

## Article

# Deep Reinforcement Learning for Dynamic Twin Automated Stacking Cranes Scheduling Problem

Xin Jin, Nan Mi, Wen Song  and Qiqiang Li \*

Institute of Marine Science and Technology, Shandong University, Qingdao 266237, China;  
201820627@mail.sdu.edu.cn (X.J.); 201916207@mail.sdu.edu.cn (N.M.)

\* Correspondence: wensong@email.sdu.edu.cn (W.S.); qqli@sdu.edu.cn (Q.L.)

**Abstract:** Effective dynamic scheduling of twin Automated Stacking Cranes (ASCs) is essential for improving the efficiency of automated storage yards. While Deep Reinforcement Learning (DRL) has shown promise in a variety of scheduling problems, the dynamic twin ASCs scheduling problem is challenging owing to its unique attributes, including the dynamic arrival of containers, sequence-dependent setup and potential ASC interference. A novel DRL method is proposed in this paper to minimize the ASC run time and traffic congestion in the yard. Considering the information interference from ineligible containers, dynamic masked self-attention (DMA) is designed to capture the location-related relationship between containers. Additionally, we propose local information complementary attention (LICA) to supplement congestion-related information for decision making. The embeddings grasped by the LICA-DMA neural architecture can effectively represent the system state. Extensive experiments show that the agent can learn high-quality scheduling policies. Compared with rule-based heuristics, the learned policies have significantly better performance with reasonable time costs. The policies also exhibit impressive generalization ability in unseen scenarios with various scales or distributions.

**Keywords:** automated stacking crane (ASC); dynamic scheduling; self-attention; deep reinforcement learning (DRL)



**Citation:** Jin, X.; Mi, N.; Song, W.; Li, Q. Deep Reinforcement Learning for Dynamic Twin Automated Stacking Cranes Scheduling Problem. *Electronics* **2023**, *12*, 3288. <https://doi.org/10.3390/electronics12153288>

Academic Editors: Tianfei Zhou, Xiankai Lu and Wenguan Wang

Received: 6 July 2023

Revised: 26 July 2023

Accepted: 29 July 2023

Published: 31 July 2023



**Copyright:** © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

In smart cities, intelligent transportation systems (ITSs) are highly anticipated to improve transportation efficiency and promote sustainable transportation development [1]. For port cities, the rapid growth in container logistics has brought great challenges to the throughput and operation efficiency of container terminals. With the advancement of the Internet of Things (IoT), Automated Container Terminals (ACTs) have emerged as a popular way to address the aforementioned issue. ACTs have not only become a research hot topic in academia [2] but have also been implemented in several ports, such as Qingdao Port and Yangshan port in Shanghai. As the starting point of the loading operation and the end point of the unloading operation, the storage yard is critical to the handling efficiency of ACTs [3]. Automated Stacking Cranes (ASCs) are the most important equipment in the yard. Therefore, effective ASC scheduling is of great importance for improving the terminal operational efficiency. Meanwhile, energy consumption can be reduced by optimizing the run time of ASCs.

An automated storage yard often consists of several blocks. Each block contains a number of bays for container storage and is equipped with two ASCs (called twin ASCs) to move containers. In this paper, we consider the scheduling of twin ASCs in one block. Most existing works consider twin ASCs scheduling as a static problem. However, in reality, the containers often arrive dynamically due to the uncertain arrival of vessels and the scheduling of previous operations before storage in the yard, such as the unloading sequence of quay cranes (QCs) [4] and dispatching of Automated Guided Vehicles (AGVs) [5].

Therefore, we consider a more realistic twin ASCs scheduling problem with dynamic container arrival, which brings the challenge of uncertainty. In such cases, the limited information with regard to containers is not enough for Mathematical Programming (MP) to construct the entire set of constraints, and the incomplete task information cannot support meta-heuristics in order to explore the entire operation process. In addition, considering the NP-hardness of the problem, none of the previous methods can effectively handle a dynamic environment, which requires a real-time response [6]. In reality, such dynamic scheduling problems are often solved by rule-based heuristics. However, existing scheduling rules are mostly designed based on human intuition; hence, the quality of solutions is severely limited by their greedy nature and the lack of comprehensive observation of system information.

Essentially, the dynamic twin ASCs scheduling problem can be considered as a variant of the Dynamic Job-shop Scheduling Problem (DJSP). The container handling tasks and ASCs can be regarded as the jobs and machines, respectively. The dynamic attribute is the dynamic arrival of import containers. Recently, motivated by its impressive advancements in Reinforcement Learning (RL) [7,8] and Deep Neural Networks (DNN) [9], DRL has been employed to solve NP-hard combinatorial optimization problems [10] such as the JSP and the traveling salesman problem (TSP) [11]. In DRL, the complex computation in traditional methods is displaced by elaborated DNN, which endows the agent with high-quality solutions and millisecond inference speed. Therefore, DRL has great potential in tackling dynamic scheduling problems. However, the dynamic twin ASCs scheduling problem is distinguished from standard JSP by its unique characteristics that cannot be handled by existing DRL-based methods. Firstly, before handling a container, the ASC must move to the position of the container from the end point of its previous handling. The time for such movement (known as setup time) depends on the handling sequence. Hence, location-related information is critical in minimizing the run time of ASCs. Secondly, the dynamic arrival of import containers could cause severe congestion in ACT. Hence, it is important to capture the congestion-related information by analyzing the sequential information of dynamic containers. As a result, the agent needs to learn the location-related information and congestion-related information—which have different semantics and require different neural structures—simultaneously. Such multi-aspect learning is challenging for existing DRL-based scheduling methods.

In this paper, we propose an end-to-end DRL to automatically learn effective heuristics for solving the dynamic twin ASCs scheduling problem. Compared with rule-based heuristics, the learned policies have noticeably better scheduling performance with similar time costs. Specifically, we first propose a Markov Decision Process (MDP) model to simulate the dynamic handling process of a block with twin ASCs. Based on the MDP model, a novel neural structure is proposed to map the observation to a scheduling policy. In particular, we propose the dynamic masked self-attention (DMA) mechanism to grasp location-related information between containers while dealing with information interference caused by ineligible containers. Further, we propose local information complementary attention (LICA) to extract the congestion-related information of import containers and fuse it with the location-related information. Finally, we design a size-agnostic policy network to calculate the distribution over action spaces of variable size. Benefiting from the size-agnostic property, the trained agent can be generalized to handle problems of varying scales (i.e., the number of containers). We train the proposed network using Proximal Policy Optimization (PPO) [12]. Extensive experiments show that the learned policies perform remarkably well on problems of various scales. Moreover, the learned policies exhibit strong generalization performance on instances with various scales or different distributions.

The main contributions of this study are listed as follows:

1. The DRL method is introduced to the dynamic twin ASCs scheduling problem for the first time. The millisecond decision time, outstanding scheduling performance and robust generalization capability are extremely promising for dynamic scheduling in real scenarios.
2. We propose a novel MDP model for the dynamic scheduling problem, which comprehensively simulates the dynamic handling process in the container block.
3. We propose a novel attention mechanism, DMA, which can automatically ignore information interference and grasp pairwise location relationships between containers.
4. We propose a novel attention-based framework, LICA, which complements the container feature embeddings extracted by DMA with congestion-related information.

The remainder of this paper proceeds as follows: Section 2 summarizes relevant works. The research problem is formally described in Section 3. Section 4 describes the proposed neural structure and DRL method in detail. The comparative experiment results are presented and discussed in Section 5. Lastly, Section 6 concludes the paper.

## 2. Literature Review

In this section, we briefly review the twin ASCs scheduling methods and DRL-based scheduling methods.

### 2.1. Twin ASCs Scheduling Approaches

The twin ASCs scheduling problem has drawn more attention recently due to its critical impact on ACT operational efficiency. Through handshake operations, the twin ASCs achieve effective collaborative container handling. To facilitate this collaboration, a handshake area is introduced. The effect of the handshake area on operation efficiency and the performance of scheduling rules are discussed by Gharehgozli et al. [13]. In order to address the issue of twin ASCs interference in a block, Carlo et al. [14] presented and tested 14 priority scheduling rules. To improve the efficiency of storing imported containers, Briskorn et al. [15] developed two meta-heuristics to minimize the makespan. Han et al. [16] comprehensively formulated twin ASCs handling processes and proposed MP and Genetic Algorithm (GA) to minimize the makespan. Oladugba et al. [17] reformulated the twin yard crane scheduling problem in [16] and proposed a new heuristic, a modified Johnson algorithm, to solve it.

More recently, several articles have attempted to consider dynamic properties of the ASCs scheduling problem. Jaehn and Kress et al. [18,19] investigated twin ASCs scheduling scenarios in which inbound containers arrive in a sequential order. For this problem, they propose several scheduling rules, a dynamic programming (DP) algorithm and a related beam search heuristic to minimize the dwell time of the vessel. For a similar problem, Lu et al. [20] propose a particle swarm optimization algorithm (PSO) based on a graph theory model to optimize ASC run time and the waiting time of AGVs. Zheng et al. [21] investigated the twin ASCs scheduling problem with dynamic processing time. They calculate the dynamic processing time for each task and use proposed heuristics and GA to minimize the largest tardiness. However, the dynamic attributes in these studies are deterministic and known, so they are still a type of static scheduling problem [22].

While cooperative scheduling of twin ASCs has been widely studied, none of the traditional methods can effectively address the dynamic twin ASCs scheduling problem. Considering the non-negligible computation time and the NP-hardness of the dynamic twin ASCs scheduling problem, MP methods and meta-heuristics have difficulty achieving high-quality solutions within the constrained time. simultaneously Rule-based heuristics may not produce satisfactory solutions due to their greedy nature and the lack of comprehensive observation of system information.

### 2.2. DRL Methods for Scheduling Problems

DRL-based methods are more promising than traditional methods in solving complex scheduling problems due to their low reliance on human experience and fast solving speed

(after training). Recently, DRL has been employed to solve various scheduling problems such as permutation flow-shop [23], hybrid flow-shop [24] and flexible job-shop [25]. In particular, JSP has drawn much attention in DRL research as a well-known and standardized scheduling problem. Lin et al. [26] propose a deep Q network to choose rules for addressing the JSP under the smart factory framework. To overcome the performance limitation caused by scheduling rules, Zhang et al. [27] propose an end-to-end DRL method that is based on Graph Neural Networks (GNNs) [28] to explore the space of dispatching rules for JSP. Park et al. [29] take a similar idea and propose a GNN-based DRL to solve JSP. Furthermore, they propose a multi-agent scheduler based on type-aware graph attention, ScheduleNet [30], to solve various types of multi-agent scheduling tasks, including routing problems and JSP.

