Transportation Research Part C 52 (2015) 132–143

Contents lists available at ScienceDirect

Transportation Research Part C

journal homepage: www.elsevier.com/locate/trc

The container retrieval problem with respect to relocation

Dung-Ying Lin[a], Yen-Ju Lee[b], Yusin Lee[b,*]

[a] Department of Transportation and Communication Management Science, National Cheng Kung University, Taiwan
[b] Department of Civil Engineering, National Cheng Kung University, No. 1 University Road, Tainan City 70101, Taiwan

A R T I C L E I N F O

Article history:
Received 14 September 2014
Received in revised form 6 December 2014
Accepted 23 January 2015

Keywords:
Container retrieval
Rail mounted gantry crane (RMGC)
Container yard
Multi-lift
Heuristic

A B S T R A C T

The demand for container terminal yards is growing significantly faster than the supply of available land; therefore, containers are typically stacked high to better utilize the land space in container yards. However, in the process of container retrieval, non-productive reshuffling may be required to relocate the containers that are stacked on top of the target container. Container retrieval is directly related to the operational efficiency of terminals. Because the industry has become increasingly competitive, it has become critical to introduce a systematic approach to retrieving containers. In this study, we develop a heuristic that can generate feasible working plans for rail-mounted gantry cranes (RMGC) in container yards to minimize the number of container movements while taking the RMGC working time into consideration. The methodology takes into consideration the case that containers are grouped in terms of their retrieval order. Multi-lift RMGC models also are studied. Comprehensive numerical experiments reveal that the method runs faster than other methods published in the literature by several orders of magnitude; additionally, our method is able to solve instances larger than practical use. The number of movements approaches a theoretical lower bound, and the numerical results clearly demonstrate the tradeoff between the number of movements and the working time, and provide useful insights for yard planning.

1. Introduction

Recent decades have seen rapid growth in the marine container shipping industry. Containerized trade was estimated to have expanded at an average rate of 8.2% per year between 1990 and 2010 (Meng et al., 2014). To address the challenge of rapid growth, major carriers have built large vessels in the pursuit of lower costs and increased competitiveness. The maximum vessel size was 4300 TEUs in 1988, went up to 15,500 TEUs in 2006, and is 18,000 TEUs in 2014 (Tran and Haasis, 2015). As a result, terminal operators find elevated pressure in maintaining service quality due to increasing container throughput. The operational problems of container terminals merit additional study and have attracted significant attention in recent years. Stahlbock and Voss (2008) and Vis and de Koster (2003) provided a comprehensive overview of the operations and problems in container terminals and referred to solution methods for these critical issues.

For container terminals, one of the performance measures for customer service is the berthing time, which consists primarily of the time required for container loading and unloading for containerships. In most large yards, containers tend to be stacked high to better utilize the land space because the demand for land in yards grows significantly faster than the supply. To achieve high working efficiency, stacks are arranged side by side to form bays, and bays are organized into blocks.

* Corresponding author. Tel.: +886 6 2757575x63118; fax: +886 6 2358542.
E-mail address: yusin@mail.ncku.edu.tw (Y. Lee).

http://dx.doi.org/10.1016/j.trc.2015.01.024
0968-090X/© 2015 Elsevier Ltd. All rights reserved.

However, reshuffle operations may be unavoidable in the process of retrieving containers from high stacks. A reshuffle is an operation in which a container-lifting equipment is used to relocate a container from one stack to another; typically, a reshuffle is undertaken when one container blocks the access to the target container. These operations are unproductive and time/resource consuming, and should be avoided whenever possible to enhance the terminal performance.

A number of papers (for example, Kim et al., 2000; Dekker et al., 2006; Han et al., 2008; Wan et al., 2009; Zhang et al., 2010; Choe et al., 2011) have addressed the problem of assigning storage locations for containers to minimize reshuffling during the loading process. Unfortunately, reshuffling can hardly be completely avoided, for several reasons. First, the export containers typically begin to arrive at a terminal more than two weeks in advance of the shipping time, which is well before the loading sequence is available. Second, the information provided regarding an export container at the time of its arrival is often imprecise. Finally, even under ideal conditions, the limited storage capacity of a yard makes eliminating the need to reshuffle nearly impossible in practice.

