of scheduling. More dynamic uncertainty features of the terminal can be included to improve the proposed DRL approach. A more complex problem where the import and export containers are handled in mixed operations could be considered in future studies.

Author Contributions: Conceptualization, X.Z., C.L. and Y.W.; methodology, X.Z., C.L., Y.W., J.S. and G.L.; software, X.Z.; writing—original draft preparation, C.L., Y.W. and J.S.; writing—review and editing, X.Z., Y.W. and G.L.; project administration. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key Research and Development Program of China (No.2019YFB1704403), the National Natural Science Foundation of China (No.71972128), the Soft Science Research Project of National Natural Science Foundation of Shanghai Science and Technology Innovation Action Plan (No.22692111200) and the Shanghai Sailing Program (21YF1416400).

Acknowledgments: The authors are grateful to the editors and the anonymous reviewers for the numerous valuable suggestions and comments.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Wu, Y.; Li, W.; Petering, M.E.H.; Goh, M.; Souza, R.D. Scheduling Multiple Yard Cranes with Crane Interference and Safety Distance Requirement. *Transp. Sci.* **2015**, *49*, 990–1005. [CrossRef]
2. Chen, X.; He, S.; Zhang, Y.; Tong, L.; Shang, P.; Zhou, X. Yard crane and AGV scheduling in automated container terminal: A multi-robot task allocation framework. *Transp. Res. Part C Emerg. Technol.* **2020**, *114*, 241–271. [CrossRef]
3. Yang, Y.; Zhong, M.; Dessouky, Y.; Postolache, O. An integrated scheduling method for AGV routing in automated container terminals. *Comput. Ind. Eng.* **2018**, *126*, 482–493. [CrossRef]
4. Xu, Y.; Qi, L.; Luan, W.; Guo, X.; Ma, H. Load-In-Load-Out AGV Route Planning in Automatic Container Terminal. *IEEE Access* **2020**, *8*, 157081–157088. [CrossRef]
5. Zhong, M.; Yang, Y.; Dessouky, Y.; Postolache, O. Multi-AGV scheduling for conflict-free path planning in automated container terminals. *Comput. Ind. Eng.* **2020**, *142*, 106371. [CrossRef]
6. Zhang, Q.L.; Hu, W.X.; Duan, J.G.; Qin, J.Y. Cooperative Scheduling of AGV and ASC in Automation Container Terminal Relay Operation Mode. *Math. Probl. Eng.* **2021**, *2021*, 5764012. [CrossRef]
7. Klein, C.M.; Kim, J. AGV dispatching. *Int. J. Prod. Res.* **1996**, *34*, 95–110. [CrossRef]
8. Sabuncuoglu, I. A study of scheduling rules of flexible manufacturing systems: A simulation approach. *Int. J. Prod. Res.* **1998**, *36*, 527–546. [CrossRef]
9. Shiue, Y.-R.; Lee, K.-C.; Su, C.-T. Real-time scheduling for a smart factory using a reinforcement learning approach. *Comput. Ind. Eng.* **2018**, *125*, 604–614. [CrossRef]