As mentioned previously, the dynamic twin ASCs scheduling problem can be regarded as a DJSP with sequence-dependent setup times. Therefore, the above DRL methods are not applicable since they are designed for static problems. To overcome this issue, several DRL methods for DJSP have emerged recently. Zhao et al. [31] propose a Deep Q Network (DQN) for DJSP with dynamically arriving jobs. The trained agent selects dispatching rules to minimize the delay time. Wang et al. [32] and Zhang et al. [33] propose a DRL method and a multi-agent DRL method to realize the end-to-end solution for DJSP. However, the methods in [31–33] all adopt a multilayer perceptron (MLP) network to extract state features. Such a simple neural structure cannot grasp the location-related information and the congestion-related information from the raw features. As will be shown in the experiments, the MLP-based policy network performs poorly due to the insufficient feature extraction ability.

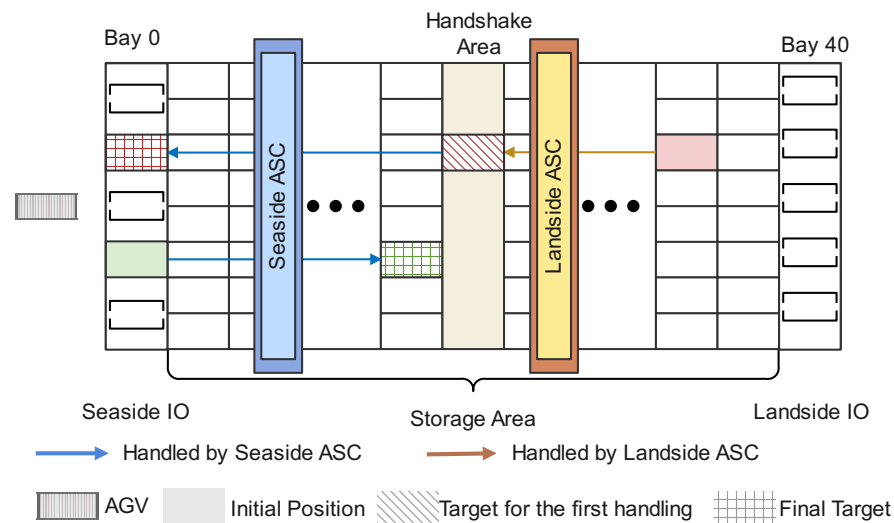
As discussed above, neither the traditional methods nor the existing DRL methods can effectively tackle the dynamic twin ASCs scheduling problem studied in this paper due to its complexity and unique properties, including the dynamic arrival of containers and the sequence-dependent setup times.

### 3. Problem Description

In this section, we elaborate on the dynamic twin ASCs scheduling problem with the dynamic arrival of import containers.

In ACT, the automated yard consists of a number of blocks perpendicular to the quay. We consider the scheduling of twin ASCs for one block, the layout of which is demonstrated in Figure 1. The block consists of a number of bays  $B = \{0, 1, \dots, \beta\}$ , which are numbered in order from quay to land. Bay 0 and  $\beta$  are called the seaside IO and landside IO, respectively, and can be regarded as a buffer with a certain capacity. The seaside IO is responsible for the container shift between the block and the AGV, and the landside IO is employed to shift containers between the block and external trucks. Bays  $1, \dots, \beta - 1$  constitute the space for storing and stacking containers and are called the storage area. There are two ASCs, called twin ASCs, for container handling in the block. In a container handling process, the travel distance of ASCs in the row direction is generally much smaller than that in the bay direction. Furthermore, the motion in both directions can be performed simultaneously. Without loss of generality, we only consider the bay-side movement of the ASC. Twin ASCs are more efficient with a balanced workload on the seaside and landside [34]. Consequently, we set the bay that is the average of all container handling midpoints as the handshake area to promote workload balance. Seaside ASC can transport containers between the seaside IO and the handshake area. The work range of the landside ASC is situated between the handshake area and the landside IO. Since the two ASCs share a common track, they cannot overlap or cross each other. For ease of description, we only consider the interference situation with non-crossing and non-overlapping constraints. Note that safety distance constraints can be incorporated into the presented methods [18,19].

In actual ACT, to facilitate import/export management, container transportation between blocks and vessels and between blocks and external trucks cannot be performed simultaneously [35]. Here, we only consider the container movement between blocks and vessels. When a vessel enters the terminal, a group of import containers needs to be transported by AGVs from the quay to the seaside IO for storage. Considering its limited capacity, if the seaside IO is already full, the AGV with import containers must wait for other containers in the seaside IO to be removed before unloading the containers. Simultaneously a group of export containers need to be transported from the seaside IO to the quay by empty AGVs. Figure 1 gives an example wherein the import container (green) needs to be moved into the block and the export container (red) needs to be moved to the seaside IO.



**Figure 1.** Layout of the block and process of container handling.

The container handling process is performed over a finite time horizon  $T = \{0, 1, \dots, \omega\}$ . The set of all containers is denoted as  $C = C_I \cup C_E$ , where  $C_I$  and  $C_E$  are the set of import and export containers, respectively, and the import containers in  $C_I$  transported by AGVs arrive dynamically at the seaside IO. For the convenience of formulation, the arrival of imported containers is defined as a Poisson process. Without loss of generality, we denote a container  $c \in C$  as a tuple  $\langle t_c^A, b_c^S, b_c^D \rangle$ , which comprises its arrival time  $t_c^A \in T$ , start position  $b_c^S \in B$ , and destination position  $b_c^D \in B$ . Specifically, for import containers  $c \in C_I$ ,  $t_c^A$  is the arrival time of the AGV with container  $c$ ,  $b_c^S = 0$ , and  $b_c^D \neq 0$ . Benefiting from the stability of AGV transport, once the container is loaded onto the AGV by the quay crane, the arrival time to the seaside IO can be expected (i.e., the arrival time  $t_c^A$  is observable in a time window before  $t_c^A$ ) [4,5]. Export containers  $c \in C_E$  are located in the block at the beginning, i.e.,  $t_c^A = 0$ ,  $b_c^S \neq 0$ , and  $b_c^D = 0$ .

As mentioned above, AGVs with import containers  $c \in C_I$  may wait at the seaside IO due to its congestion. This waiting time  $t_c^W$  is a variable determined by the twin ASCs scheduling results, which directly affects the seaside IO occupancy. As a result, any import container  $c \in C_I$  can only be handled by the ASC until it is unloaded in the seaside IO; thus, its earliest accessible time is  $t_c^{AC} = t_c^A + t_c^W$ . Until it is moved out the seaside IO by an ASC, it will occupy a position within the seaside IO. Since export containers do not need to wait, we have  $t_c^{AC} = t_c^A = 0, \forall c \in C_E$ . However, the export containers need to be handled to the seaside IO and remain there until there is an empty AGV to transport them away. Additionally, it is assumed that the arrival of empty AGVs also follows a Poisson process. Once the import container is handled into the seaside IO, the container will be removed at the earliest feasible time indicated in the arrival time list of empty AGVs.



For a container  $c$ , if  $b_c^S$  and  $b_c^D$  are on the same side of the handshake area (e.g., the green one in Figure 1), it needs to be handled once by the seaside ASC. Otherwise, it needs to be handled twice by different ASCs. The target position of the first handling is the handshake area, and the target position of the second handling is container's destination position  $b_c^D$  (e.g., the handling process of the red container in Figure 1). Here, we denote the seaside and landside ASC as  $ASC_0$  and  $ASC_1$ , respectively, and model each handling as an operation  $i \in O$ , where  $O = O_0 \cup O_1$ , with  $O_0$  and  $O_1$  being the operations of  $ASC_0$  and  $ASC_1$ , respectively. The operation to each container  $c$  cannot start before its earliest accessible time  $t_c^{AC}$  and is non-preemptive (i.e., it must be handled to the target position of the operation once started).

The processing time of each operation  $i$  consists of three parts: (1) fixed handling time, (2) picking up and dropping down time of ASC and (3) waiting time caused by possible ASC interference. The fixed handling time is the time of the ASC moving from its start position  $b_i^S$  to its target position  $b_i^T$ . From the above, the processing time  $p_i$  can be formulated as:

$$p_i = |b_i^T - b_i^S| / \eta + 2\tau, \forall i \in O, \quad (1)$$

where  $\eta$  is the velocity of the ASC, and  $\tau$  is the picking up and dropping down time of the ASC.

Before handling an operation  $i$ , the ASC must move from its current position (i.e., the target position  $b_j^T$  of the previous operation  $j$  for this ASC) to the location of the next container (i.e., the start position  $b_i^S$  of  $i$ ). This movement can be regarded as the setup before operation, and the corresponding setup time  $t_{ji}$  can be formulated as:

$$t_{ji} = |b_j^T - b_i^S| / \eta, \forall i, j \in O_k, k \in \{0, 1\}, \quad (2)$$

where  $j$  indexes the previous operation of  $i$  on the same ASC. At the beginning, the seaside ASC and the landside ASC are located at the seaside IO and the landside IO, respectively. Clearly, the setup time depends on the scheduling of the handling sequence.

During handling of operation  $i$ , if both ASCs need to pass through the handshake area at the same time, one ASC must wait outside (determined by a priority rule) to avoid ASC interference. The time of such waiting, denoted as  $t_i^W$ , is a variable related to the scheduling decisions.

In this article, we attempt to improve the service quality and economic benefits by minimizing the congestion and operation cost. To this end, we adopt the integrated objective in [20]:

$$\min : F = T_{wait} + T_{run}, \quad (3)$$

where  $T_{wait}$  and  $T_{run}$  are the total waiting times of AGVs with import containers and the cumulative run time of ASCs to complete all operations, respectively. They represent the congestion of AGVs in the block and the operating costs of ASCs, respectively. Concretely, they are formulated as:

$$T_{wait} = \sum_{c \in C_I} t_c^W, \quad T_{run} = \sum_{i \in O} (p_i + t_i^W + t_{ji}). \quad (4)$$

**Remark 1.** Abstractly, the import containers can be regarded as dynamically arriving jobs, while the export containers are static. One ASC (machine) can only process one job at one time and must move to the position of the next container before handling it (setup). In addition, the ASC must comply with the non-overlap constraint during all handling processes. To sum up, the scheduling problem studied in this article can be viewed as a DJSP problem with a sequence-dependent setup time and machine interference constraints.