In this study, we investigate the container retrieval problem considering the relocations that occur when all of the containers stacked in a multiple-bay block must be loaded onto a ship in a given order. The problem also entails the determination of how relocated containers should be stored within the block if reshuffles must occur. In practice, the containers in the yard are often divided into groups according to the ship planning requirements, which concerns the position of each container in the ship. During loading, the containers must be retrieved group-by-group in a pre-determined order, although the containers of the same group can be retrieved in any order. Given an initial layout that specifies the set of containers as well as the location of each container in the block, the optimization goal is to minimize the total number of container movements. Variations of the container retrieval problem are explored in this research. In the first variation, we assume that only a single-lift rail-mounted gantry crane (RMGC) can be used. The second variation extends the first problem by investigating a situation in which the crane has multi-lift capability; that is, the crane is equipped with multiple spreaders, thus has the ability to lift multiple containers simultaneously.

To the best of our knowledge, only a limited number of papers have addressed the container retrieval problem while recognizing relocation events. The work by Kim and Hong (2006) suggested two methods for determining the locations of relocated containers during pickup operations. Their work assumes that relocations occur only in the same bay, and only at moments in which a target container must be retrieved. However, as will become clear later, it is sometimes beneficial to pre-relocate a container, especially when the yard is utilized close to capacity. Lee and Hsu (2007) provided a multi-commodity flow-based mathematical model for the container pre-marshalling problem, and presented an extension of the model to solve for an optimal plan to transfer a given initial yard layout to another specified final layout. A slight modification of that extension will make their model applicable to the container retrieval problem. Nonetheless, due to the complexity of the underlying multi-commodity flow model, numerically solvable problem instances are far smaller in scale than those seen in reality. Lee and Lee (2010) presented a three-phase heuristic that yields a working plan for a crane to retrieve all of the containers from a given yard according to a prescribed order. However, the empirical study undertaken in this paper indicates that our solution heuristic outperforms theirs, both in solution efficiency and solution quality. Caserta et al. (2011) proposed an algorithm, inspired by the corridor method, to identify a working plan that minimizes the number of relocations when retrieving a subset of containers from the yard in a given order. When retrieving a target container, their algorithm is allowed to relocate only the containers found directly above the target container in the same stack using a last-in-first-out (LIFO) policy, which also excludes the possibility of pre-relocation. A related problem, studied by Meisel and Wichmann (2010), is to convert the configuration of a bay on a ship into another configuration within a minimum working time. Their work assumed that reshuffled containers are always relocated exactly once. Based on this and other assumptions, Meisel and Wichmann (2010) constructed a binary integer program that minimizes the working time, and they proposed a heuristic to solve it. Forster and Bortfeldt (2011) proposed a tree search heuristic to find a sequence of crane moves in order to retrieve all containers from a block. Numerical results suggest that this method yields better quality solution than those obtained by Lee and Lee (2010), and takes less than 10 s to solve. Forster and Bortfeldt (2012) studied the container relocation problem, recognizing the grouping practice in container shipping. Pre-relocation is also considered, in that the algorithm is able to relocate a container at an appropriate time, before another container stacked beneath it is due to be retrieved. This work allows relocation within the same bay only, and assumes that the time required to move a container between any two stacks are the same. Yet, the multi-lift capability of RMGC was not explored in their work.

The current paper is differentiated from earlier works by taking into consideration some features that are critical when developing container retrieval plans in reality. Specifically, our heuristic explicitly utilizes opportunities to pre-relocate a container, allows inter-bay relocation, and takes the crane movement time into consideration when relocating containers. Furthermore, an extended version of the heuristic covers multiple-lift cranes. It is also worthwhile to note that all of the papers reviewed above that concerns the container retrieval problem attempts to solve the problem by constructing mathematical models, with the only exception of Forster and Bortfeldt (2012), which takes a procedural approach. Mathematical models are powerful tools, but they are also known for their relatively low flexibility. They also tend to be harder to solve, limiting the size of solvable instances. Due to this reason, none of the works presented computational results against real-sized instances, which can involve thousands of containers. Our work develops a procedural approach that is computationally efficient, and is able to solve container retrieval problems exceeding realistic sizes. Furthermore, minimizing the number of container movements has been the optimization goal for related models, but reducing the overall working time is also important in reality. By taking the crane movement time into consideration, we demonstrate, numerically, that the movement count and the working time do not necessarily mirror each other. Instead, the tradeoff between movement count and working time is evident and will be discussed in later sections.

The structure of this paper is as follows: Section 2 describes the container retrieval problem; Section 3 explains the solution approaches used to solve the proposed problem; and the analyses of the empirical results of the approaches are presented in Section 4. The final section concludes the study, making recommendations for future research.

