applied sciences

MDPI

Article

Deriving Situation-Adaptive Policy for Container Stacking in an Automated Container Terminal

Taekwang Kim [1] and Kwang Ryel Ryu [2,*]

[1] Department of Information Convergence Engineering, Pusan National University, Busan 46241, Korea; kimt86@pusan.ac.kr
[2] School of Computer Science and Engineering, Pusan National University, Busan 46241, Korea
* Correspondence: krryu@pusan.ac.kr

Abstract: Determining where to stack the containers at the storage yard of a container terminal is an important problem because that decision critically affects the efficiency of container handling in the yard and, eventually, the efficiency of the vessel operations, which is considered the most important for the productivity of the whole terminal. One limitation of the stacking policies previously proposed is that they are static in nature. Although good locations for stacking may change as the workload of vessel operation changes, the previous policies are insensitive to such changes. Failure to recommend good locations leads to elongated operations of yard cranes and thus makes it hard for them to keep up with the workload of vessel operation. In this paper, we propose a method for deriving a dynamic policy that can adapt to the workload of vessel operation that changes over time. Our method derives two boundary policies: one for very high workload and the other for very low. Then, a policy appropriate for any intermediate workload can be synthesized from the two boundary policies through interpolation. Simulation experiments showed that the proposed policy significantly reduced overall container handling time compared to the previous static policy. When measured in terms of the time the transportation vehicles wait for container handling services, the improvement was approximately 19%.

Keywords: container terminal; storage yard; container stacking; situation-adaptive policy; optimization; genetic algorithm

Citation: Kim, T.; Ryu, K.R. Deriving Situation-Adaptive Policy for Container Stacking in an Automated Container Terminal. *Appl. Sci.* **2022**, *12*, 3892. https://doi.org/10.3390/app12083892

Academic Editors: Nuno Lau, Rui Araújo, António Pedro Aguiar, Rodrigo Ventura and João Fabro

Received: 1 March 2022
Accepted: 11 April 2022
Published: 12 April 2022

1. Introduction

One of the most important operational goals of a container terminal is to minimize the vessel turnaround time by maximizing the efficiency of *vessel operations* with regard to loading or unloading containers onto or from the vessels. The *outbound* containers brought in by the trucks from inland are stored in the yard until they are loaded onto the vessels. On the other hand, the *inbound* containers unloaded from the vessels dwell in the storage yard until they are claimed by the external trucks for inland transportation. As buffer storage for both inbound and outbound containers, the operational efficiency in the storage yard critically affects the overall productivity of the terminal. A very important factor that affects the operational efficiency in the storage yard is the determination of the stacking locations of the containers that arrive at the yard. If, for example, a container just unloaded from a vessel is stacked on top of another to be loaded soon, the upper one has to be relocated to a different stack when retrieving the lower one. Such *rehandling* of containers should be minimized in order to maximize the operational efficiency in the yard. When the containers are loaded onto a vessel, they follow a predetermined sequence. The loading sequence is determined at the planning stage, taking into account the vessel stability, ports of destination, and the efficiency of operation at the storage yard. Still, rehandling is the major cause of loading delay because the retrieval schedules are usually unknown at the time the containers arrive at the yard, and thus, they can be stacked at the wrong locations. In this paper, we deal with the problem of determining good locations for stacking not

only the containers that newly arrive at the yard but also those that are relocated within the yard.

Perhaps the simplest approach to determining the stacking locations in the yard is using a heuristic rule based on a simple criterion such as preferring the nearest location or preferring the stack-top of containers of the same category [1–3]. More recent and advanced methods use multiple criteria to evaluate candidate stacking locations from various perspectives [4–6]. These methods use *scoring functions* to calculate the score of a candidate location by a weighted sum of the results of evaluations based on various criteria. Since the scores and the resulting best stacking location depend on the weight combination, or *weight vector*, used in the scoring function, the weight vector can be regarded as a *stacking policy*. To find a good weight vector, i.e., a good stacking policy, References [4,6] use genetic algorithms (GA) in which each candidate policy is evaluated through simulations of applying the policy to various scenarios of operations in the yard and measuring the resulting performance. However, the policy found in this way is the one whose *average* performance in various scenarios is the best. In certain situations, there may be other policies that work better than the average policy.

What we propose in this paper is a method for deriving a stacking policy, which can be adapted to changing situations by having its weight vectors adjusted. As an indicator of the situation, we use the workload of vessel operation because good stacking locations are very much dependent on it. Our method is based on the idea that a new policy may be synthesized if we are given two boundary policies: one for a very low workload and the other for a very high workload. We assume that a good policy for any intermediate situation can be derived by taking an interpolation of the weight vectors of the two boundary policies. We use a GA to search for not only the two boundary policies but also the two numeric values that quantify the two fuzzy terms 'very low' and 'very high'. Our GA can be seen as a reinforcement learning algorithm conducting a search in the policy space [7] instead of learning a value function based on rewards from the environment. Experimental results show that our method performs better by dynamically adapting the stacking policy to varying situations than the previous methods that use a static best on-average policy.

The rest of the paper is organized as follows. Section 2 gives a detailed description of the operations in the storage yard of an automated container terminal. Section 3 reviews the related works, and Section 4 describes the stacking policy based on the scoring functions. Section 5 explains how we derive a situation-adaptive stacking policy by using a GA. Section 6 reports the results of experiments, and Section 7 discusses how we can extend our method to the cases with multiple situation indicators. Finally, Section 8 gives some concluding remarks.

2. Operations in the Storage Yard of an Automated Container Terminal

The descriptions in this section are mostly based on the material given in [4,6,8]. As can be seen in Figure 1, the automated container terminal can be largely divided into four regions: quay, apron, storage yard, and hinterland. The quay is where the vessels berth and a number of quay cranes (QC) load/unload containers onto/from the vessels. The apron is the area for the automated guided vehicles (AGV) to deliver containers between the quay and the storage yard. The hinterland is where the external trucks (ET) bring in containers to the storage yard or bring out the containers picked up from the storage yard. The storage yard consists of dozens of rectangular blocks that are laid out in the perpendicular direction to the quay. Each block consists of hundreds of container stacks several tiers high, where the stacks are arranged in dozens of bays in the perpendicular direction and in several rows in the horizontal direction. Since a bay is of the length of a 20 ft container, a stack of 40 ft container spans two consecutive bays. For safety reasons, the containers cannot be stacked together if their sizes are not the same. Each block is equipped with two automated stacking cranes (ASC) to handle the containers. Since the two ASCs are of the same size, one cannot move across the other and thus may interfere with each other at close ranges. The container transfer to and from an AGV is made by the seaside

ASC at a seaside handover point (HP) located at the seaside end of the block, while the transfer to and from an ET is made by the landside ASC at a landside HP located at the opposite end.