## 4. Methodology

In this section, we introduce our method in detail. Firstly, the MDP model is established, followed by the definition of raw state features. Then, our neural architecture is introduced in detail, including the two novel design: DMA and LICA. Finally, the actor-critic-based training algorithm is presented.

### 4.1. Markov Decision Process

In this part, we formulate an event-driven MDP model to simulate the dynamic twin ASCs scheduling problem.

#### 4.1.1. State

The decisions are made at the initial time or when the ASC completes a handling operation. If there is no eligible container to handle for the current decision step, the environment transits to the next decision step. The ASC to be scheduled at  $t$  is denoted as  $ASC_t$ . The state at the decision step  $t$ , denoted as  $S_t$ , contains information about containers and the block at  $t$ , such as the destination bay of import containers, the arrival time of observable import containers, the origin position of export containers, the eligibility of all containers, the current position of ASCs, and the occupancy status of the seaside IO. Note that the eligible containers are the containers that can be handled by the ASC scheduled at step  $t$ . Specifically, these are containers that have not been handled to their destination and whose current operations are located in the scheduled ASC's work range. Here,  $ASC_t$  is the ASC that just completed its handling operation.

#### 4.1.2. Action

An action  $a_t \in A_t$  at step  $t$  is to select an eligible container for  $ASC_t$ , i.e., select a job for the machine. The action space  $A_t$  is the set of eligible containers. As the handling progresses, the eligible actions for each step  $t$  are different, both the manual rules and the DRL techniques must eliminate ineligible containers when choosing actions in order to ensure the validity of the action.

#### 4.1.3. Transition

Once the action  $a_t$  is taken, the selected container will be transported to the target bay, and the state  $S_t$  will transit to the next state  $S_{t+1}$ . In this process, the scheduled ASC moves to the current location of the selected container and transports the container to its target bay (i.e., the destination bay or the handshake area). Considering the possible ASC interference during the state transition, we apply the First-In-First-Out (FIFO) rule to prioritize ASCs so as to satisfy the non-overlap constraints. This enables resolving possible interference at a negligible cost.

#### 4.1.4. Reward

To minimize the objective (3), the reward function  $R(S_t, a_t)$  is designed as the negative increment value of AGV waiting time and work time for ASCs after taking action  $a_t$  at step  $t$ . The mathematical expression is as follows:

$$R(S_t, a_t) = T_{wait}(t+1) - T_{wait}(t) + \sum_{k \in \{0,1\}} [T_{run}^{ASC_k}(t+1) - T_{run}^{ASC_k}(t)], \quad (5)$$

where  $T_{wait}(t)$  is the total waiting time of AGVs with import containers at step  $t$ , and  $T_{run}^{ASC_k}(t)$  is the accumulated run time of twin ASCs at step  $t$ .

#### 4.1.5. Policy

For the state  $S_t$ , the stochastic policy  $\pi(a_t|S_t)$  is a probability distribution over the action space  $A_t$ . Due to the dynamic nature of the investigated problem, we use the greedy

strategy to select the action with the highest probability. For agent training, however, we sample actions according to  $\pi(a_t|S_t)$  for the purpose of exploration.

#### 4.2. Raw Features

To comprehensively represent the state  $S_t$ , we design a set of raw features to describe the states of all containers and resources (ASCs) in the block. For each container  $c \in C$ , we record the following information as raw features:

1. The current position of  $c$ , which is an integer specifying the index of its current bay;
2. The destination of  $c$ , which is an integer specifying the index of its destination bay;
3. The distance between container  $c$  and the current position of  $ASC_t$ ;
4. The eligibility of  $c$ , which is a binary value specifying if  $c$  is accessible for the scheduled ASC at  $t$  (0) or not (1);
5. The number of import containers in the seaside IO, which is an integer representing the occupancy of the seaside IO by imported containers;
6. The number of export containers in the seaside IO, which is an integer representing the occupancy of the seaside IO by exported containers;
7. The difference between the arrival time of  $c$  and the current time: for special cases, it is set to 0 if  $c$  has already arrived, or a large number  $\mathcal{M}$  if its arrival time is out of the observation time window.

Here, features (1)–(4) are location-related features, and features (5)–(7) are congestion-related features. Note that feature (7) is exclusive to import containers.

#### 4.3. Feature Extraction Network

While the above features are informative, efficient feature extraction from complex redundant raw features is extremely challenging for existing neural network structures. Firstly, the location relationship between the containers and ASCs is pair-wise information, which is subject to the information interference of ineligible containers. Secondly, the effective fusion of the extracted congestion-related embeddings and location-related embeddings is challenging for existing methods. To tackle the above problems, two novel neural architectures, DMA and LICA, are elaborated as follows.

##### 4.3.1. DMA

According to the problem description in Section 3, the characteristics of the ASC container handling process can be summarized as follows: Firstly, ASCs handle the containers in a one-dimensional space (i.e., travel between bays). Secondly, before handling the subsequent container, the ASCs must move from the location of their previous operation to the location of the subsequent container (i.e., setup). It is obvious that the pairwise position relationship between containers and ASCs is essential to optimize the run time of ASCs. Driven by this intuition, we attempt to introduce a self-attention mechanism [36] into the feature extraction network to grasp the pairwise location relationships. A self-attention mechanism has been employed to solve some combinatorial optimization problems, such as routing problems [37–40]. However, most of these techniques follow the encoder–decoder structure [41], which is not applicable for dealing with complex features that change constantly [42]. Therefore, we adopt a step-wise decision framework: i.e., the network extracts embeddings and makes decisions at each step instead of using a decoder. Additionally, the dynamic nature of the research problem makes the extraction of embeddings subject to the interference of redundant information (i.e., the features of ineligible containers). As will be shown in the experiments, directly applying the attention mechanism results in poor performance. To tackle this issue, we propose a novel attention mechanism, DMA, to eliminate the interference of redundant information. The details are introduced in combination with Figure 2 as follows.



In order to unify the model input dimensions, the location-related features are projected to a fixed dimension  $d_h = 128$  as the input feature matrix  $X$ . Subsequently, Query ( $Q$ ), Key ( $K$ ) and Value ( $V$ ) can be obtained by linear projections:

$$Q = X \times W_q, K = X \times W_k, V = X \times W_v, \quad (6)$$

where  $W_q \in \mathbb{R}^{d_h \times d_q}$ ,  $W_k \in \mathbb{R}^{d_h \times d_k}$  and  $W_v \in \mathbb{R}^{d_h \times d_v}$  are trainable parameters, and  $d_q = d_k = d_v = 64$  in our method.

Based on matrices  $Q$  and  $K$ , the compatibility matrix  $M \in \mathbb{R}^{N \times N}$  can be calculated by

$$M = (QK^\top) / \sqrt{d_k}, \quad (7)$$

where  $N = |C|$  is the number of containers, and  $\sqrt{d_k}$  is the scaling factor to regulate the attention weight after softmax.

In DMA, the compatibility matrix  $M$  is masked by a dynamic mask to eliminate the attention to the information of ineligible containers. Specifically, for each column in the compatibility matrix, the elements are identified according to the eligibility of containers at current state  $S_t$  and are masked out from the softmax by overwriting their score as follows:

$$M'(i, j) = \begin{cases} -\infty, & \text{if container } i \text{ is ineligible} \\ M(i, j), & \text{otherwise,} \end{cases} \quad (8)$$

where  $M(i, j)$  is the element in row  $i$  and column  $j$  of the compatibility matrix  $M$ .

Then, attention can be calculated by row-wise softmax, and the output can be obtained by the weighted summation of value vectors in  $V$ . The above process can be written as:

$$A(X) = \text{Softmax}(M')V. \quad (9)$$

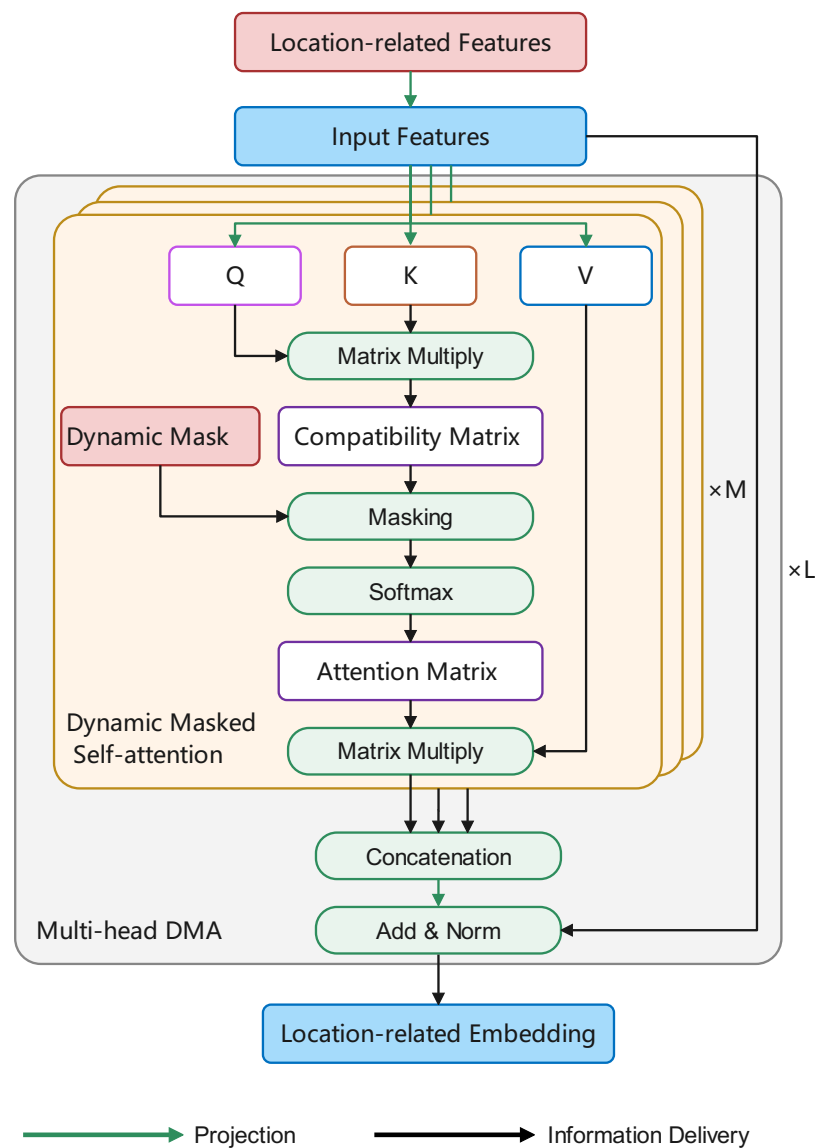
Inspired by multihead self-attention (MHA) [36], we propose multihead DMA to capture more features from different high-dimensional spaces. As shown in Figure 2,  $M$  DMA modules, which do not share parameters, are performed in parallel. Then, we fuse all the outputs by concatenation and project this back to the  $d_h$  dimensional space by a fully connected layer. Finally, the output embeddings of dimension  $N \times d_h$  are obtained by skip-connection and layer normalization. The above process can be formulated as follows:

$$\hat{h} = \text{Concat}(A_1(X), \dots, A_M(X)), \quad (10)$$

$$\text{DMA}(X) = \text{LN}(X + \text{FC}(\hat{h})), \quad (11)$$

where  $A_1(X), \dots, A_M(X)$  are the outputs of MDA modules,  $\text{Concat}(\cdot)$  is the horizontal concatenation operator,  $\text{FC}(\cdot)$  is a fully connected layer, and  $\text{LN}(\cdot)$  is the layer normalization operator. In our method, we stack  $L = 2$  multihead DMA layers to obtain the location-related embeddings  $E_l \in \mathbb{R}^{N \times d_h}$  so as to balance performance and computational efficiency.