2. The container retrieval problem

Given an initial layout for a container yard, the container retrieval problem solves for a movement sequence that retrieves all of the containers from a yard in a certain order so that the number of container movements is minimized. Without loss of generality, we assume that all of the containers are of identical size and belong to G different groups. The container groups are numbered with consecutive integers, starting from 1, which we refer to as the group indexes. Each container group possesses a unique index, and the groups with smaller indexes must be retrieved earlier. In an ideal case where all of the containers are stacked in a way that is fully compatible with the retrieval order, no extra movement is needed in the retrieval process, and the minimum number of movements is equal to the total number of containers. However, in a realistic situation, some containers may block the retrieval of others, and it is apparent that the blocking containers must be relocated. From an optimization standpoint, the challenging problems are: (1) when to remove the blocking containers and (2) where to relocate them. In this research, we focus on yards that are equipped with an RMGC as their major piece of container handling equipment. We assume the RMGC is capable of moving containers between different bays, and there is only one RMGC in the working area.

The storage yard of a container terminal is divided into large storage spaces, called blocks. Fig. 1 conceptually illustrates a typical storage block in a container terminal, displayed together with a twin-lift RMGC.

The smallest storage unit is a slot for a single container; in such a storage position, a container lies naturally either on the ground or exactly on top of another container of the same size. The container slots are located on top of each other to form a stack. The levels of the slots are referred to as tiers. Multiple stacks form a bay in which containers rest against each other by their long sides. Stacks placed alongside each other in a block form a row. In this paper, the triple (bay, row, tier) defines the (x,y,z) coordinates of a slot in a block. The lifting equipment largely determines the dimension of the blocks. The type of lifting equipment determines the bay layout. For yards equipped with RMGCs, the gauge of the crane determines the width (the number of rows) of a bay, which can reach ten rows or more. The stack height is limited by the lifting height, and the number of bays in the block is determined by the length of the rail. When retrieving a container, the crane picks up the container and places it on a waiting truck. We assume that trucks wait next to row 1 at the same bay of the target container.

RMGCs of newer models, especially the fully-automated ones, are able to move containers between bays without the help of a truck. Before the crane moves the gantry or the trolley, it normally lifts its spreaders to the highest position, which is higher than any stack. This practice allows the crane to move freely in the block, even when carrying containers along.

3. The solution heuristic

To solve the container retrieval problem described in the earlier section, we propose a series of heuristics that are suitable for variations of the container retrieval problem. The solution heuristics are detailed in this section.

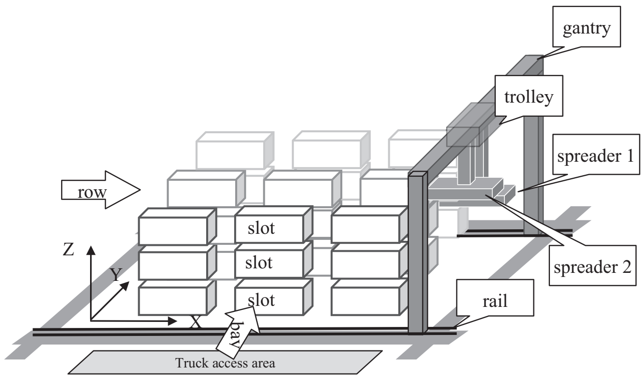

Fig. 1. A typical storage block in a container terminal.

3.1. Background

Retrieving containers from a yard involves a number of container movements; each movement consists of a lift-up operation, followed by a place-down operation. For ease of explanation, we assign each container a unique identification number. These numbers are consecutive integers starting from 1, and we adapt the convention that containers with larger group indexes (which should be retrieved later) always have larger identification numbers. For example, consider six containers belonging to three groups, with each group having two containers. Then, the containers with identification numbers 1 and 2 belong to group 1, containers 3 and 4 belong to group 2, and containers 5 and 6 belong to group 3. Under this convention, in the special case that each group has only one container, a container's identification number is identical to its group index. To simplify presentation, we maintain this assumption for most of the paper, allowing us to represent each operation with a triplet (i,s,P) in which the three elements correspond to the container identification number/group index (i), the stack involved (s), and the operation type (P). The stacks are numbered consecutively according to the triplet (bay, row, and tier), and the stack number (s) is an integer from 1 to the total number of stacks. In this work, we reserve the designation of stack 0 for the truck that carries the container away from the yard. Operation type (P) is either U for lift-up or D for place-down. For example, the pair of operations $(5,3,U)$ and $(5,0,D)$ represents the movement of lifting container 5 out of stack 3 and onto a truck. Hence, a working plan to retrieve all of the containers from a yard is an ordered set of operations. Initially, we assume that there is only one spreader on the trolley. In this case, each type U operation must be followed by a type D operation to complete a container movement. The situation is more complicated when the RMGC is equipped with more than one spreader. The optimization of multiple spreader scenarios is discussed later.