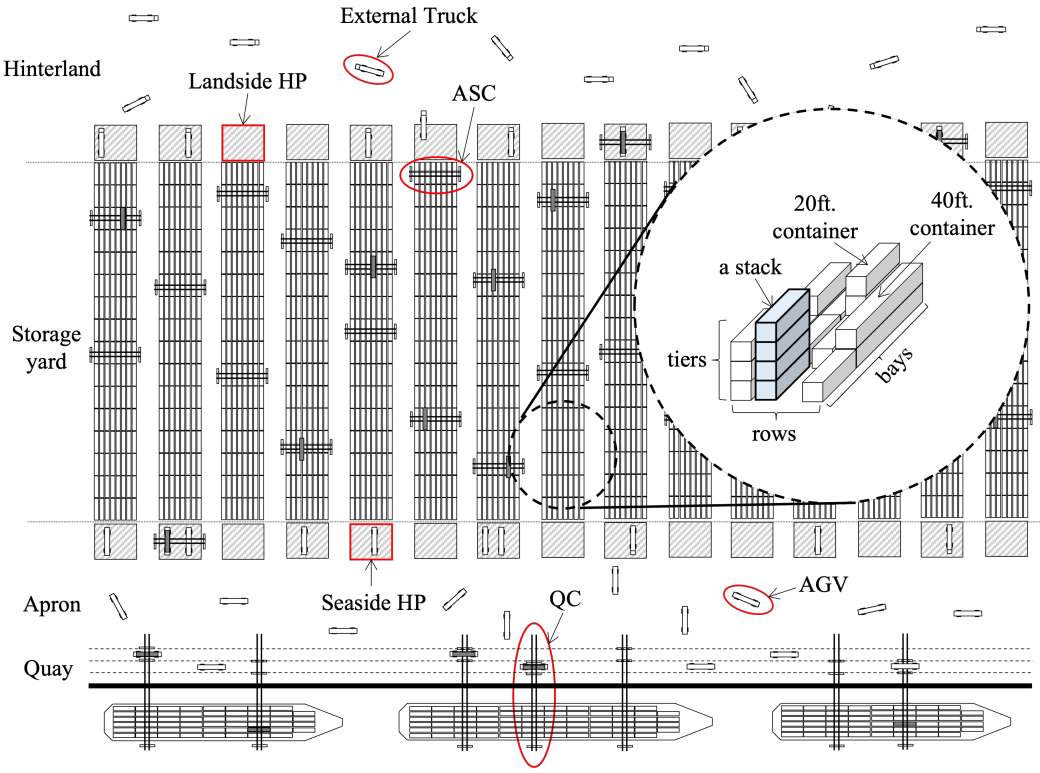


Figure 1. Layout of an automated container terminal.

The containers in the storage blocks are categorized into three groups by their directions of the flow of logistics: inbound, outbound, and *transshipment* containers. The inbound and outbound containers are already mentioned in the previous section. The transshipment containers are those that are unloaded from certain vessels and stored in the yard, but unlike the inbound containers, they are to be loaded onto other vessels for further sea transportation. A container stored in a block is often relocated to some other place within the block. If container X is placed on top of container Y in a stack but Y has to be taken out earlier than X, then X has to be moved to another location before Y can be retrieved. This relocation, or rehandling, is hard to avoid because the retrieval schedule of containers is usually unknown at the time the containers arrive at the yard and are piled up. Thus, we need a good policy that can select good stacking locations for the incoming containers so that both the container handling time and the possibility of rehandling are minimized. Note that a stacking policy should also be able to recommend good stacking locations for the rehandled containers to minimize further rehandling. Rehandling is considered the biggest cause of delay of container handling in the storage yard.

There arises another difficulty with container handling when the two ASCs of a block cannot move across each other. To avoid collision, an ASC sometimes has to stop and wait until the other one finishes its job and backs up. This *interference* deteriorates the throughput of the ASCs. Interferences are more likely to occur as the travel distance of an ASC becomes longer because it gets close to the other ASC with a higher probability. Unfortunately, the ASCs often cannot avoid long-distance travel in such a block layout as that shown in Figure 1. The outbound containers brought in by the ETs enter the block through the landside HPs and are usually stored at locations near the landside end. However, they eventually go out of the block through the seaside HPs when they are loaded onto their target vessels. On the other hand, the inbound containers unloaded from the vessels

enter the block through the seaside HPs and then later exit through the landside HPs. These long-distance movements of containers in opposite directions easily lead to ASC interferences. One way of minimizing such interference is to have the containers tactically relayed through cooperation between the two ASCs. As an example, an inbound container stacked near the seaside end of the block can be moved first to a certain intermediate location by the seaside ASC and then to its final destination HP at the landside by the landside ASC. The first movement of this relay operation is called *repositioning*. Note that the stacking locations for the containers to be repositioned should also be determined by the stacking policy.

Given a stacking policy, container handling in a storage block is typically performed in the following way. Whenever an ASC finishes its current job, it selects the most urgent job from the job queue that contains all the jobs requested for the next horizon of length, say thirty minutes. The jobs in the queue include those to be undertaken according to the loading/unloading schedules and those requested from the ETs that have already arrived at the landside HPs but have not been serviced yet. The ETs expected to arrive at the block during the next horizon are not counted because their arrival time is highly unpredictable. If the current ASC is the landside ASC, the most urgent job would be the ET job with the longest waiting time. If the current ASC is the seaside ASC, the most urgent job would be either a loading or an unloading job with the earliest deadline according to their schedules. When the job selected is a stacking job, the stacking policy examines all the available slots in the block and recommends the best one as the stacking location. An available slot can be found at the top of every stack unless it has already reached the allowed maximum tier. However, the stack should not belong to the bay where the other ASC is currently working. When the job selected is a retrieval job, no reference to the stacking policy is necessary unless there are other containers above the target container. If there are some containers above, they all must be relocated one after another to the locations recommended by the stacking policy. When the retrieval job selected requires a travel distance longer than a given threshold, the target container should be repositioned to the location recommended by the stacking policy before it can be sent to its destination HP by the other ASC.

The efficiency of ASC operation in a block can be assessed by measuring the AGV delay and the ET waiting time, where the former is counted far more important than the latter. A good stacking policy makes the ASC operation efficient by reducing the interference, rehandling, and the overall container handling time, which makes it possible for the ASCs to provide the AGVs and ETs with faster services. However, it takes quite a long period of observation to see how good a stacking policy is. Whether or not the stacking locations recommended are good can be seen better when the containers are retrieved out of the block than when they come into the block. Since the average dwell time of containers in container terminals is often longer than a week, the AGV delay and ET waiting time should be measured for a long period during which time enough containers are retrieved, under the condition that the containers keep coming in and going out fairly constantly during that period of time.

3. Related Works

There are some previous works on container stacking that deal with the problem of allocating the storage spaces for incoming containers. Reference [9] showed how to organize the storage area to minimize the number of container handling moves given a fixed amount of space, based on simple models that capture the relationship between the handling moves and the amount of available space. Reference [10] developed a space allocation method for inbound containers so as to minimize the expected number of rehandles. References [11,12] used a mixed-integer programming model together with some heuristics to allocate storage space for outbound and transshipment containers, respectively. Reference [13] applied a constraint satisfaction technique to allocate spaces to outbound containers. Reference [14] developed a space allocation method that can cope with the uncertainties in loading/unloading times of vessels. Reference [15] used a genetic

algorithm to optimize the space allocation problem to avoid bottlenecks in storage yard operations and to minimize vessel service time. All these methods allocate a bulk of storage locations for reservation prior to the arrival of containers. If the arrival plan changes, the storage space must be reallocated. In contrast, the methods discussed below designate specific storage locations for each individual container at the time of its arrival.