**Remark 2.** Compared with the standard attention mechanism, the DMA can automatically eliminate the interference of redundant information (i.e., the features of ineligible containers). Benefiting from the multihead DMA, the feature extraction network can effectively learn to grasp the pairwise location-related relationships between containers, which is significant to minimize the run time of ASCs. Because the interference of redundant information is circumvented, the agent can converge to a better result in the training process. Furthermore, the multihead DMA has a size-invariant property with regard to the number of containers, which enables the network to work on problems of different scales using the same parameters.



**Figure 2.** Multihead dynamic masked self-attention (DMA).

#### 4.3.2. LICA

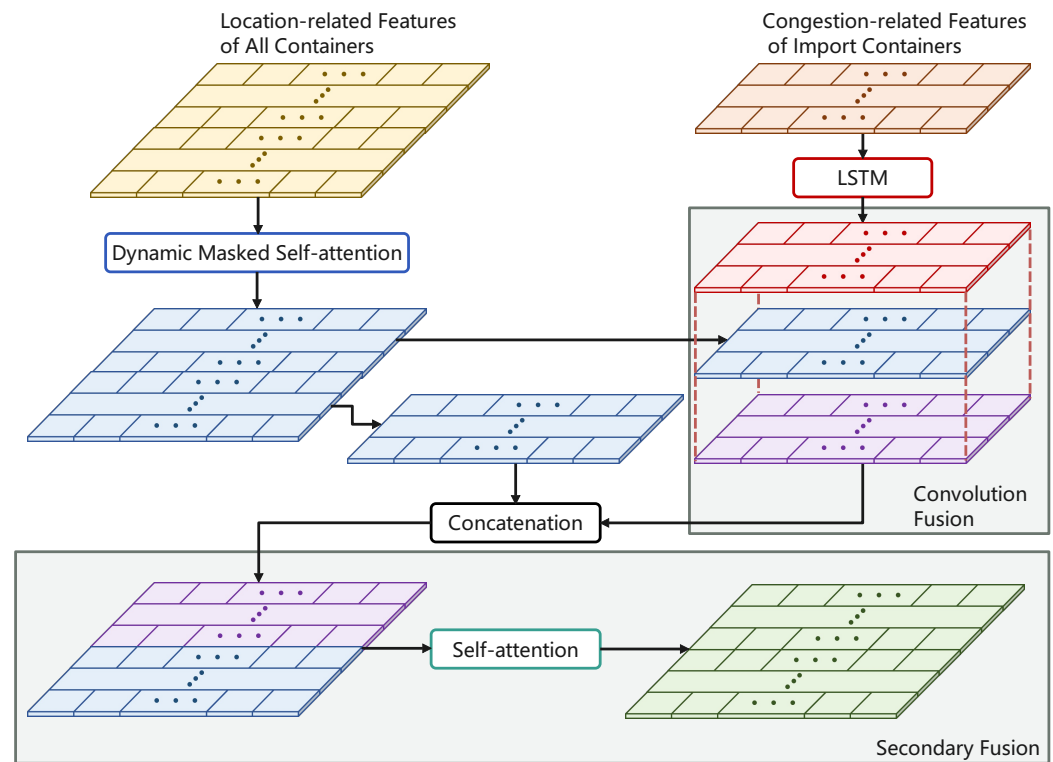
On the other hand, the congestion-related information is critical to optimizing the total waiting time of AGVs with import containers. Due to the different characteristics between location-related information and congestion-related information, utilizing one neural architecture to extract the embeddings of both kinds of information cannot achieve satisfactory performance. Considering the sequential property of congestion-related information, Long Short-Term Memory (LSTM) [43] is employed to extract the embeddings from congestion-related features. Feature (7) is exclusive to imported containers, and the congestion-related information mainly affects the priority of import containers. Therefore, the congestion-related embeddings  $E_c \in \mathbb{R}^{N_i \times d_h}$  are only captured for import containers, where  $N_i$  is the number of import containers. In order to effectively fuse two types of information with different semantics, we design a novel neural framework, LICA, of which the inputs are location-related embeddings captured by DMA and congestion-related embeddings grasped by LSTM. The details are as follows.

As illustrated in Figure 3, the location-related embeddings  $E_l \in \mathbb{R}^{N \times d_h}$  obtained by DMA (i.e., the blue one in Figure 3) and congestion-related embeddings  $E_c \in \mathbb{R}^{N_i \times d_h}$  calculated by LSTM (i.e., the red one in Figure 3) have different dimensions. In LICA,

the location-related embeddings are split into two pieces: location-related embeddings of import containers  $E_{li} \in \mathbb{R}^{N_i \times d_h}$  and location-related embeddings of export containers  $E_{le} \in \mathbb{R}^{N_e \times d_h}$ , where  $N_e$  is the number of export containers. Then, the location-related embeddings of import containers and the congestion-related embeddings are fused by a convolution fusion module. The first step is stacking them into a tensor  $\mathcal{T}_{lc} \in \mathbb{R}^{N_i \times d_h \times 2}$ . Then,  $\mathcal{T}_{lc}$  is transformed into a new matrix by multiple point-wise convolution layers as the fusion embedding of import containers:

$$E_{fi} = PwConv(\mathcal{T}_{lc}), \quad (12)$$

where  $E_{fi} \in \mathbb{R}^{N_i \times d_h}$ , and  $PwConv(\cdot)$  is an operator representing multiple point-wise convolution layers and has two hidden layers with dimensions of 16 and 4, respectively.



**Figure 3.** Local information complementary attention (LICA) framework with DMA and LSTM.

Following the convolution fusion, the new embeddings of import containers  $E_{fi}$  include the location-related information and congestion-related information. To fuse the embeddings of import containers and export containers effectively, a secondary fusion module based on MHA is proposed. Specifically, we concatenate  $E_{fi}$  and  $E_{le}$  together, and a further fusion of them are performed by a standard MHA:

$$E_{fe} = MHA\left(Concat\left(E_{fi}, E_{le}\right)\right), \quad (13)$$

where  $E_{fe} \in \mathbb{R}^{N \times d_h}$  is the output of the LICA as well as the embeddings captured by the feature extraction network.

**Remark 3.** The proposed fusion framework, LICA, is capable of fusing embeddings with different dimensions. Benefiting from LICA's superior fusion capabilities, the location-related embedding is complemented by the congestion-related information. The embeddings  $E_{fe}$  captured by the feature extraction network can effectively represent the location-related relationship and the congestion state of the block. Moreover, the size-insensitive property is retained. Therefore, the proposed feature extraction network provides efficient information processing for decision making.

#### 4.4. Actor–Critic Framework and Training Algorithm

Based on the feature extraction network, we construct an actor–critic framework, which consists of a policy network and a value network, and train it by PPO.

##### 4.4.1. Policy Network

For mapping the extracted embeddings to policy  $\pi(a_t|S_t)$ , a size-agnostic policy network is proposed in our method. The main part of this policy network is the multilayer perceptron (MLP), which consists of fully connected layers (FCs). A vector representing the priority of all containers is calculated by MLP:

$$Score = MLP_{\pi}(E_{fe}), \quad (14)$$

where  $E_{fe} \in \mathbb{R}^{N \times d_h}$ ,  $Score \in \mathbb{R}^{N \times 1}$ ,  $MLP_{\pi}$  is an MLP (activated by a rectified linear unit (ReLU)) with two hidden layers with dimensions of 64 and 32, respectively.

To guarantee the correctness of the learned policy, we mask out the ineligible containers prior to softmax. The process of calculating the policy from  $Score$  can be formulated as:

$$Score'_c = \begin{cases} -\infty & \text{if container } c \text{ is ineligible} \\ Score_c & \text{else,} \end{cases} \quad (15)$$

$$\pi(a_t|S_t) = Softmax(Score'). \quad (16)$$

Following the aforementioned process, actions with higher scores that are not masked have a higher probability of being selected. The probabilities for ineligible containers are 0 and, hence, they will never be selected.

##### 4.4.2. Value Network

In the actor–critic framework, the calculation of the value function is needed to estimate the cumulative rewards of the current state  $S_t$  and to guide the training of the agent. For this requirement, we propose an MLP-based value network, which has three hidden layers with dimensions of 256, 128 and 64, respectively, and the dimension of the output is 1. The input is a vector obtained by column summation of the embedding  $E_{fe}$ . The calculation of the value can be formulated as:

$$v(S_t) = MLP_v(\text{mean}(E_{fe})), \quad (17)$$

where  $MLP_v$  is the MLP in the value network. Its hidden layers are activated by ReLU, but its output layer is not activated.

##### 4.4.3. Training Algorithm

Based on the aforementioned networks, an actor–critic framework is constructed. In our approach, the Proximal Policy Optimization (PPO) algorithm [12] with Adam optimizer is employed to train the neural network. The algorithm's details are presented in Algorithm 1. In order to boost the learning efficiency, we create  $\mathcal{N}$  parallel simulation environments with different training sets to collect rollout data. Each simulation environment collects data every  $\mathcal{C}$  steps (i.e., collection steps). According to the collected data, the network parameters are updated for  $\mathcal{I}$  iterations with batch size  $\mathcal{B}$ . After  $\mathcal{E}$  epochs of training, the model with optimal results is selected for scheduling.