Consider the simple example depicted in Fig. 2 in which there are three stacks in the block. Stack 1 holds four containers, while the other two stacks are empty. A feasible working plan to retrieve all of the containers is $(1,1,U)$, $(1,0,D)$, $(4,1,U)$, $(4,3,D)$, $(3,1,U)$, $(3,2,D)$, $(2,1,U)$, $(2,0,D)$, $(3,2,U)$, $(3,0,D)$, $(4,3,U)$, $(4,0,D)$. This plan retrieves the containers in the required order, where those with smaller identification numbers are always retrieved earlier, as mentioned above. In the process, containers 3 and 4 require extra movements because they block access to container 2, which must be retrieved earlier. For this single stack example, one can estimate that the least number of extra (or non-productive) movements to retrieve all of the containers is two, because there are two containers blocking at least one other container (containers 3 and 4 are blocking container 2). Thus, the entire stack requires a minimum of six movements to complete the retrieval. Because this working plan has six movements, it is optimal in terms of the number of movements. By performing the same calculation on every stack and summing up the required least number of movements for each stack, one can obtain a lower bound for the number of movements required to empty the block. This bound is not necessarily tight because it is possible that some containers might require more than one move in the retrieval process, especially when the slots are nearly full. The same lower bound also is used in Lee and Lee (2010) and will be used to evaluate the numerical results in this paper.

The discussion above naturally divides the containers in the initial layout into two types: those that do not block other containers (type A) and those that do (type B). Suppose a feasible working plan always relocate a type B container to a stack which is either empty or stores no container that has a smaller identification number. Then each of the type A containers can be retrieved in just one movement, and each of the type B containers will need just one relocation before its retrieval. Thus the working plan achieves optimality in terms of number of movements. Based on this observation, our heuristic attempts to minimize the number of movements by carefully selecting appropriate stacks for type B containers to relocate to, and by searching for opportunities to pre-relocate type B containers. When there are multiple choices of stacks to relocate a container to, a set of rules guide the heuristic to select one such that the current relocation is less likely to result in a type B container be relocated more than once in the future. Movement distances are also considered to decrease the crane's working time.

3.2. The heuristic for generating a feasible working plan for single-lift RMGC

Before constructing the working plans, we first introduce the criteria that can be used to assess the feasibility of a working plan. A working plan can be determined to be infeasible, should any one of the following situations occur:

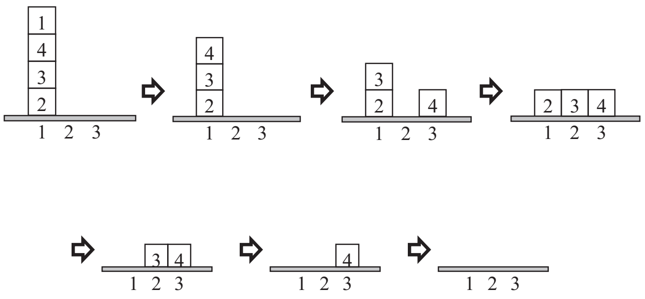

Fig. 2. An illustration of a container retrieval process.

1. The target container to be picked up is not located on the top of the stack.
2. The height of the target stack exceeds its limit after placing a container onto that stack.
3. The crane attempts to pick up a container when all of the spreaders are full.
4. A container is retrieved when other containers with smaller group indexes are still in the block.
5. The crane attempts to put down a container that it is not being held.
6. Not all of the containers in the block are retrieved when the working plan is complete.

A working plan that does not violate any of the above criteria is considered feasible. Given the initial layout of a block and a corresponding working plan, one can evaluate the feasibility of the working plan via simulation or alternative methods by examining the aforementioned criteria. Next, we introduce a heuristic to generate a feasible working plan. For the sake of clarity, we first introduce the following notation, which will be used throughout this paper.