The majority of the previous works on stacking locations have used rules or heuristics. Reference [16] considered the configuration of the container stack and the weight distribution of containers in the yard-bay to derive a decision tree model that determines the storage location of each outbound container. The decision tree can be deemed as another representation of a set of rules. Reference [1] proposed stacking rules that recommend the containers belonging to the same category be stacked together. Containers of the same category are of the same weight class and size, have the same destination port, and are loaded onto the same vessel. Reference [2] proposed rules that consider not only the category but also the height of the stacking position. Reference [3] suggested a heuristic rule for determining the locations of relocated containers to minimize the number of relocations during the retrieval process. Reference [17] conducted simulation studies to investigate the effect of using information about container departure times and the tradeoff between stacking farther away versus stacking close to the HPs. It used simple stacking rules that are designed to work in perpendicularly laid out storage blocks in an automated container terminal. Reference [18] presented what they call a hybrid sequence stacking method that determines the stacking locations of outbound containers considering the container weights. Proposing an ideal configuration of a yard bay to avoid rehandling, this method tries to stack the incoming containers so that their positions are as close to the ideal configuration as possible.

Some previous works tried to adopt more AI (artificial intelligence) techniques based upon rules or heuristics. Reference [19,20] derived stacking policies for outbound containers considering the uncertainties in their weights with the purpose of minimizing rehandling. Their policy consists of three precedence rules, each for a container weight group, where the rules are optimized by a simulated annealing algorithm. Reference [21] proposed a heuristic method to stack outbound containers. This method evaluates each candidate stacking location for an incoming container through a simulation and selects the best one. In the simulation, after stacking the container at the candidate location, the remaining containers arriving in a random sequence are stacked following a heuristic priority rule and the resulting performance is measured. Reference [22] used simple rules that determine the stacking positions based on the stack height and the estimated time of retrieval. The rules adopt fuzzy logic to represent their conditions to deal with a high degree of uncertainty in the arrival of containers at the yard. Reference [23] proposed a multi-agent system for container stacking, in which the stack agent recommends a stacking position by consulting the knowledge base composed of if-then rules that check various conditions such as the container types, the configuration of the storage space, and the occurrence of exceptional events. Each stacking decision is evaluated by the evaluation agent that rejects the decision when unacceptable. If a decision turns out to be unacceptable, a learning mechanism is activated to add a new rule to the knowledge base so that the rules responsible for the wrong decision can be disabled. This learning mechanism makes the proposed system adaptive to changes, while the adaptation is mainly focused on the disturbances and unexpected events. Reference [24] investigated the impact of container stacking methods regarding how they deal with uncertainties in container terminals and reduce container handling costs. The stacking methods studied, however, determine only the best yard-bays but not the specific stacking slots.

Compared to the works discussed above, References [4,6] are much more closely related to our work. The stacking policy in these researches employs scoring functions that evaluate a candidate stacking location from various perspectives using different criteria, where the score of a location is calculated as the weighted sum of the scores for those criteria. The policy uses different scoring functions for different container types because

each container type requires its own evaluation criteria for stacking. However, all those scoring functions with different weight vectors are together treated as a single policy. This policy is optimized by a search using a GA, where a candidate policy is evaluated by simulating the operations at a block under the policy for a certain period of time and measuring the resulting performance. For this simulation, Reference [4] provided a pool of operation scenarios of various kinds for a more accurate policy evaluation. To evaluate a policy, it is applied to a randomly selected subset of those scenarios, and the resulting performances were averaged. Therefore, the policy thus optimized can be said to be the best on average. Given a certain situation, there may be some other policy that works better than the on-average best policy. Another limitation with such policy is that the policy cannot change as the operational environment changes. As an effort to overcome this problem, Reference [5] proposed an online search algorithm that dynamically adjusts and optimizes a stacking policy by continuously generating variants of stacking policies and evaluating them while they are actually being applied for determining the stacking positions. However, this online search cannot keep up with the rapid changes in a situation. When the situation changes, the performance of the current policy begins to deteriorate, at which time that of a variant of the current policy may show a better performance. If this happens, the current policy would be switched to that variant, but only after experiencing some deterioration. In general, we cannot expect such a good variant to appear quickly at the right moment. Therefore, we can say the online search is not really reactive to changes, but it just gradually adapts to changes. Another drawback is that the online search cannot find really good policies because of its limited explorative capacity. Since all the variants must be simulated and tested online, it is hard to generate and test a number of variants under a real-time constraint, which deteriorates the search performance.

The stacking policy derived by the method proposed in this paper is a significant improvement on the policy proposed in [4,6]. While [4,6] look for a policy whose average performance in various situations is the best, our method derives a policy that can quickly adapt to changing situations. When the situation changes, our policy immediately reacts to the change and provides a newly customized action. This improvement was possible because the policy that we deal with was based on scoring functions and the weight vectors used in those functions were easily adjustable. Most previous works reviewed above in the second and the third paragraphs of this section use policies based on if-then rules. Those rules are carefully crafted by the designers rather than being optimized by any algorithms. They are hard to be automatically modified upon situation changes. While the method proposed in [5] looks closest to ours in that the policy can adapt to changing situations, its adaptation is slow or gradual rather than immediate or reactive. The different characteristics of the related works discussed so far are compared and summarized in Table 1. The works reviewed in the first paragraph of this section are not included in the table because they have little relevance to our work.

Table 1. Comparison of the proposed method with the related works that determine the stacking locations for individual containers.

Characteristics	Related References and Their Characteristics				
	[1–3,16–18,21–24]	[19,20]	[4,6]	[5]	Proposed Method
Rule-based	Yes	Yes	No	No	No
Score-based	No	No	Yes	Yes	Yes
Optimized	No	Yes	Yes	Yes	Yes
Situation adaptive	No	No	No	Yes	Yes
Reactive to changes	No	No	No	No	Yes

4. Stacking Policy Based on Scoring Functions

This section describes the stacking policies proposed by [4,6]. The stacking location of an incoming container is determined in two stages. First, the container is assigned to a

block in the storage yard taking into account various operational conditions in all of the blocks. Second, a specific location within the assigned block is selected from among the candidate locations based on several criteria such as the distance to the destination, the stack height, the likelihood of rehandling, and so on. The stacking policies described in this paper are used in the second stage to determine a specific stacking location within the designated block.

To determine a stacking location within a block, all the available slots in the block are evaluated by using a scoring function and then the one with the best score is chosen. Note that the slot determination is required not only for the containers newly coming into the block but also for those that are rehandled or repositioned within the block. Furthermore, a good target slot can be different depending on whether the container is an inbound, an outbound, or a transshipment container. Table 2 shows that the stacking policy uses different scoring functions for different container types. The score $s_i(x)$ of a slot x for the ith container type is calculated by the weighted sum given below:

$$s_i(x) = \sum_i w_{i,j} C_{i,j}(x) \tag{1}$$

where $C_{i,j}(x)$ is the evaluation value of slot x according to the jth criterion for the ith container type and $w_{i,j}$ is the weight for $C_{i,j}$. The stacking policy of Table 2 consists of seven scoring functions, each of which employs a different subset of eight criteria. Notice that the decision by the policy may change as the values of the weights of the criteria change.

Table 2. Scoring function for each of the seven container types.