**Algorithm 1:** Training Algorithm

---

**Input:** feature extractor network  $e_\zeta$  with trainable parameters  $\zeta$ ; actor network  $\pi_\theta$  with trainable parameters  $\theta$ ; critic network  $v_\phi$  with trainable parameters  $\phi$ ; total episode  $\mathcal{E}$ , number of parallel environments  $\mathcal{N}$ , collection steps  $\mathcal{C}$ , update epochs  $\mathcal{I}$ , Minibatch  $\mathcal{B}$ ;

```

1 Generate  $\mathcal{N}$  parallel environments;
2 while  $e < \mathcal{E}$  do
3   Initialize the rollout buffer  $M_b$  as an empty list;
4   for  $n = 1, 2, \dots, \mathcal{N}$  do
5     while  $c < \mathcal{C}$  do
6       sample an instance without putting it back;
7       reset the environment with selected instance;
8        $t \leftarrow 0$ ;
9       while not Done do
10        get state  $S_t$ ;
11        sample  $a_t$  based on  $\pi(a_t|S_t)$ ;
12        receive reward  $r_t$  and next state  $S_{t+1}$ ;
13         $t \leftarrow t + 1$ ;
14        append( $S_t, a_t, r_t, S_{t+1}$ ) to  $M_b$ ;
15         $c \leftarrow c + 1$ ;
16    $R = v_\phi(S_t)$ ;
17   for  $i = 1, 2, \dots, \mathcal{I}$  do
18     reset gradients:  $d\zeta \leftarrow 0$ ;  $d\theta \leftarrow 0$ ;  $d\phi \leftarrow 0$ ;
19     for  $j = 1, 2, \dots, \mathcal{N} \times \mathcal{C}/\mathcal{B}$  do
20       for  $i \in \{1, 2, \dots, \mathcal{B}\}$  do
21          $R \leftarrow r_i + \gamma R$ ;  $\delta \leftarrow R - v_\phi(S_i)$ ;
22          $d\zeta \leftarrow d\zeta + \sum_{M_b} \delta \nabla e_\zeta(S_i)$ ;
23          $d\theta \leftarrow d\theta + \sum_{M_b} \delta \nabla \log \pi(a_i|e_\zeta(S_i))$ ;
24          $d\phi \leftarrow d\phi + \sum_{M_b} \delta \nabla v_\phi(a_i|e_\zeta(S_i))$ ;
25       clip  $d\zeta, d\theta$  and  $d\phi$  by  $[-0.2, 0.2]$ ;
26       update  $\zeta, \theta$  and  $\phi$  by  $\frac{d\zeta}{|M_b|(d-d_s)}$ ,  $\frac{d\theta}{|M_b|(t-t_s)}$  and  $\frac{d\phi}{|M_b|(t-t_s)}$ ;
27    $e \leftarrow e + 1$ ;

```

---

**5. Experiment**

The experimental details and performance of the proposed method are presented and discussed in this section.

**5.1. Experiment Settings****5.1.1. Simulation Environment Settings**

The parameters of the simulated scheduling environment are determined according to scenarios in real-world ACT and previous research [16,20]. In this article, the block has 41 bays (39 bays for storage containers), which is a medium-scale block in a real ACT. The capacity of the seaside IO is five. For ease of description, the time-related parameters are set to integers according to the proportional relationship in the actual handling process. Concretely, the time of an ASC moving the distance of one bay is defined as one unit of time, and the time for the ASC to pick up or drop a container is set as  $\tau = 3$ . Following most research, we consider that the ASCs travel at a constant velocity  $\eta = 1$  (one bay per unit time). Considering the transportation process of the AGVs [44] and the proportional relationship between AGV transportation time and ASC handling time in the actual ACT operation [2], the observation time window is set as 60. For import containers out of the observation time window, the difference between the arrival time and the current time is set as a large number  $\mathcal{M} = 80$ .



### 5.1.2. Instance Generation

To provide training data for DRL and to evaluate the performance, the instances are randomly generated by a generator with four parameters: the number of containers ( $N$ ), the percentage of import containers (POI), the mathematical expectation of the interval time between AGVs with import containers ( $1/\lambda_i$ ) and the mathematical expectation of the interval time between empty AGVs ( $1/\lambda_e$ ). Here,  $N$  is used to control the problem scale, and  $N$  containers are separated into  $N_i$  import containers and  $N_e$  export containers according to the POI. For the import containers, their origin bays are the seaside IO (bay 0), and the destination bays are drawn from the uniform distribution  $U(1, 39)$ . On the contrary, the destination bays of export containers are the seaside IO, and their origin bays are drawn from the uniform distribution  $U(1, 39)$ . In dynamic scheduling problems, the Poisson process is generally used to model dynamic arrival [6]. Here, the arrival time of AGVs with import containers and the arrival time of empty AGVs are generated by Poisson process with  $\lambda_i$  and  $\lambda_e$ , respectively.

We generate five groups of instances of different scales for training and testing by setting  $N$  to 20, 30, 40, 50 and 60, respectively denoted as C20, ..., C60. For training, we set the POI = 50%,  $\lambda_i = 1/26$  and  $\lambda_e = 1/30$ . For each scenario, we generate 30,000 instances for training and an additional 1000 instances for testing.

### 5.1.3. Baselines

In this paper, we compare the scheduling performance of our method against a set of rule-based heuristics and two DRL baselines. The heuristics are well-known scheduling rules for dynamic JSP with sequence-dependent setup times [45] and twin ASCs scheduling [13]. Specifically, the rules are Random, Shortest Processing Time (SPT), Longest Processing Time (LPT), Shortest Setup Time (SST) (i.e., the nearest neighbor rule in [13]) and Prioritize Buffer Clearing (PBC) (i.e., prioritize moving containers out of the seaside IO). To demonstrate the effectiveness of our feature extraction network design, we also compare with two feature extraction structures: one based on the standard attention mechanism (AM) as in [37,42] and one based on the multilayer perceptron (MLP) as in [31–33]. Concretely, the AM model is a two-layer MHA with eight heads, and the MLP has two hidden layers, which have 64 and 128 neurons, respectively.

### 5.1.4. Hyperparameters and Implementation

In our method, we perform  $\mathcal{E} = 2000$  epochs of training with  $\mathcal{N} = 10$  parallel environments. For PPO, we set the clip range as 0.2, batch size  $\mathcal{B} = 2048$ , number of update iterations  $\mathcal{I} = 50$  and learning rate  $Lr = 1 \times 10^{-4}$ . To boost the learning efficiency on large-scale problems (C50 and C60), we adopt the “warm start” method for training. Specifically, we load the model trained on the C40 (i.e.,  $N = 40$ ) instance set and train it on the corresponding scale training set for  $\mathcal{E} = 1000$  epochs.

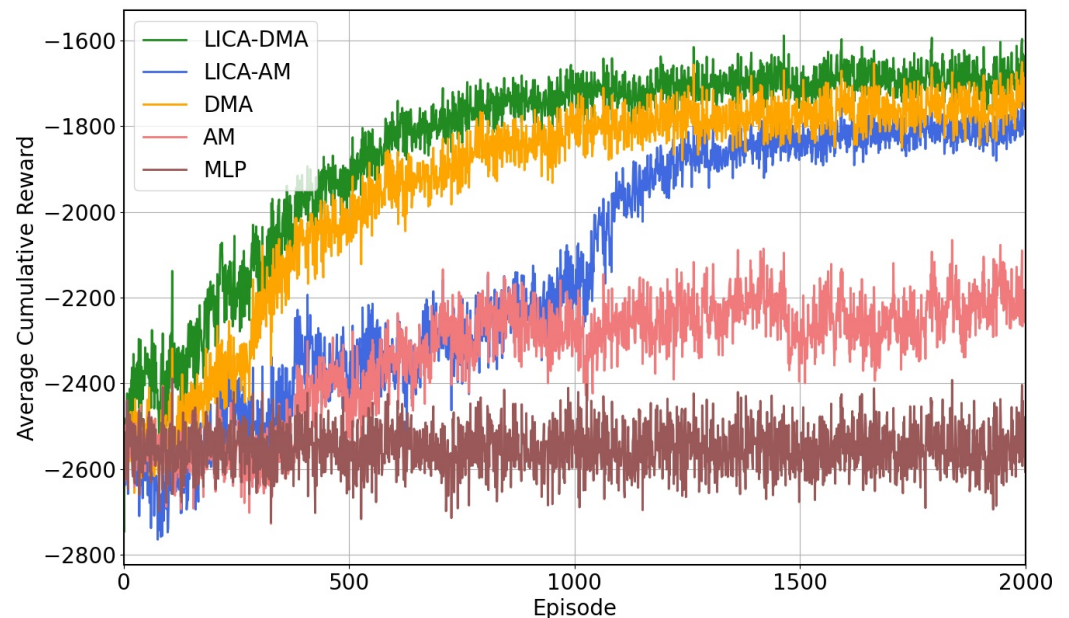
Our method and baselines are implemented in Python. The environment and DRL algorithm are implemented based on OpenAI Gym and Stable-baselines3. The hardware we use is a machine with an Intel Core i9-10920X CPU and a single Nvidia GeForce 2080Ti GPU.

## 5.2. Training and Ablation Experiments

Here, we perform an ablation study to evaluate the effectiveness of the two key components of our method: i.e., DMA and LICA. We compare five models with different feature extractors, including: the MLP model (MLP), the standard self-attention model (AM), the DMA model (DMA), the LICA model with standard self-attention (LICA-AM), and the LICA model with DMA (LICA-DMA). We train the above models on the representative medium-scale problem C40 and evaluate them on the testing set. The training curves and testing performance are demonstrated in Figure 4 and Table 1, respectively.

As shown in Figure 4, the MLP model with a simple neural structure fails to learn effective policies due to its poor feature extraction capability. Therefore, it is excluded from the following comparison. DMA converges faster than AM and converges to much higher

cumulative rewards, showing the effectiveness of its capability in automatically eliminating interference and extracting the location-related interrelationships between containers and ASCs. The advantage of DMA over AM is also validated in Table 1. Obviously, DMA is more competent in extracting embeddings under information interference than AM.