S	The set of stacks
H	The maximum height of a stack
n	The total number of containers in the yard
G	The total number of groups in the yard
P_b	The penalty factor for bay selection
P_r	The penalty factor for row selection
$h(s)$	The number of containers residing in stack s, $s \in S$
$s(c)$	The stack in which container c is located
$bay(s)$	The bay number of stack s
$row(s)$	The row number of stack s
$top(s)$	The container located at the top of stack s. Further, we set $top(s) = n + 1$, if stack s is empty
$\min_g$	The smallest group index in stack s
$g(c)$	The group index of container c

The heuristic works by retrieving the containers one-by-one and terminates when all n containers have been retrieved. To simplify the model, we assume that a virtual container is placed at the bottom of every stack. The ID number of the virtual container is $n + 1$, and the group index of the container is $G + 1$. These virtual containers are never moved or retrieved. This setting allows us to design the heuristic without considering the fact that some stacks might become empty during the retrieval process. In other words, from the model's point of view, no empty stack exists under this assumption. The property enables the heuristic to fully utilize the empty stacks that might come into and out of existence throughout the retrieval process.

The heuristic is illustrated in Fig. 3. Let g be the current group destined for retrieval. If all of the containers in group g are retrieved, we go to the next group. Otherwise, among all of the containers in group g, we select a target container, c, according to the following rule. If one of the spreaders is holding a group g container, the one in the spreader should be retrieved since it is already lifted up. Otherwise, we select the one with the least number of overlaying containers. In other words, the container in group g that is on the top of a stack should be retrieved first. The rest of the containers in group g are retrieved in the order of increasing overlaying container number. For instance, if a container in group g has two overlaying containers, it will be retrieved before another container in the same group that was placed beneath three containers. Ties are broken arbitrarily, since there is no definite rule to determine the better choice among, say, two containers of group g that are both on top of their stack.

If the selected container c is located at the top of its stack, it can be retrieved directly. Otherwise, let $d = top(s(c))$ be the identification number of the container on the top of the stack in which c resides, and $g(d)$ is its group index. We devise the following steps, based on $g(d)$. Suppose that there exists at least one stack s, where $h(s) < H$ and $\min_g(s) > g(d)$; then, container d can be moved to this stack without generating additional relocations, regardless of the arrangement of the existing containers in that stack. We select an ideal stack t for this container, based on the rule presented below (Rule *I*), if more than one such stack exists. Alternatively, if there is no such stack, container d must be re-relocated at least one more time, no matter where it is relocated to at this moment. In this case, we select another stack u based on Rule *III* (presented later), and then we move the container into it.

Let us first consider the case in which we can find at least one stack s, satisfying $h(s) < H$ and $\min_g(s) > g(d)$. As might be expected, sometimes multiple stacks can satisfy these two criteria. If this is the case, the best strategy is to use the stack with the smallest $\min_g(s)$ so that other stacks with larger $\min_g(s)$ can be saved for later relocation use, since the containers in a stack with a larger $\min_g(s)$ value are less likely to be blocked by other containers. The location of the stack also must be considered. Because we assume that the truck waits next to row 1, selecting a stack with a smaller row number saves the trolley travel time when moving the container to the truck later. Additionally, the gantry travel time can be reduced by selecting a stack in a nearby bay as the target. Below, Rule *I* was designed based on this logic.

Rule *I*. From among all of the stacks $s \in S \setminus \{s(c)\}$ satisfying $h(s) < H$ and $\min_g(s) > g(d)$, select a stack t with the smallest stack selection index, $SSI(s)$, as the target stack.

The stack selection index is calculated with Eq. (1), where the first term on the right hand side is the minimum group index among all of the containers in stack s. Additionally, the following term is the row number of stack s multiplied by