Container Type		Scoring Function
Incoming	Inbound	$s_1 = w_1 D_{to} + w_2 D_{from} + w_3 H + w_4 E + w_5 S$
	Outbound	$s_2 = w_6 D_{to} + w_7 D_{from} + w_8 H + w_9 E + w_{10} S + w_{11} G + w_{12} P$
	Transship	$s_3 = w_{13} D_{to} + w_{14} H + w_{15} E + w_{16} S + w_{17} G$
Rehandle	Inbound	$s_4 = w_{18} D_{to} + w_{19} D_{from} + w_{20} H + w_{21} E + w_{22} S$
	Outbound and Transship	$s_5 = w_{23} D_{to} + w_{24} D_{from} + w_{25} H + w_{26} E + w_{27} S + w_{28} G$
Reposition	Inbound	$s_6 = w_{29} D_{to} + w_{30} D_{from} + w_{31} H + w_{32} E + w_{33} S + w_{34} T$
	Outbound and Transship	$s_7 = w_{35} D_{to} + w_{36} D_{from} + w_{37} H + w_{38} E + w_{39} S + w_{40} G + w_{41} T$

The criterion D_{to} is the distance to the candidate stacking location from the current location of the target container, and D_{from} is the distance from the candidate location to the outgoing HP of the container. These two criteria are calculated differently for different types of containers, as illustrated in Figure 2. D_{to} and D_{from} give the same value if the target container is a transshipment container. H is the height of the stack underneath the candidate location. The higher the stack, the greater the likelihood of rehandling. E is an indicator of whether or not the candidate location is an empty ground. The empty grounds have to be saved as much as possible in preparation for the possible shortage of stacking locations. S is the amount of reduction in empty ground slots available for container stacking. For safety reasons, containers of different sizes cannot be stacked together. A 40 ft container occupies two adjacent stacks, as shown in Figure 1. Therefore, placing a 20 ft container on one of the two consecutive empty ground slots not only uses one ground slot for a 20 ft container but also reduces the availability of ground slots for 40 ft containers.

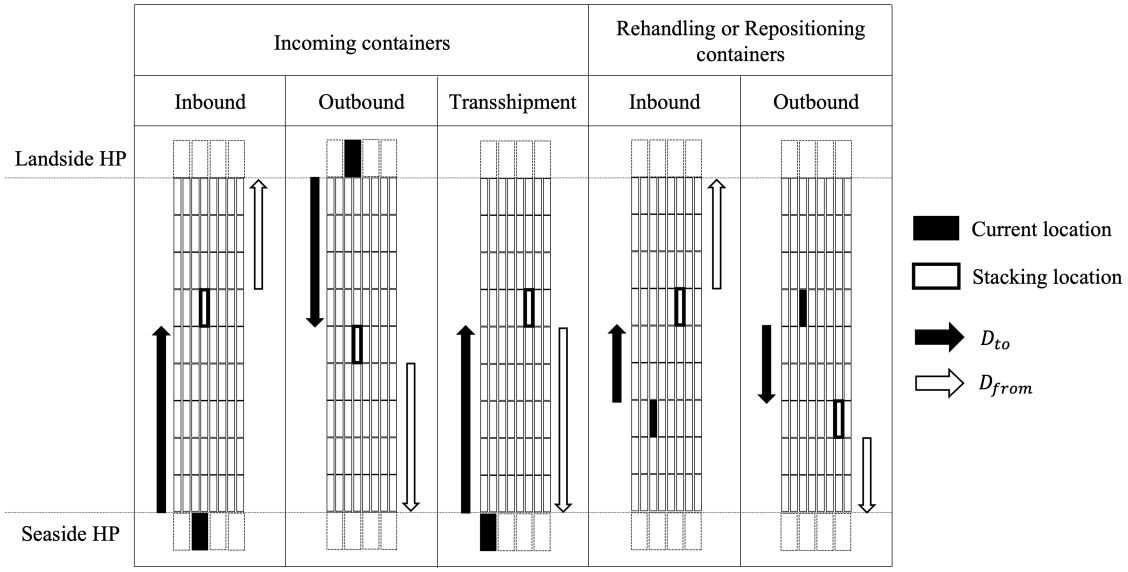


Figure 2. D_{to} and D_{from} depending on the type of container.

T indicates whether or not the container right underneath the candidate location has been temporarily repositioned to that slot. T is used only in the scoring functions for the containers to be repositioned. Since a repositioned container will soon be moved to an outgoing HP, rehandling is quite likely to occur if there is some other container on top of it. However, if the container on top of it is also a repositioned container, rehandling can be avoided by simply moving the upper one to its own outgoing HP before moving the lower one. Criterion T encourages the repositioned containers to be kept together in the same stacks. This is desirable to save stacking slots available for the containers newly coming in or for those rehandled. G is the estimated likelihood of the occurrence of rehandling when a container to be loaded to a vessel is stacked on top of others. If the container was just stacked and all the containers underneath belong to the same category, no rehandling occurs during loading. Otherwise, the underneath containers belonging to different categories can cause rehandlings. As mentioned before, the containers belonging to the same category are to be loaded onto the same vessel, have the same port of destination, are of the same size, and are of the same weight class.

P appears only in the scoring function s_2, which is specialized for the outbound containers brought in by the ETs. It is the preference value of a candidate location depending on which region of the block the location belongs to. It is preferable that the outbound containers to be loaded sooner are stacked closer to the seaside. Figure 3 illustrates the distributions of the preferences over different regions in a block for the outbound containers of different loading times. The block is divided into five regions, and there are three different urgency levels for loading. The preferences are distributed differently for each urgency level, resulting in fifteen preference values each for a region and an urgency level. In the previous work [6], these preference values as well as all the weight values in Table 2 were determined by running a GA-based search algorithm.

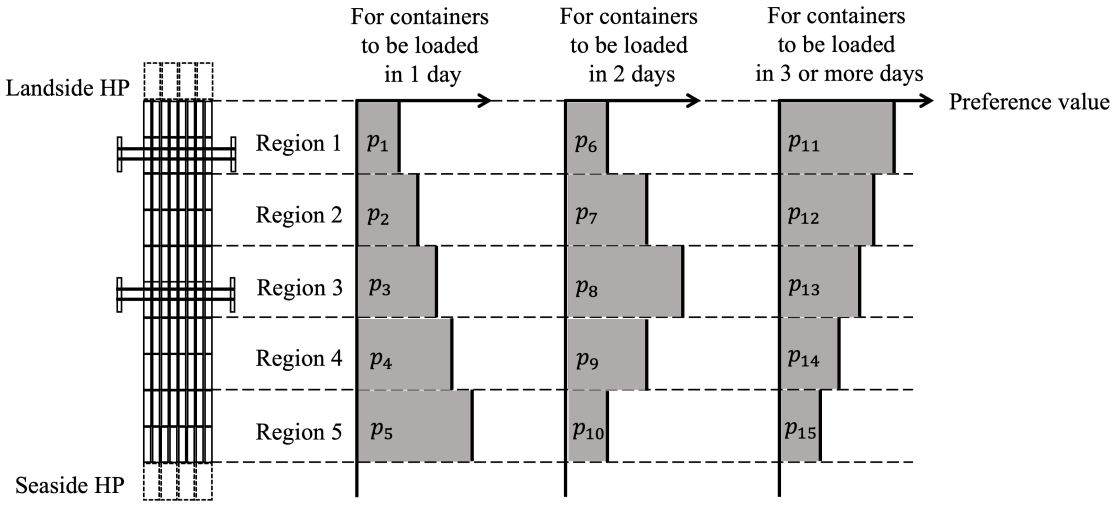


Figure 3. An example of preference distributions over the stacking regions within a block for the outbound containers with different loading times.