10. Angeloudis, P.; Bell, M.G.H. An uncertainty-aware AGV assignment algorithm for automated container terminals. *Transp. Res. Part E Logist. Transp. Rev.* **2010**, *46*, 354–366. [CrossRef]
11. Gawrilow, E.; Klimm, M.; Möhring, R.H.; Stenzel, B. Conflict-free vehicle routing. *EURO J. Transp. Logist.* **2012**, *1*, 87–111. [CrossRef]
12. Cai, B.; Huang, S.; Liu, D.; Dissanayake, G. Rescheduling policies for large-scale task allocation of autonomous straddle carriers under uncertainty at automated container terminals. *Robot. Auton. Syst.* **2014**, *62*, 506–514. [CrossRef]
13. Clausen, C.; Wechsler, H. Quad-Q-learning. *IEEE Trans. Neural Netw.* **2000**, *11*, 279–294. [CrossRef] [PubMed]
14. Jang, B.; Kim, M.; Harerimana, G.; Kim, J.W. Q-Learning Algorithms: A Comprehensive Classification and Applications. *IEEE Access* **2019**, *7*, 133653–133667. [CrossRef]
15. Tang, H.; Wang, A.; Xue, F.; Yang, J.; Cao, Y. A Novel Hierarchical Soft Actor-Critic Algorithm for Multi-Logistics Robots Task Allocation. *IEEE Access* **2021**, *9*, 42568–42582. [CrossRef]
16. Watanabe, M.; Furukawa, M.; Kakazu, Y. Intelligent AGV driving toward an autonomous decentralized manufacturing system [Article; Proceedings Paper]. *Robot. Comput.-Integr. Manuf.* **2001**, *17*, 57–64. [CrossRef]
17. Xia, Y.; Wu, L.; Wang, Z.; Zheng, X.; Jin, J. Cluster-Enabled Cooperative Scheduling Based on Reinforcement Learning for High-Mobility Vehicular Networks. *IEEE Trans. Veh. Technol.* **2020**, *69*, 12664–12678. [CrossRef]
18. Kim, D.; Lee, T.; Kim, S.; Lee, B.; Youn, H.Y. Adaptive packet scheduling in IoT environment based on Q-learning. *J. Ambient Intell. Hum. Comput.* **2020**, *11*, 2225–2235. [CrossRef]
19. Fotuhi, F.; Huynh, N.; Vidal, J.M.; Xie, Y. Modeling yard crane operators as reinforcement learning agents. *Res. Transp. Econ.* **2013**, *42*, 3–12. [CrossRef]
20. de León, A.D.; Lalla-Ruiz, E.; Melián-Batista, B.; Marcos Moreno-Vega, J. A Machine Learning-based system for berth scheduling at bulk terminals. *Expert Syst. Appl.* **2017**, *87*, 170–182. [CrossRef]
21. Jeon, S.M.; Kim, K.H.; Kopfer, H. Routing automated guided vehicles in container terminals through the Q-learning technique. *Logist. Res.* **2010**, *3*, 19–27. [CrossRef]
22. Choe, R.; Kim, J.; Ryu, K.R. Online preference learning for adaptive dispatching of AGVs in an automated container terminal. *Appl. Soft Comput.* **2016**, *38*, 647–660. [CrossRef]
23. Wan, Z.; Li, H.; He, H.; Prokhorov, D. Model-Free Real-Time EV Charging Scheduling Based on Deep Reinforcement Learning. *IEEE Trans. Smart Grid* **2019**, *10*, 5246–5257. [CrossRef]
24. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef]
25. Han, X.; He, H.; Wu, J.; Peng, J.; Li, Y. Energy management based on reinforcement learning with double deep Q-learning for a hybrid electric tracked vehicle. *Appl. Energy* **2019**, *254*, 113708. [CrossRef]
26. Buşoniu, L.; de Bruin, T.; Tolić, D.; Kober, J.; Palunko, I. Reinforcement learning for control: Performance, stability, and deep approximators. *Annu. Rev. Control* **2018**, *46*, 8–28. [CrossRef]
27. Kubalik, J.; Derner, E.; Zegklitz, J.; Babuska, R. Symbolic Regression Methods for Reinforcement Learning. *IEEE Access* **2021**, *9*, 139697–139711. [CrossRef]
28. Montague, P.R. Reinforcement learning: An introduction. *Trends Cogn. Sci.* **1999**, *3*, 360. [CrossRef]
29. Pan, J.; Wang, X.; Cheng, Y.; Yu, Q.; Jie, P.; Xuesong, W.; Wang, X. Multisource Transfer Double DQN Based on Actor Learning. *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *29*, 2227–2238. [CrossRef]
30. Stelzer, A.; Hirschmüller, H.; Görner, M. Stereo-vision-based navigation of a six-legged walking robot in unknown rough terrain. *Int. J. Robot. Res.* **2012**, *31*, 381–402. [CrossRef]
31. Zheng, J.F.; Mao, S.R.; Wu, Z.Y.; Kong, P.C.; Qiang, H. Improved Path Planning for Indoor Patrol Robot Based on Deep Reinforcement Learning. *Symmetry* **2022**, *14*, 132. [CrossRef]
32. Liu, B.; Liang, Y. Optimal function approximation with ReLU neural networks. *Neurocomputing* **2021**, *435*, 216–227. [CrossRef]
33. Wang, B.; Osher, S.J. Graph interpolating activation improves both natural and robust accuracies in data-efficient deep learning. *Eur. J. Appl. Math.* **2021**, *32*, 540–569. [CrossRef]
34. da Motta Salles Barreto, A.; Anderson, C.W. Restricted gradient-descent algorithm for value-function approximation in reinforcement learning. *Artif. Intell.* **2008**, *172*, 454–482. [CrossRef]