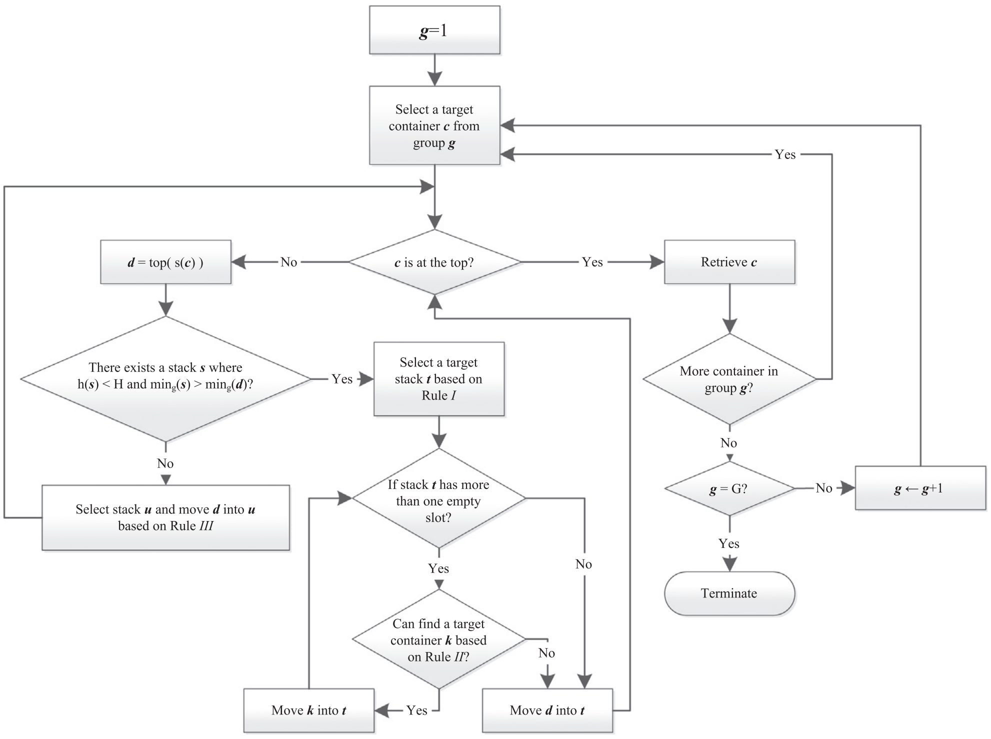
g=1
Select a target container c from group g
c is at the top?
No
Yes
d = top(s(c))
Retrieve c
There exists a stack s where h(s) < H and min_g(s) > min_g(d)?
Select a target stack t based on Rule I
More container in group g?
Select stack u and move d into u based on Rule III
If stack t has more than one empty slot?
g = G?
g ← g+1
Can find a target container k based on Rule II?
Terminate
Move k into t
Move d into t

Fig. 3. The heuristic for generating a retrieving plan.

the row penalty factor P_r, and the final term is the bay penalty factor P_b multiplied by the number of bays, should the gantry travel for the movement under consideration.

$$SSI(s) = \min_g(s) + P_r row(s) + P_b|bay(s) - bay(s(c))| \tag{1}$$

Rule *I* selects stack t as the target stack to which container d should be relocated. The bay penalty factor P_b reflects the gantry movement time. A higher penalty value indicates higher conceived importance of the gantry movement time relative to the number of movements, thus giving priority to stacks located in nearby bays, at the cost of selecting a stack that is more likely to result in additional movements in the future should container d is moved into. Similarly, the row penalty factor P_r corresponds to the trolley movement time. Because we assume that trucks that carry the containers away always wait next to row 1, it is desirable to move the relocated container to a row with a smaller row number. Again, a higher P_r value indicates a higher importance of the trolley movement time.

If stack t has available space for more than one container, the heuristic goes one step further and determines whether it should perform a pre-relocation, which moves another container k into this stack, before moving container d into it. The determining rule is Rule *II*, described below.

Rule II. Find a container k that satisfies the following four conditions. If there is more than one such container, select the container with the largest identification number.

1. Container k is of type B, which blocks the way of another container.
2. $g(k) > g(d)$.
3. Container k is located at the top of $s(k)$.
4. $\min_g(t) - 5 < g(k) < \min_g(t)$.

If there is a container k that satisfies all of the above conditions, it is desirable to move container k into stack t before moving container d. Because container k blocks the access to another container, at least one relocation of container k in the future before being retrieved is unavoidable. Therefore, moving container k to stack s before moving container d does not increase the total number of container movements. Moreover, once such a container is moved to stack s, it no longer blocks the other containers. Thus, this preventive movement obviates the possibility that container k will be unable to find an appropriate stack for relocation in the future. If more than one container satisfies all of the conditions above, Rule *II* selects the container with the largest group index because containers with larger group indexes have fewer choices for relocation. Finally, for the same reason explained in Rule *I*, stacks with larger $\min_g(t)$ values are able to accommodate a wider range of containers without the generation of additional relocation movement. Sometimes, such stacks are scarce; thus, it is undesirable to move a container with a small group index into a stack with a large $\min_g(t)$ value. Condition 4 in Rule *II* is designed to prevent such a situation, where the parameter 5 is an empirical value. We suspect that it is vaguely relevant to the height of the stacks. However, extensively numerical experiments or mathematical proof may be required to obtain the concrete evidence or determine the logic behind it.

Recall that we are now attempting to retrieve container c, which is blocked by container d. When searching for a stack to which container d can be relocated, if $\min_g(s) < g(d)$ holds for all non-full stacks s, then container d will block other containers no matter where it is relocated. If this situation never happens, the number of movements in the resulting working plan will be equal to the lower bound. Otherwise, each occurrence of this situation increases the resulting number of movements by one. In this case, we select a target stack u for container d based on Rule *III*, as described below.