Table 3 summarizes the eight criteria explained above. To calculate the weighted sum of the respective subsets of these criteria, the policy of Table 2 uses 41 weights. The evaluation values of all the criteria are normalized to [0, 1] and the weight values are constrained to [–1, 1]. Note that a weight can be negative if the value of the corresponding criterion affects the policy adversely. As the decision made by the policy depends on the weight combination or the weight vector, the weight vector is considered as the policy. When the policy is optimized by using a GA, each candidate policy is evaluated by applying it to a variety of scenarios of operations in a block and averaging the resulting performances. In this way, the optimization algorithm derives a policy that works the best on average. Given a certain situation, however, a different policy might perform better than this best on-average policy. In the next section, we explain how we derive a policy that can be dynamically adapted to changing situations.

Table 3. Evaluation criteria used in the scoring functions.

Criterion	Description
D_{to}	Distance to the stacking location from the container pick-up position
D_{from}	Distance from the stacking location to a departure HP
H	Height of the stack at the candidate location
E	Indicator for an empty ground
S	Reduction in empty space availability
P	Regional preference within a block for the incoming outbound containers
G	Likelihood of rehandling
T	Indicator of having a repositioned container underneath

5. Proposed Method

In this research, we consider the current workload of vessel operation as the only important indicator of the current operational situation in a storage block. The workload of vessel operation from the standpoint of a block is the workload of its seaside ASC that handles the containers to be loaded to or unloaded from the vessels. Since any delay by the seaside ASC leads to a delay of the vessel operation at the quay, its efficient operation is critical. It may not be desirable, for example, that the seaside ASC spends too much time in container stacking when the seaside workload is high. The containers arriving at the seaside HPs are better stacked at locations not far from those HPs in order not to have other vessel operations delayed. However, a desired amount of such adjustment of travel distance for stacking cannot be made in any obvious way. It is difficult to invent a formula

relating the amount of adjustment with the seaside workload that is continuously changing over time. One intuitive approach to dealing with continuously changing situations would be to divide the situations into a finite number of representative situations and then to derive a specialized policy for each representative situation. Note that a coarse division would not be effective enough because each constituent policy would suffer from the same problem of showing only the best on-average performance although to a lower degree. On the other hand, a very fine division requires too many policies that all must be derived through computationally expensive optimization search.

Another approach one may think of would be to turn the criteria used in the scoring functions of the policy into functions of the seaside workload. While most criteria are clearly independent of the seaside workload, the regional preference P seems to be dependent on it. When the seaside workload is heavy, preferring the seaside regions is not desirable because the chances of interference with the seaside ASC get high. In fact, the preference as a function of the seaside workload seems necessary more for the inbound than the outbound containers because the seaside ASC may not want to travel a long distance for container stacking when its load is heavy. Furthermore, there are some weights whose desirable values seem to depend on the seaside workload, although the corresponding criteria are not. Some of the examples are the weights for D_{to} and D_{from}. The value of criterion D_{to} for an inbound container should be considered more importantly (i.e., should be given a larger weight) to save the travel time as the seaside workload gets higher. However, we do not know how exactly the values of the criteria or weights should change as a function of the seaside workload. In our proposed method, therefore, we exclude P from the policy, and instead, we synthesize a new policy from two boundary policies whenever needed: one for a very low workload and the other for a very high workload. For the synthesis, we take an interpolation of the two boundary policies. When we use a GA to search for the two boundary policies, we simultaneously search for the two threshold values to quantify the two fuzzy terms 'very low' and 'very high'.

Let s represent the current workload that is measured by adding up the estimated processing times of all the vessel jobs scheduled to be done by the seaside ASC within the next horizon of length h seconds from the current point of time. A vessel job is either a loading or an unloading job. For a loading job, the seaside ASC makes an empty trip from its current location to the location of the target container, picks up the container, makes a loaded trip to a seaside HP, and puts the container down on top of an AGV waiting there. Among these actions, container pickup can take longer if it involves rehandlings. For an unloading job, the seaside ASC undertakes an empty trip from its current location to the seaside HP where the AGV bringing the target container is parked, picks up the container from the AGV, undertakes a loaded trip to the designated stacking location, and puts the container down at that location. Since the loading and unloading schedules for each vessel are predetermined at the planning stage well before the real operation starts, the workload of vessel operations within a horizon can be easily estimated.

We use θ_l and θ_h to denote the threshold values for the extreme or boundary workloads; the workload is said to be very low if $s \leq \theta_l$ and very high if $s \geq \theta_h$. The seaside ASC is said to be overloaded if $s > h$, as the time taken to finish the works planned for the horizon exceeds the length of the horizon. Let π_l and π_h be the policies specialized for the situations of very low workload and very high workload, respectively. Then, the policy π_s for workload s with $\theta_l < s < \theta_h$ can be synthesized from π_l and π_h by deriving new weight values to be used in the scoring functions of π_s through interpolations between the corresponding weights in π_l and π_h. The ith weight $w_{s,i}$ to be used in policy π_s is calculated as

$$w_{s,i} = \frac{(\theta_h - s)w_{l,i} + (s - \theta_l)w_{h,i}}{\theta_h - \theta_l} \tag{2}$$

where $w_{l,i}$ and $w_{h,i}$ are the ith weights in π_l and π_h, respectively. As s gets closer to θ_h, $w_{s,i}$ is influenced more by $w_{h,i}$ than $w_{l,i}$, or the other way around. Note that the score $\pi_s(x)$ for

a candidate slot x by the synthesized policy π_s can be directly calculated from the scores $\pi_l(x)$ and $\pi_h(x)$ as

$$\pi_s(x) = \frac{(\theta_h - s)\pi_l(x) + (s - \theta_l)\pi_h(x)}{\theta_h - \theta_l} \tag{3}$$

without actually deriving the individual weights constituting π_s because the values θ_h, θ_l, and s in Equation (2) are independent of i.

We use a GA for our optimization, which is basically the same as that used in [4]. We optimize not only π_l and π_h but also the two threshold values θ_l and θ_h. Figure 4 shows the representation of the candidate solution adopted by our GA. Since each policy consists of 40 weights after dropping out the criterion P, there are 82 real values to be optimized in total, where v_l and v_h are constrained to be in [0, 2] and the weight values in [–1, 1]. During the evaluation, v_l and v_h are decoded to θ_l and θ_h, respectively, by having them multiplied to the length of horizon h. The reason for setting the upper bound of v_l and v_h to 2 is that the workload of vessel operation measured in time can exceed the length of the horizon when overloaded.

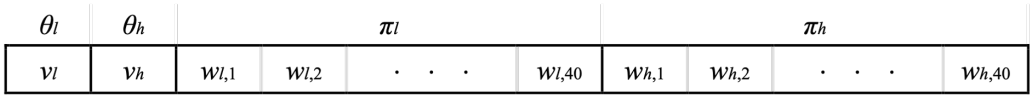


Figure 4. Representation of candidate solution.