**Figure 4.** Training curves of DRL methods with different neural architectures.

**Table 1.** Scheduling performance of rules and DRL methods on medium-scale (C40) problem.

Method	$T_{run}$	$T_{wait}$	Obj.
Random	1790.41(140.16)	766.30(435.69)	2556.72(548.29)
SPT	1744.30(140.39)	424.11(333.45)	2168.42(446.88)
LPT	1799.49(144.94)	1291.49(489.62)	3090.99(611.57)
SST	<sup>†</sup> 1533.78(155.59)	319.02(289.14)	<sup>†</sup> 1852.80(417.50)
PBC	1892.11(145.52)	<sup>†</sup> 202.32(278.61)	2094.43(388.36)
AM	1781.74(131.58)	362.59(339.29)	2144.33(435.13)
DMA	1517.98(142.05)	159.55(219.59)	1677.53(334.33)
LICA-AM	1698.65(134.32)	<b>74.52(156.00)</b>	1773.17(256.37)
<b>LICA-DMA</b>	<b>1505.89(152.36)</b>	138.81(202.99)	<b>1644.70(326.79)</b>

"Obj.", " $T_{run}$ " and " $T_{wait}$ " are the average objective, run time of ASCs and the total waiting time of AGVs on 1000 testing instances, respectively. <sup>†</sup> means the average result of the best rule-based heuristics, while **Bold** means average result of the best method among all methods, and the scalar in () is standard deviation.

Through the LICA framework, the location-related embeddings are complemented by the congestion-related information obtained by LSTM. Therefore, as demonstrated in Figure 4 and Table 1, the framework can effectively enhance the DRL model's performance. During training, LICA-AM converges to higher rewards than AM. Similarly, LICA-DMA performs better than DMA in training. For the evaluation performance, LICA-AM significantly improves AM by reducing  $T_{wait}$ . Moreover, LICA-DMA can grasp high-quality location-related embeddings and fuse them efficiently. As a result, LICA-DMA performs the best both in training and evaluation since it can coordinate  $T_{run}$  and  $T_{wait}$  more effectively.

### 5.3. Performance Comparison for Different Training Scales

In this subsection, we evaluate the performance of our method on different scales and compare it with other baselines. Concretely, the agents are trained on training sets of different scales (C20, ..., C60) and are tested on the corresponding scale testing sets with

the same distribution. Results are summarized in Tables 1 and 2. The proportions of  $T_{run}$  and  $T_{wait}$  in the objective on different scales are shown in Figure 5.

As shown in Tables 1 and 2, PBC is the best rule for minimizing  $T_{wait}$ , and SST is the best rule for minimizing  $T_{run}$  and the integrated objective. On the smallest scale C20, the DRL method with AM obtains the minimum objective, while our method has extremely similar performance. Except on C20, LICA-DMA consistently outperforms all baselines on all problem scales, and the difference is noticeable, particularly for large-scale problems. In addition, the policies learned by our approach have the lowest standard deviation on most scales. This suggests that our method has remarkable stability.

**Table 2.** Scheduling performance of rule-based heuristics and DRL methods on different scales.

		Random	SPT	LPT	SST	PBC	AM	LICA-DMA
C20	$T_{run}$	851.11(91.88)	837.93(85.56)	833.36(96.15)	<sup>†</sup> 682.24(69.70)	931.53(100.96)	<b>668.37(63.71)</b>	669.38(63.24)
	$T_{wait}$	39.57(48.92)	13.27(28.64)	72.53(64.16)	2.56(6.96)	<sup>†</sup> <b>0.73(5.03)</b>	1.42(4.84)	1.56(4.84)
	Obj.	890.69(126.63)	851.20(100.98)	905.89(146.94)	<sup>†</sup> 684.80(72.71)	932.26(102.44)	<b>669.79(65.26)</b>	670.95(64.70)
C30	$T_{run}$	1321.75(127.51)	1283.99(122.76)	1320.65(130.84)	<sup>†</sup> 1088.14(126.19)	1414.78(130.98)	1068.90(116.85)	<b>1049.55(115.13)</b>
	$T_{wait}$	277.38(203.40)	129.94(139.27)	510.07(245.41)	66.82(86.29)	<sup>†</sup> 36.70(71.37)	30.25(57.22)	<b>29.81(55.04)</b>
	Obj.	1599.12(306.72)	1413.93(238.29)	1830.72(358.10)	<sup>†</sup> 1154.96(196.73)	1451.48(181.63)	1099.16(157.37)	<b>1079.36(154.34)</b>
C50	$T_{run}$	2275.57(164.11)	2222.69(164.81)	2288.58(163.40)	<sup>†</sup> <b>1997.89(175.84)</b>	2379.37(166.19)	2275.33(168.84)	2120.47(182.81)
	$T_{wait}$	1514.66(731.03)	922.83(563.95)	2414.45(803.49)	807.33(533.50)	<sup>†</sup> 588.73(545.53)	836.62(664.84)	<b>299.19(371.68)</b>
	Obj.	3790.23(862.71)	3145.52(701.82)	4703.03(939.65)	<sup>†</sup> 2805.21(678.79)	2968.10(674.39)	3111.95(786.80)	<b>2419.66(509.94)</b>
C60	$T_{run}$	2747.31(173.79)	2689.26(177.66)	2754.90(179.33)	<sup>†</sup> <b>2441.54(192.38)</b>	2851.28(179.16)	2721.38(170.87)	2527.86(188.19)
	$T_{wait}$	2425.99(1074.21)	1578.18(860.18)	3707.13(1169.84)	1475.11(850.50)	<sup>†</sup> 1119.50(888.36)	1602.48(909.05)	<b>719.89(704.74)</b>
	Obj.	5173.31(1212.29)	4267.44(1007.66)	6462.03(1317.93)	<sup>†</sup> 3916.65(1009.92)	3970.79(1027.76)	4323.86(1051.04)	<b>3247.75(854.45)</b>

“Obj.”, “ $T_{run}$ ” and “ $T_{wait}$ ” are the average objective, run time of ASCs and the total waiting time of AGVs on 1000 testing instances, respectively. <sup>†</sup> means the average result of the best rule-based heuristics, while **Bold** means average result of the best method among all methods, and the scalar in () is standard deviation.



**Figure 5.** Average scheduling result analysis on small (C20), medium (C40) and large (C60) scale problem.

As the problem scales up, the complexity of the problem increases, the information interference increases, and the proportion of  $T_{wait}$  in the objective increases. The DRL method with AM cannot handle the aforementioned issues; thus, scheduling performance degrades significantly as the problem scale increases. Starting from C40, it cannot even outperform SST. However, our method, LICA-DMA, can effectively overcome the above issues. Consequently, our method maintains excellent scheduling performance on various

problem scales. As shown in Figure 5, our method can effectively minimize the run time of ASCs in small-scale problems. Furthermore, in medium-scale and large-scale problems, the proposed approach can minimize the integrated objective by weighing  $T_{run}$  and  $T_{wait}$ . From the comparison of DRL methods in Table 3, we can infer the superiority of our method. The scheduling performance of LICA-DMA is significantly better than AM in most scale problems. Compared with SST, the proposed DRL method yields  $T_{run}$  improvements of 1.88%, 3.55% and 1.82% for small and medium-scale issues. For large-scale problems, the agent is required to place greater emphasis on the influence of  $T_{wait}$  on the integrated objective. Consequently, it needs to sacrifice some  $T_{run}$ . For traffic congestion, the proposed DRL method exhibits improvements in  $T_{wait}$  when compared to the PBC rule. Specifically, the DRL method achieves enhancements of 18.77%, 31.39%, 49.18% and 35.7% for C30–C60 scales. The millisecond decision time,  $T_d$ , can meet the real-time scheduling requirements for decision speed. Furthermore, the increase in  $T_d$  with problem scale is within a reasonable range. Although the training time of LICA-DMA is longer than AM, the training time is acceptable in most circumstances due to the offline training of DRL.

**Table 3.** Performance comparison of DRL models on different scales.

		C20	C30	C40	C50	C60
$T_{tr}$	AM	2.77 h	5.29 h	8.78 h	6.69 h	9.46 h
	LICA-DMA	4.63 h	9.09 h	15.16 h	11.52 h	16.38 h
$T_d$	AM	2.76 ms	2.75 ms	3.12 ms	3.14 ms	3.14 ms
	LICA-DMA	5.11 ms	5.29 ms	5.90 ms	6.06 ms	6.18 ms
Impr.	AM	2.19%	4.83%	4.30%	−10.93%	−10.40%
	LICA-DMA	2.02%	6.55%	11.23%	13.74%	17.08%

$T_{tr}$  is the training time (in hours),  $T_d$  is the time for making one decision (in milliseconds). Impr. is the improvement against the best rule.

#### 5.4. Generalization Performance for Various Scales

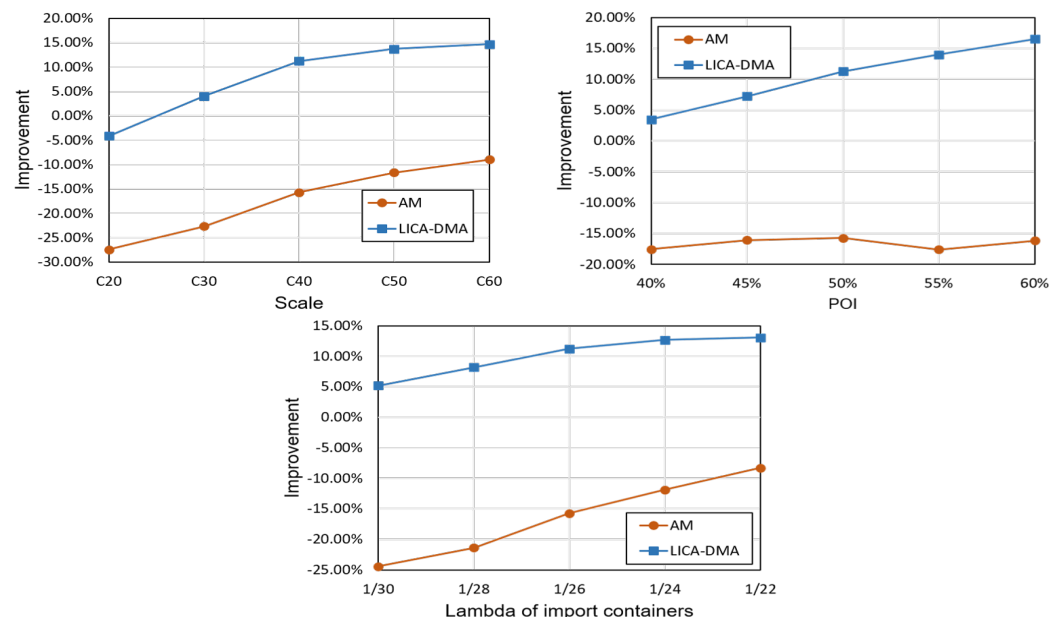
In practice, the scale of the scheduling task (i.e., the number of containers) could be different from those in training. Therefore, robust generalization capability is very important for the practical application of scheduling methods. To verify our method's generalization capability, the agent trained at scale C40 is generalized to schedule the dynamic twin ASCs scheduling problems ranging from C20 to C60. The average scheduling results are exhibited in Table 4, while the improvement of the generalized agent against the best existing methods are described in Figure 6 (left).