Rule III. Among all stacks other than $s(c)$ that are not full, select stack u which $\min_g(u)$ is largest, for the relocation of container d.

The time at which container d must be relocated again after being moved into stack u is determined by $\min_g(u)$. Rule *III* is designed to postpone this re-relocation by selecting a stack with a larger $\min_g(u)$ value. The later that re-relocation occurs, the fewer remaining containers there are in the block. Thus, the easier it will be to find an appropriate stack for relocation. Additionally, recall that the heuristic regards empty stacks as if they possess a virtual container and that its group index is $G + 1$. This design encourages Rule *III* to utilize the empty stacks for relocation.

We demonstrate the heuristic above with a simplified single-bay block example, the initial layout of which is displayed in Fig. 4. In this example, there are 6 rows in the bay, which holds a total of 12 containers with the maximum height H of 6. The numbers shown in the blocks are container ID numbers, which also are group indexes. We also set $P_r = P_b = 0$ for simplicity. The heuristic works as follows.

Iteration 1 is straightforward because container 1 is at the top of its stack and can be retrieved immediately. Fig. 5 shows the layout after this movement. Iteration 2 attempts to retrieve container 2, which is blocked by container 6. Because $\min_g(1) = 9$, $\min_g(4) = 10$ and $\min_g(6) = 8$ are stacks with minimum container group indexes greater than 6, we select stack $t = 6$ as the target stack for the relocation of container 6 according to Rule *I*. Further, because stack 6 has available space for more than one container, the heuristic moves container $k = 7$ into stack 6 before moving container 6, according to Rule *II*. Then, container 2 is retrieved, resulting in the layout shown in Fig. 6.

The target container for iteration 3, container 3, is blocked by container 11 under this layout. Every stack in the block has at least one container whose identification number is less than 11; therefore, the heuristic relocates container 11, according to Rule *III*, to stack 4, because $\min_g(4)$ is the largest among all of the possible stacks. The layout after this iteration is displayed in Fig. 7. The remaining iterations are straightforward and are omitted. The resulting working plan is depicted in Fig. 8 and is a feasible plan that can be used in practice.

3.3. Multiple spreaders

Due to their higher productivity, container terminals are increasingly adopting multiple-lift capability models in their operations. Although multiple-lift cranes are still limited to the quay, future multiple-lift yard cranes can theoretically perform tasks that single-lift models cannot. For example, when a twin-lift crane picks up a container and moves it to a truck, the crane also can stop during the process to pick up another container, and relocate the second container to another stack

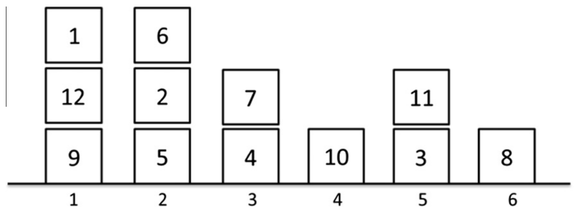

Fig. 4. The initial layout of the yard.

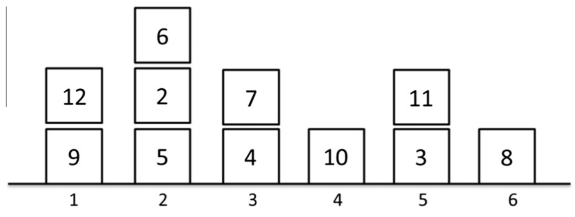

Fig. 5. The layout after container 1 has been retrieved.

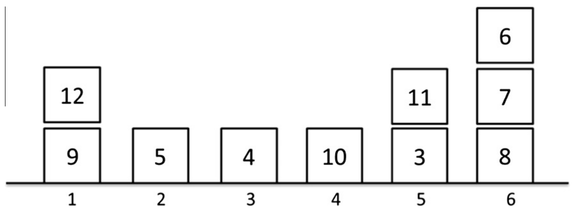

Fig. 6. The layout after container 2 has been retrieved.

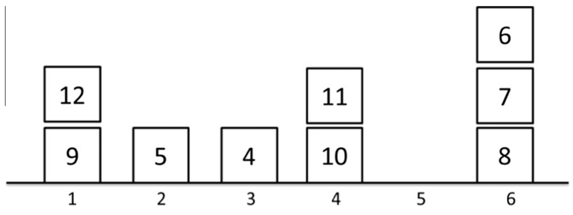

Fig. 7. The layout after container 3 has been retrieved.