To evaluate a candidate policy during the search, the policy is applied through simulation to a set of scenarios randomly chosen from the provided pool and the resulting performances are averaged. The pool contains various scenarios of different difficulty levels; a scenario is difficult if the workloads of the ASCs are high. The length of a scenario is three weeks, which is long enough to measure the efficiency of the ASC operation because enough of the containers that arrived during this period are retrieved. During the first two weeks, the stacking yard, or block, is initialized starting from an empty yard without simulating the ASC's movements. Then, from the beginning of the third week, the efficiency of the ASC operation is measured with their movements simulated realistically, reflecting acceleration, deceleration, and interferences. More details on this crane simulation can be found in [25]. For an evaluation of the performance in the third week of a scenario, a candidate policy $\langle \theta_l, \theta_h, \pi_l, \pi_h \rangle$ is applied to the scenario. When a container has to be stacked, the workload s of vessel operation for the next horizon of length h seconds is estimated. If $s \leq \theta_l$ or $s \geq \theta_h$, then π_l or π_h becomes the policy to be used, respectively. Otherwise, a new policy π_s specialized for the workload s is synthesized through interpolation and then applied for stacking. Note that we need to synthesize only one of the seven scoring functions shown in Table 2 depending on the type of container to be stacked. This synthesis and application of a new policy are repeated every time a container is stacked. When the simulation of a scenario is over, the performance of the candidate policy is measured by the following objective function:

$$f(\pi) = W_1 \cdot D_{AGV}(\pi) + W_2 \cdot D_{ET}(\pi) \tag{4}$$

where π is the stacking policy under evaluation, $D_{AGV}(\pi)$ is the average (per container) AGV delay observed under π, $D_{ET}(\pi)$ is the average waiting time of ETs under π, and W_1 and W_2 are the respective weights for D_{AGV} and D_{ET}. W_1 is usually much larger than W_2 because the seaside operations are considered much more important than the landside operations. The final evaluation is obtained by averaging the objective values measured from all the scenarios.

6. Experimental Results

We used the algorithm named NTGA for policy optimization, which is the same one as that used in [4] to derive the static stacking policy described in Section 4. Our

parameter setting of NTGA is shown in Table 4. Since NTGA requires a random subset of operation scenarios selected from a pool to evaluate each candidate policy, we generated 1000 scenarios to constitute a pool. Each scenario consists of container handling jobs to be completed for three weeks in a block that is 46 bays long, 8 rows wide, and 5 tiers high. The job requests are made by the AGVs and ETs that arrive at the seaside and landside HPs, respectively. They either bring in a container to be stored in the block or ask for a container to be picked up from the block. The average number of AGVs arriving per day is approximately from 220 to 300, and that of ETs is from 70 to 110. More requests are from the AGVs than ETs because there are transshipment containers whose proportion among the containers unloaded from the vessels is about 50% in our scenarios. The average daily workload of the ASCs increases with the number of requests, but the workload continuously changes within a day as the requests are not evenly distributed over time.

Table 4. Parameter setting of NTGA in our experiments.

Parameter	Setting Value
Population size	100
Mating pool size	2
Buffer pool size	50
Sampling size	2 (initially 4)
Crossover operator	Simulated binary crossover
Crossover probability	0.9
Mutation operator	Polynomial mutation
Mutation probability	0.01
Number of evaluations	100,000

Using this pool of scenarios, we derived both the static stacking policy of [4] and the dynamic stacking policy proposed in this paper. Then, the two policies were applied to 100 scenarios that were separately generated following the same distribution as that used for generating the scenarios of the above pool. The two weight values W_1 and W_2 of Equation (4) were empirically set to 50 and 1, respectively, not only when the policies were derived but also when they were tested. The results obtained by measuring the AGV delay and ET waiting time are shown in Table 5. We can see that the proposed policy outperforms the static policy in terms of both AGV delay and ET waiting time. The improvement is 22.3% for AGV delay and 16.3% for ET waiting time. We also confirmed that the proposed policy performs significantly better than the static policy by using a paired t-test with a confidence level of higher than 99.99%. The data for our experiments and the execution of our program can be found in [26].

Table 5. Performance comparison of the two policies.

	Static Policy	Dynamic Policy
AGV delay	80.0	62.2
ET waiting time	285.3	238.8

To obtain some hints about how the dynamic policy works, we investigated the behaviors of the two boundary policies, i.e., π_l for a very low workload and π_h for a very high workload. The two threshold values θ_l and θ_h (see Figure 4) found by our search algorithm for distinguishing the very low and very high workloads were 274 s and 1892 s, respectively. Recall that the length of our horizon is 1800 s; the workload of 274 s really looks very low, and that of 1892 s is over the capacity. If we want to quantify the overall stack preference of a boundary policy π_b for the ith container type (see Table 2 for the seven container types), we apply π_b to a scenario as if it is a static policy and pay special attention to the moments of stacking the ith type containers. Whenever we come across such a moment during the simulation, we not only apply π_b to stack the container at the

best location as usual, but also calculate the scores for all of 46 × 8 slots as if they are all candidate locations and just save the scores separately. For the latter calculation, the constraint of a maximum possible tier of five is relaxed. Furthermore, we assume that we can stack 20 ft containers even on top of 40 ft ones. When the simulation is over, we obtain the stack preference by averaging the separately saved scores for every slot in the block.

Figure 5a compares the stack preferences of π_l and π_h for the incoming inbound containers, where the slots of better scores are indicated by a darker shade. When the seaside workload is very low, the best locations by π_l are distributed toward the landside end. This is quite reasonable because the inbound containers will eventually leave the block through the landside HPs. Since the seaside workload is low, the seaside ASC does not hesitate to travel a long distance to stack the containers at the landside end so that later retrieval by the ETs is expedited. However, we can see that the locations toward the seaside end of the block are considered not the worst but somewhat preferable by π_l. Note that π_l is used not only in the situations of very low workloads but also in the situations of intermediate workloads through interpolation with π_h. If we separately derived a static policy specialized for a very low workload, it might not prefer any seaside locations at all. On the other hand, when the seaside workload is very high, π_l prefers the locations closer to the seaside end than those farther away. When the seaside ASC is very busy, it should avoid long-distance travel as not to delay the services to the AGVs waiting at the seaside HPs. Figure 5b shows the overall stack preferences of π_l and π_h for the incoming outbound containers. We can see that the stack preferences are almost the opposite of what we have seen for the inbound containers. Figure 5c shows the stack preferences of the static policy for the inbound (shown in the upper part) and outbound (shown in the lower part) containers. It seems that the static policy generally prefers the locations closer to the departure HPs regardless of the seaside workload.

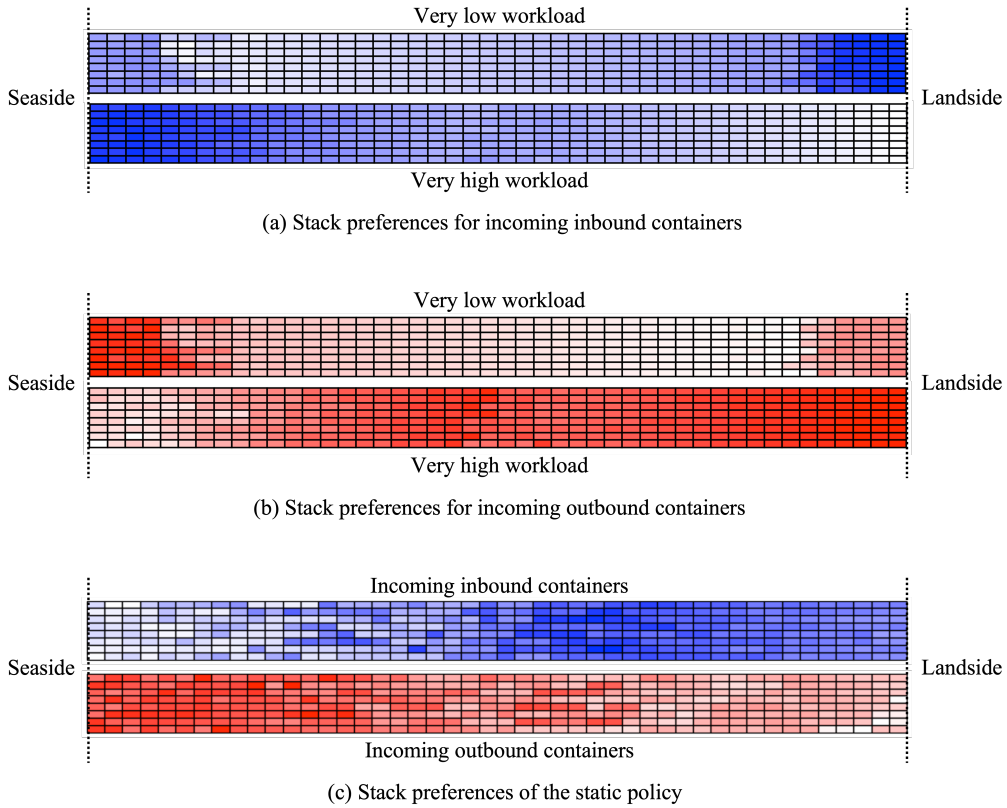


Figure 5. Comparison of the stack preferences.