**Table 4.** Generalization performance on different problem scales.

Scale		AM	LICA-DMA
C20	$T_{run}$	865.58(97.51)	712.73(71.44)
	$T_{wait}$	7.24(22.22)	0.59(2.97)
	Obj.	872.82(105.94)	713.33(72.11)
C30	$T_{run}$	1318.43(125.67)	1086.39(110.75)
	$T_{wait}$	98.70(140.12)	21.88(48.37)
	Obj.	1417.13(232.24)	1108.27(142.16)
C50	$T_{run}$	2266.37(153.50)	1975.72(191.24)
	$T_{wait}$	866.39(612.24)	445.25(424.00)
	Obj.	3132.75(729.82)	2420.97(585.66)
C60	$T_{run}$	2735.67(165.16)	2429.99(209.61)
	$T_{wait}$	1533.02(925.14)	912.11(728.61)
	Obj.	4268.69(1051.62)	3342.10(903.70)

“Obj.”, “ $T_{run}$ ” and “ $T_{wait}$ ” are the average objective, run time of ASCs and the total waiting time of AGVs on 1000 testing instances, respectively. The scalar in () is standard deviation.

As the experimental results show, the generalization results of LICA-DMA exceed the best rule on most scales, but the generalization results of DRL with AM fail to do so. Compared with SST, the improvements of LICA-DMA on various scales are  $-4.17\%$ ,  $4.04\%$ ,  $11.23\%$ ,  $13.70\%$  and  $14.67\%$ . As the improvement curves of the DRL methods show, the improvement increases with the growth of the scale. This phenomenon may be caused by the fact that the inherent enhancements of methods grow more quickly than the decline brought on by scale differences. However, the generalization of AM is worthless due to its poor scheduling performance. In addition, compared with the LICA-DMA models trained with the corresponding scale instances, the gaps of the generalized LICA-DMA model on various scales are only  $6.32\%$ ,  $2.68\%$ ,  $0.05\%$  and  $2.90\%$ , respectively. In general, the results suggest that the policies learned by LICA-DMA have robust generalization capability for problems with different scales.



**Figure 6.** Generalization analysis: improvement curves of DRL models against the best rule for varying scales (left), POIs (right) and  $\lambda_i$  (bottom).

### 5.5. Generalization Performance for Different Distributions

In practice, the distribution of containers in the scheduling task could be different from that in training. In the dynamic twin ASCs scheduling problem, the main item of distribution is the percentage of import containers (POI). In order to validate our method's generalization performance on different POIs, we test the policy trained on the C40 scale with POI = 50% and the testing instances with POI = {40%, 45%, 55%, 60%}. Due to the limited space, policies trained on other scales with similar performance are not discussed here. Additionally, we only compare with SST here, while other rule-based heuristics are dropped due to their relatively poor performance.

Comparing the experimental results in Table 5, we can see that the policies learned by our method, LICA-DMA, obtained the lowest objective for all POIs, whereas the policies learned by AM cannot outperform the best-rule SST.

The improvement curves of generalization models against the best scheduling rule on varying POIs are described in Figure 6 (right). Specifically, AM fails to exceed the best-rule SST, and the improvements of LICA-DMA for various POIs are  $3.49\%$ ,  $7.25\%$ ,  $11.23\%$ ,  $13.95\%$  and  $16.51\%$ , respectively. The curve of AM is not sensitive to different POIs, but it does not show that the policy learned by AM model has good generalization ability due to its poor scheduling performance. As the curve of LICA-DMA shows, the improvement increases as the POI increases. This phenomenon is caused by the following reason: our approach handles blocking more effectively than SST. As the POI increases, the



growth of the inherent enhancement of our method is faster than the decrease caused by the POI difference. As a result, the generalization ability of our method for different POIs is indirectly demonstrated.

**Table 5.** Generalization performance on problems with different POIs.

POI		SST	AM	LICA-DMA
40%	$T_{run}$	1519.11(151.82)	1786.94(134.03)	1523.14(142.02)
	$T_{wait}$	95.52(113.54)	110.79(138.10)	35.19(68.39)
	Obj.	1614.63(241.12)	1897.73(238.16)	1558.33(187.67)
45%	$T_{run}$	1520.18(158.07)	1770.91(136.92)	1508.00(153.19)
	$T_{wait}$	186.92(184.87)	210.22(229.46)	75.30(120.18)
	Obj.	1707.10(318.65)	1981.13(332.85)	1583.31(247.99)
55%	$T_{run}$	1541.45(152.32)	1800.53(134.61)	1510.88(151.25)
	$T_{wait}$	471.82(353.01)	567.04(446.75)	221.48(260.22)
	Obj.	2013.28(481.47)	2367.56(550.35)	1732.36(385.94)
60%	$T_{run}$	1550.20(157.41)	1804.27(150.85)	1518.61(159.70)
	$T_{wait}$	673.46(493.41)	778.45(581.94)	337.94(373.56)
	Obj.	2223.66(630.99)	2582.72(702.49)	1856.55(510.25)

“Obj.”, “ $T_{run}$ ” and “ $T_{wait}$ ” are the average objective, run time of ASCs and the total waiting time of AGVs on 1000 testing instances, respectively. The scalar in () is standard deviation.

### 5.6. Generalization for Different AGV Arrival Processes

In an actual ACT, the number of AGVs assigned to each loading/unloading operation is varied, and the relative positions of the blocks to the vessel are also different. Therefore, the AGV arrival process is variable. In this article, the AGV arrival process is abstracted to a Poisson process, so we use  $\lambda_i$  and  $\lambda_e$  to regulate the arrival concentration of AGVs carrying import containers and empty AGVs. To validate the generalization performance on different AGV arrival processes, the policy learned on C40 training instances ( $\lambda_i = 1/26$ ,  $\lambda_e = 1/30$ ) is employed to solve problems with different  $\lambda_i$  (1/22, 1/24, 1/28 and 1/30) and  $\lambda_e$  (1/26, 1/28, 1/32 and 1/34).

The scheduling performance of SST and DRL-based generalization models on different  $\lambda_i$  is exhibited in Table 6. Although the results of the AM-based generalization model fail to exceed the best scheduling rule, SST, the scheduling performance of the generalization model based on LICA-DMA is noticeably superior to other baselines.

**Table 6.** Generalization performance for problems with different  $\lambda_i$ .

$\lambda_i$		SST	AM	LICA-DMA
$\frac{1}{22}$	$T_{run}$	1597.23(140.51)	1802.14(130.35)	1582.56(146.17)
	$T_{wait}$	685.25(384.42)	670.20(464.91)	402.35(333.17)
	Obj.	2282.48(499.23)	2472.34(562.67)	1984.91(450.78)
$\frac{1}{24}$	$T_{run}$	1571.34(153.28)	1795.89(134.94)	1549.77(156.12)
	$T_{wait}$	491.21(331.52)	511.90(396.33)	251.12(256.65)
	Obj.	2062.55(458.90)	2307.79(496.48)	1800.88(384.95)
$\frac{1}{28}$	$T_{run}$	1493.44(160.58)	1775.80(137.52)	1473.26(152.73)
	$T_{wait}$	195.00(216.08)	273.90(308.34)	77.39(144.28)
	Obj.	1688.44(352.54)	2049.69(411.90)	1550.65(271.05)
$\frac{1}{30}$	$T_{run}$	1451.95(162.48)	1767.06(144.07)	1439.83(142.47)
	$T_{wait}$	99.88(148.05)	164.16(222.13)	31.52(80.58)
	Obj.	1551.83(288.34)	1931.22(328.26)	1471.34(201.04)

“Obj.”, “ $T_{run}$ ” and “ $T_{wait}$ ” are the average objective, run time of ASCs and the total waiting time of AGVs on 1000 testing instances, respectively. The scalar in () is standard deviation.

The improvement curves against the best scheduling rule for varying  $\lambda_i$  are described in Figure 6 (bottom). As the curves show, the improvement increases with the growth

of  $\lambda_i$ . This phenomenon may be caused by the fact that the inherent enhancements of methods grow more quickly than the decline brought on by  $\lambda_i$  differences. Indirectly, the generalization ability of the DRL methods is demonstrated. However the generalization of the AM-based model is worthless in practice due to its unsatisfactory scheduling results. As the LICA-DMA curve shows, the generalized model remarkably outperforms SST. Specifically, the improvements to various  $\lambda_i$  are 5.19%, 8.16%, 11.23%, 12.69% and 13.04%, respectively.

From the characteristics discussed in Section 3, we can infer that the effect of  $\lambda_e$  on congestion is very limited. In the generalization experiment with different  $\lambda_e$ , for each method, the scheduling differences caused by different  $\lambda_e$  are extremely small. Therefore, the details of the generalization experiment for different  $\lambda_e$  are not discussed here due to the limited space.

**Remark 4.** *Benefiting from the LICA-DMA, the agent can eliminate the information interference of ineligible containers automatically and can complement the container information with congestion-related information. Compared with existing methods, our method has superior scheduling performance and robust generalization capability, which is promising for practical applications.*

## 6. Conclusions and Future Work

Efficient dynamic scheduling of twin ASCs can significantly boost the handling efficiency of the storage yard, which is crucial to improving the service quality of ACT and reducing energy consumption. In this paper, we propose an end-to-end DRL method to solve the dynamic twin ASCs scheduling problem for the first time. To tackle redundant and incomplete state information, we propose a size-insensitive neural architecture, LICA-DMA, which is able to automatically eliminate the information interference of ineligible containers and can complement the container information with congestion-related information. As the extensive experimental results show, the policies learned by DRL with LICA-DMA can obtain significantly better solutions with extremely low time complexity. Furthermore, the learned policies exhibit robust generalization on different scales and distributions. The proposed approach is promising for improving the handling efficiency of storage yards in complex practical scenarios. In addition, this work extends the technical framework of DRL to solve dynamic scheduling problems under complex constraints. For future work, we intend to extend the DRL-based scheduling to the entire ACT by integrating other equipment in the ACT, such as automated guided vehicles (AGVs) and quay cranes (QCs).