7. Cases with Multiple Situation Indicators

Thus far, we have been assuming that the seaside workload is the only indicator we consider to represent the situation. Although rare practically in container terminals, we can imagine the cases in which the situation is represented by multiple indicators. Our method described in Section 5 can be extended to cover such cases by generalizing the interpolation to a weighted average of multiple relevant terms. Consider, for simplicity of explanation, the case with two indicators s_1 and s_2. We need two threshold values for each indicator to distinguish between very low and very high values, i.e., $\theta_{1,l}$ and $\theta_{1,h}$ for s_1, and $\theta_{2,l}$ and $\theta_{2,h}$ for s_2. This leads to four extreme or boundary situations $B_{l,l}$, $B_{l,h}$, $B_{h,l}$, and $B_{h,h}$, where $B_{l,l}$ is the set of situations with $s_1 \leq \theta_{1,l}$ and $s_2 \leq \theta_{2,l}$, $B_{l,h}$ with $s_1 \leq \theta_{1,l}$ and $s_2 \geq \theta_{2,h}$, $B_{h,l}$ with $s_1 \geq \theta_{1,h}$ and $s_2 \leq \theta_{2,l}$, and $B_{h,h}$ with $s_1 \geq \theta_{1,h}$ and $s_2 \geq \theta_{2,h}$. We use $\pi_{l,l}$, $\pi_{l,h}$, $\pi_{h,l}$, and $\pi_{h,h}$ to denote the policies specialized for the boundary situations $B_{l,l}$, $B_{l,h}$, $B_{h,l}$, and $B_{h,h}$, respectively. Given an intermediate situation s other than those boundary situations, the policy π_s for s can be synthesized from $\pi_{l,l}$, $\pi_{l,h}$, $\pi_{h,l}$, and $\pi_{h,h}$ by taking a weighted average after normalizing the indicator values. The normalization is necessary to compensate the different scales of different indicators.

Figure 6a represents the space of all situations on a two-dimensional plane formed by two coordinates, one for indicator s_1 and the other for indicator s_2. We can see how the areas of the four boundary situations $B_{l,l}$, $B_{l,h}$, $B_{h,l}$, and $B_{h,h}$ are located in relation to the threshold values of the two indicators. A and B in the figure are two situations other than the boundary situations. A is an intermediate situation whose indicator values do not go over any threshold. B is not quite an intermediate situation because one of its indicator s_2 takes a value below the lower threshold $\theta_{2,l}$. Figure 6b shows the situations in Figure 6a after a normalization, where each indicator value s_i is transformed to $(s_i - \theta_{i,l})/(\theta_{i,h} - \theta_{i,l})$. In Figure 6b, the areas of boundary situations are marked by the corresponding policies, and the distances to those areas from A and B are indicated by $d_{i,j}$'s. Let π_A denote the policy for situation A in the figure. Then, the score $\pi_A(x)$ for a candidate slot x in situation A can be calculated by a weighted average of the scores given by $\pi_{l,l}$, $\pi_{l,h}$, $\pi_{h,l}$, and $\pi_{h,h}$:

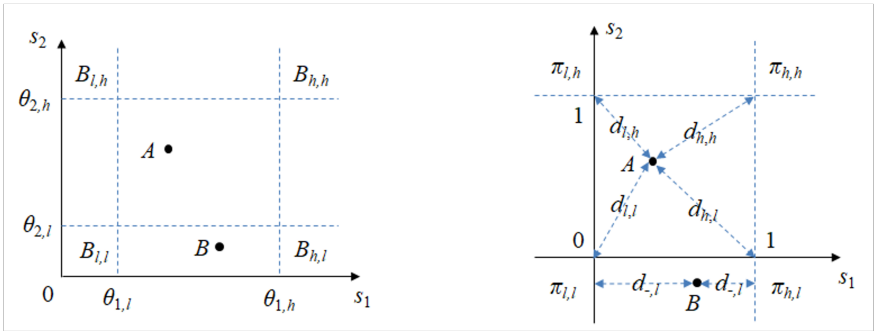
$$\pi_A(x) = \frac{\frac{1}{d_{l,l}(A)}\pi_{l,l}(x) + \frac{1}{d_{l,h}(A)}\pi_{l,h}(x) + \frac{1}{d_{h,l}(A)}\pi_{h,l}(x) + \frac{1}{d_{h,h}(A)}\pi_{h,h}(x)}{\frac{1}{d_{l,l}(A)} + \frac{1}{d_{l,h}(A)} + \frac{1}{d_{h,l}(A)} + \frac{1}{d_{h,h}(A)}} \tag{5}$$

where $d_{i,j}(A)$ is the distance from A to the area of $\pi_{i,j}$, and $1/d_{i,j}$ can be interpreted as the respective closeness. If π_B denotes the policy for situation B, the score $\pi_B(x)$ for a candidate slot x in situation B can be similarly calculated as

$$\pi_B(x) = \frac{\frac{1}{d_{-,l}(B)}\pi_{l,l}(x) + \frac{1}{d_{-,h}(B)}\pi_{h,l}(x)}{\frac{1}{d_{-,l}(B)} + \frac{1}{d_{-,h}(B)}} \tag{6}$$

where terms related to $\pi_{l,h}$ and $\pi_{h,h}$ are not included because s_2 is extremely low in B and thus they are irrelevant. Note that Equation (6) is equivalent to the linear interpolation we calculated in Equation (3).

The formulation given above can be easily extended to the cases with more than two situation indicators in principle. However, extensions to such cases would be practically infeasible because the number of boundary policies increases exponentially to 2^n, where n is the number of situation indicators. As we have seen in Figure 4, our chromosome for the search of the policy already consisted of 82 real-numbered genes when there was a single indicator. If there were two situation indicators, the number of genes should have increased to 164. This number doubles each time another indicator is added, resulting in a huge search space.



(a) Before normalization (b) After normalization

Figure 6. Situation space displayed on a two-dimensional plane formed by the two coordinates of the situation indicators s_1 and s_2.

8. Concluding Remarks

We derived a dynamic or situation-adaptive stacking policy and compared its performance with that of a static policy that is insensitive to the situation change. The result of simulation experiments tells us that the dynamic policy clearly outperforms the static policy. In the experiments, the policy was optimized using randomly generated operation scenarios and then tested using a different set of scenarios separately generated from the same distribution. This separation was an effort to guarantee the generality of the observed performance of the policy in similar situations. However, the policy needs to be newly optimized and tested for being applicable to a new container terminal whose distribution of the operations is different from that of our target container terminal.