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

**Funding:** This research is funded by the National Natural Science Foundation of China under grant 62102228 and by the Shandong Provincial Natural Science Foundation under grant ZR2021QF063.

**Data Availability Statement:** Not applicable.

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

## References

1. Dey, K.; Fries, R.; Ahmed, S. Future of transportation cyber-physical systems—Smart cities/regions. In *Transportation Cyber-Physical Systems*; Elsevier: Amsterdam, The Netherlands, 2018; pp. 267–307.
2. Kizilay, D.; Eliyi, D.T. A comprehensive review of quay crane scheduling, yard operations and integrations thereof in container terminals. *Flex. Serv. Manuf. J.* **2021**, *33*, 1–42. [\[CrossRef\]](#)
3. Jin, X.; Duan, Z.; Song, W.; Li, Q. Container stacking optimization based on Deep Reinforcement Learning. *Eng. Appl. Artif. Intell.* **2023**, *123*, 106508. [\[CrossRef\]](#)

4. Zhu, S.; Tan, Z.; Yang, Z.; Cai, L. Quay crane and yard truck dual-cycle scheduling with mixed storage strategy. *Adv. Eng. Inform.* **2022**, *54*, 101722. [\[CrossRef\]](#)
5. Chen, X.; He, S.; Zhang, Y.; Tong, L.C.; Shang, P.; Zhou, X. Yard crane and AGV scheduling in automated container terminal: A multi-robot task allocation framework. *Transport. Res. C-Emer.* **2020**, *114*, 241–271. [\[CrossRef\]](#)
6. Zhang, F.; Mei, Y.; Nguyen, S.; Zhang, M. Evolving Scheduling Heuristics via Genetic Programming with Feature Selection in Dynamic Flexible Job-Shop Scheduling. *IEEE Trans. Cybern.* **2021**, *51*, 1797–1811. [\[CrossRef\]](#)
7. 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. [\[CrossRef\]](#)
8. Song, W.; Cao, Z.; Zhang, J.; Xu, C.; Lim, A. Learning variable ordering heuristics for solving constraint satisfaction problems. *Eng. Appl. Artif. Intell.* **2022**, *109*, 104603. [\[CrossRef\]](#)
9. Song, H.; Kim, M.; Park, D.; Shin, Y.; Lee, J.-G. Learning From Noisy Labels With Deep Neural Networks: A Survey. *IEEE Trans. Neur. Net. Lear.* **2022**, 1–19. [\[CrossRef\]](#)
10. Bengio, Y.; Lodi, A.; Prouvost, A. Machine learning for combinatorial optimization: A methodological tour d’horizon. *Eur. J. Oper. Res.* **2021**, *290*, 405–421. [\[CrossRef\]](#)
11. Wu, Y.; Song, W.; Cao, Z.; Zhang, J.; Lim, A. Learning improvement heuristics for solving routing problems. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *33*, 5057–5069. [\[CrossRef\]](#)
12. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
13. Gharehgozli, A.H.; Vernooij, F.G.; Zaerpour, N. A simulation study of the performance of twin automated stacking cranes at a seaport container terminal. *Eur. J. Oper. Res.* **2017**, *261*, 108–128. [\[CrossRef\]](#)
14. Carlo, H.J.; Martínez-Acevedo, F.L. Priority rules for twin automated stacking cranes that collaborate. *Comput. Ind. Eng.* **2015**, *89*, 23–33. [\[CrossRef\]](#)
15. Briskorn, D.; Emde, S.; Boysen, N. Cooperative twin-crane scheduling. *Discrete App. Math.* **2016**, *211*, 40–57. [\[CrossRef\]](#)
16. Han, X.; Wang, Q.; Huang, J. Scheduling cooperative twin automated stacking cranes in automated container terminals. *Comput. Ind. Eng.* **2019**, *128*, 553–558. [\[CrossRef\]](#)
17. Oladugba, A.O.; Gheith, M.; Eltawil, A. A new solution approach for the twin yard crane scheduling problem in automated container terminals. *Adv. Eng. Inform.* **2023**, *57*, 102015. [\[CrossRef\]](#)
18. Jaehn, F.; Kress, D. Scheduling cooperative gantry cranes with seaside and landside jobs. *Discrete App. Math.* **2018**, *242*, 53–68. [\[CrossRef\]](#)
19. Kress, D.; Dornseifer, J.; Jaehn, F. An exact solution approach for scheduling cooperative gantry cranes. *Eur. J. Oper. Res.* **2019**, *273*, 82–101. [\[CrossRef\]](#)
20. Lu, H.; Wang, S. A study on multi-ASC scheduling method of automated container terminals based on graph theory. *Comput. Ind. Eng.* **2019**, *129*, 404–416. [\[CrossRef\]](#)
21. Zheng, F.; Man, X.; Chu, F.; Liu, M.; Chu, C. Two Yard Crane Scheduling With Dynamic Processing Time and Interference. *IEEE Trans. Intell. Transp.* **2018**, *19*, 3775–3784. [\[CrossRef\]](#)
22. Xiong, H.; Shi, S.; Ren, D.; Hu, J. A survey of job shop scheduling problem: The types and models. *Comput. Ind. Eng.* **2022**, *142*, 105731. [\[CrossRef\]](#)
23. Yang, S.; Wang, J.; Xu, Z. Real-time scheduling for distributed permutation flowshops with dynamic job arrivals using deep reinforcement learning. *Adv. Eng. Inform.* **2022**, *54*, 101776. [\[CrossRef\]](#)
24. Ni, F.; Hao, J.; Lu, J.; Tong, X.; Yuan, M.; Duan, J.; Ma, Y.; He, K. A Multi-Graph Attributed Reinforcement Learning Based Optimization Algorithm for Large-Scale Hybrid Flow Shop Scheduling Problem. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, New York, NY, USA, 14–18 August 2021; pp. 3441–3451.
25. Song, W.; Chen, X.; Li, Q.; Cao, Z. Flexible Job Shop Scheduling via Graph Neural Network and Deep Reinforcement Learning. *IEEE Trans. Ind. Inform.* **2022**, *19*, 1600–1610. [\[CrossRef\]](#)
26. Lin, C.; Deng, D.; Chih, Y.; Chiu, H.T. Smart manufacturing scheduling with edge computing using multiclass deep Q network. *IEEE Trans. Ind. Inform.* **2019**, *15*, 4276–4284. [\[CrossRef\]](#)
27. Zhang, C.; Song, W.; Cao, Z.; Zhang, J.; Tan, P.S.; Chi, X. Learning to dispatch for job shop scheduling via deep reinforcement learning. *Proc. Adv. Neur. Inf. Process. Syst.* **2020**, *33*, 1621–1632.
28. Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; Philip, S.Y. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *32*, 4–24. [\[CrossRef\]](#)
29. Park, J.; Chun, J.; Kim, S.H.; Kim, Y.; Park, J. Learning to schedule job-shop problems: Representation and policy learning using graph neural network and reinforcement learning. *Int. J. Prod. Res.* **2021**, *59*, 3360–3377. [\[CrossRef\]](#)
30. Park, J.; Bakhtiyar, S.; Park, J. ScheduleNet: Learn to solve multi-agent scheduling problems with reinforcement learning. *arXiv* **2021**, arXiv:2106.03051.
31. Zhao, Y.; Wang, Y.; Tan, Y.; Zhang, J.; Yu, H. Dynamic Jobshop Scheduling Algorithm Based on Deep Q Network. *IEEE Access* **2021**, *9*, 122995–123011. [\[CrossRef\]](#)
32. 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. [\[CrossRef\]](#)

33. Zhang, Y.; Zhu, H.; Tang, D.; Zhou, T.; Gui, Y. Dynamic job shop scheduling based on deep reinforcement learning for multi-agent manufacturing systems. *Robot. Comput.-Int. Manuf.* **2022**, *78*, 102412. [[CrossRef](#)]
34. Saanen, Y.; Valkengoed, M. Comparison of three automated stacking alternatives by means of simulation. In Proceedings of the Winter Simulation Conference, Orlando, FL, USA, 4 December 2005; p. 10. [[CrossRef](#)]
35. Park, K.; Park, T.; Ryu, K.R. Planning for remarshaling in an automated container terminal using cooperative coevolutionary algorithms. In Proceedings of the 2009 ACM Symposium on Applied Computing, New York, NY, USA, 8–12 March 2009; pp. 1098–1105.
36. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. In Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; Volume 30.
37. Kool, W.; van Hoof, H.; Welling, M. Attention, Learn to Solve Routing Problems! In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
38. Li, J.; Ma, Y.; Gao, R.; Cao, Z.; Lim, A.; Song, W.; Zhang, J. Deep Reinforcement Learning for Solving the Heterogeneous Capacitated Vehicle Routing Problem. *IEEE Trans. Cybern.* **2021**, *52*, 13572–13585. [[CrossRef](#)]
39. Xin, L.; Song, W.; Cao, Z.; Zhang, J. Step-Wise Deep Learning Models for Solving Routing Problems. *IEEE Trans. Ind. Inform.* **2021**, *17*, 4861–4871. [[CrossRef](#)]
40. Li, K.; Zhang, T.; Wang, R.; Wang, Y.; Han, Y.; Wang, L. Deep Reinforcement Learning for Combinatorial Optimization: Covering Salesman Problems. *IEEE Trans. Cybern.* **2021**, *52*, 13142–13155. [[CrossRef](#)]
41. Vinyals, O.; Fortunato, M.; Jaitly, N. Pointer networks. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015; Volume 28.
42. Song, W.; Mi, N.; Li, Q.; Zhuang, J.; Cao, Z. Stochastic Economic Lot Scheduling via Self-Attention Based Deep Reinforcement Learning. *IEEE Trans. Autom. Sci. Eng.* **2023**. [[CrossRef](#)]
43. Xu, X.; Yoneda, M. Multitask Air-Quality Prediction Based on LSTM-Autoencoder Model. *IEEE Trans. Cybern.* **2021**, *51*, 2577–2586. [[CrossRef](#)]
44. 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](#)]
45. Vinod, K.; Prabakaran, S.; Joseph, O. Dynamic due date assignment method: A simulation study in a job shop with sequence-dependent setups. *J. Manuf. Technol. Mana.* **2019**, *30*, 987–1003.

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.