The way we proposed to obtain a dynamic policy is to synthesize a new policy from two boundary policies: one for a very low and the other for a very high workload. For the synthesis, we recommended a linear interpolation of the two boundary policies. We used a GA-based search algorithm to optimize not only the two boundary policies but also the two threshold values for distinguishing the very low and very high workloads. The idea of using linear interpolation is based on the assumption that the values of the weight vectors in the policy are linearly related to the seaside workload, which is the only indicator we consider to represent the situation. Although this linearity assumption might not be really correct, our empirical study has shown that the method based on linear interpolation results in a significant improvement over the previous works. As emphasized toward the end of the literature review in Section 3, no previous work has ever tried to solve the kind of problem that we deal with in this paper. The linearity assumption, however, is not scalable to large problems because it needs boundary policies whose number increases exponentially with the number of situation indicators. Our future work, therefore, will be to drop the linearity assumption and use a multi-layer perceptron to represent a policy instead of the linear scoring functions so that we do not need to use boundary policies. Another viable approach would be to apply a reinforcement learning algorithm that learns value functions based on the rewards from the environment.

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

Funding: This work was supported by Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No. 2020-0-01450, Artificial Intelligence Convergence Research Center [Pusan National University]).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Dekker, R.; Voogd, P.; Van Asperen, E.T. Advanced methods for container stacking. *Contain. Termin. Cargo Syst.* **2006**, *28*, 563–586. [CrossRef]
2. Duinkerken, M.B.; Evers, J.J.M.; Ottjes, J.A. A simulation model for integrating quay transport and stacking policies on automated container terminals. In Proceedings of the 15th European Simulation Multiconference, Prague, Czech, 6–9 June 2001; pp. 909–916.
3. Kim, K.H.; Hong, G.P. A heuristic rule for relocating blocks. *Comput. Oper. Res.* **2006**, *33*, 940–954. j.cor.2004.08.005. [CrossRef]
4. Jang, H.; Choe, R.; Ryu, K.R. Deriving a Robust policy for container stacking using a noise-tolerant genetic algorithm. In Proceedings of the 2012 ACM Research in Applied Computation Symposium, San Antonio, TX, USA, 23–26 October 2012; ACM Press: New York, NY, USA, 2012; p. 31. [CrossRef]
5. Park, T.; Choe, R.; Kim, Y.H.; Ryu, K.R. Dynamic adjustment of container stacking policy in an automated container terminal. *Int. J. Prod. Econ.* **2011**, *133*, 385–392. [CrossRef]
6. Park, T.; Sohn, M.; Ryu, K.R. Optimizing stacking policies using an MOEA for an automated container terminal. In Proceedings of the 40th International Conference on Computers and Industrial Engineering, Awaji, Japan, 25–28 July 2010; pp. 1–6. [CrossRef]
7. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*, 2nd ed.; The MIT Press: Cambridge, MA, USA; London, UK, 2018.
8. Kim, J.; Hong, E.J.; Yang, Y.; Ryu, K.R. Noisy Optimization of Dispatching Policy for the Cranes at the Storage Yard in an Automated Container Terminal. *App. Sci.* **2021**, *11*, 6922. [CrossRef]
9. Taleb-Ibrahimi, M.; De Castilho, B.; Daganzo, C.F. Storage space versus handling work in container terminals. *Transport. Res. B Methodol.* **1993**, *27*, 13–32. [CrossRef]
10. Kim, K.H.; Kim, H.B. Segregating space allocation models for container inventories in port container terminals. *Int. J. Prod. Econ.* **1999**, *59*, 385–392. [CrossRef]
11. Kim, K.H.; Park, K.T. A note on a dynamic space-allocation method for outbound containers. *Eur. J. Oper. Res.* **2003**, *148*, 92–101. [CrossRef]
12. Lee, L.H.; Chew, E.P.; Tan, K.C.; Han, Y. An optimization model for storage yard management in transshipment hubs. *Contain. Termin. Cargo Syst.* **2006**, *28*, 539–561._6. [CrossRef]
13. Kim, K.H.; Lee, J.S. Satisfying constraints for locating export containers in port container terminals. In Proceedings of the 2006 International Conference on Computational Science and Its Applications—Volume Part III, Glasgow, UK, 8–11 May 2006; pp. 564–573. doi: 10.1007/11751595_60. [CrossRef]
14. Zhen, L. Storage allocation in transshipment hubs under uncertainties. *Int. J. Prod. Res.* **2014**, *52*, 72–88. 00207543.2013.828166. [CrossRef]
15. Said, G.A.E.N.A.; El-Horbaty, E.S.M. An intelligent optimization approach for storage space allocation at seaports: A case study. In Proceedings of the 2015 IEEE 7th International Conference on Intelligent Computing and Information Systems, Cairo, Egypt, 12–14 December 2015; pp. 66–72. [CrossRef]
16. Kim, K.H.; Park, Y.M.; Ryu, K.R. Deriving decision rules to locate export containers in container yards. *Eur. J. Oper. Res.* **2000**, *123*, 89–101. [CrossRef]
17. Borgman, B.; Asperen, E.van; Dekker, R. Online rules for container stacking. *Contain. Termin. Cargo Syst.* **2010**, *32*, 687–716. [CrossRef]
18. Chen, L.; Lu, Z. The storage location assignment problem for outbound containers in a maritime terminal. *Int. J. Prod. Econ.* **2012**, *135*, 73–80. [CrossRef]
19. Kang, J.; Ryu, K.R.; Kim, K.H. Determination of storage locations for incoming containers of uncertain weight. In Proceedings of the 19th international conference on Advances in Applied Artificial Intelligence: Industrial, Engineering and Other Applications of Applied Intelligent Systems, Annecy, France, 27–30 June 2006; pp. 1159–1168._123. [CrossRef]
20. Kang, J.; Ryu, K.R.; Kim, K.H. Deriving stacking strategies for export containers with uncertain weight information. *J. Intell. Manuf.* **2006**, *17*, 399–410. [CrossRef]
21. Zhang, C.; Wu, T.; Zhong, M.; Zheng, L.; Miao, L. Location assignment for outbound containers with adjusted weight proportion. *Comput. Oper. Res.* **2014**, *52*, 84–93. [CrossRef]
22. Ries, J.; Gonzalez-Ramírez, R.G.; Miranda, P. A fuzzy logic model for the container stacking problem at container terminals. *Lect. Notes Comput. Sci.* **2014**, *8760*, 93–111._7. [CrossRef]
23. Rekik, I.; Elkosantini, S. A multi agent system for the online container stacking in seaport terminals. *J. Comput. Sci.* **2019**, *35*, 12–24. [CrossRef]
24. Gunawardhana, J.A.; Perera, H.N.; Thibbotuwawa, A. Rule-based dynamic container stacking to optimize yard operations at port terminals. *Marit. Transp. Res.* **2021**, *2*, 100034. [CrossRef]

25. Park, T.; Choe, R.; Ok, S.M.; Ryu, K.R. Real-time scheduling for twin RMGs in an automated container yard. *Contain. Termin. Cargo Syst.* **2010**, *32*, 593–615. [CrossRef]
26. Kim, T.; Ryu, K.R. Experimental data and the executable to test the performance of situation-adaptive policy for container stacking in an automated container terminal. *Mendeley Data* **2022**, *V1*. [CrossRef]