(1,1,U), (1,0,D), (7,3,U), (7,6,D), (6,2,U), (6,6,D), (2,2,U), (2,0,D), (11,5,U), (11,4,D), (3,5,U), (3,0,D), (4,3,U), (4,0,D), (5,2,U), (5,0,D), (6,6,U), (6,0,D), (7,6,U), (7,0,D), (8,6,U), (8,0,D), (12,1,U), (12,2,D), (9,1,U), (9,0,D), (11,4,U), (11,2,D), (10,4,U), (10,0,D), (11,2,U), (11,0,D), (12,2,U), (12,0,D)

Fig. 8. A feasible working plan for the illustrative example.

before or after it delivers the first container to the waiting truck. RMGCs equipped with three or four spreaders can perform even more complicated tasks. This additional capability can potentially improve the overall efficiency of a RMGC.

The introduction of the second spreader brings fundamental changes into the problem. With only one spreader, a container that is picked up has to be placed down before lifting the next container. In other words, all the pick-up operations should be done in exactly the same order as all the put-down operations. Therefore the solution algorithm needs to consider only one set of sequence when searching for a good working plan. With multiple spreaders, the pick-up order can be different from the put-down order, and the relation between the two sets of orders are further constrained by the number of spreaders, in a way that algorithms developed for single-spreader models do not need to consider. To illustrate, consider three containers 1, 2, and 3 placed in stacked 1, with container 1 on the top and 3 at the bottom. With a RMGC equipped with only one spreader, a feasible working sequence that moves all the containers to stack 2 is $(1,1,U)$ $(1,2,D)$ $(2,1,U)$ $(2,2,D)$ $(3,1,U)$ $(3,2,D)$. The order the containers are picked up is 1,2,3, which is identical to the order they are placed down. If the crane has two spreaders, then this working sequence becomes feasible: $(1,1,U)$ $(2,1,U)$ $(2,2,D)$ $(1,2,D)$ $(3,1,U)$ $(3,2,D)$; but this working sequence is not: $(1,1,U)$ $(2,1,U)$ $(3,1,U)$ $(3,2,D)$ $(2,2,D)$ $(1,2,D)$. For the last sequence to be feasible, the crane needs three or more spreaders.

Because a multi-lift RMGC can carry more than one container at a time, the movement triplets (x,a,U) and (x,b,D) of the same container do not have to be executed consecutively. In other words, because there are multiple spreaders on the trolley, a container does not have to be put down immediately after being picked up. A container can be carried by a crane while the crane is dealing with other containers and can be put down at a suitable time to reduce the RMGC working time. Rule *IV* uses this property to reduce the number of operations in a given working plan.

Rule IV. If a container is moved to an intermediate stack and then lifted out later, and if there is an available empty spreader, the container can be kept on the spreader. Doing so reduces the number of operations by 2.

We use the working plan of Fig. 8 as an example, and we assume that there are two spreaders available. Then, instead of relocating container 6 from stack 2 to stack 6 at operation 6, the crane can keep the container in one of its spreaders, can execute the following operations with the other spreader until container 5 is retrieved, and then can retrieve container 6 by releasing it from the spreader. After releasing container 6, the rule can be applied to container 11 again. The working plan is reduced by four operations. Fig. 9 shows the result, with the cancelled operations stroked out.

(1,1,U), (1,0,D), (7,3,U), (7,6,D), (6,2,U), ~~(6,6,D)~~, (2,2,U), (2,0,D), (11,5,U), (11,4,D), (3,5,U), (3,0,D), (4,3,U), (4,0,D), (5,2,U), (5,0,D), ~~(6,6,U)~~, (6,0,D), (7,6,U), (7,0,D), (8,6,U), (8,0,D), (12,1,U), (12,2,D), (9,1,U), (9,0,D), (11,4,U), ~~(11,2,D)~~, (10,4,U), (10,0,D), ~~(11,2,U)~~, (11,0,D), (12,2,U), (12,0,D)

Fig. 9. An example working plan with two spreaders.

4. Numerical experiments

In this section, we provide computational results using the problem instances from Lee and Lee (2010) to demonstrate the performance of the heuristic proposed in this research. All of the methods described are implemented in C++ programming language. The examples are solved on a personal computer with an Intel$^{(R)}$ Core™ i7-2600 CPU running at 3.4 GHz with 16 GB memory. We assume that the speed of the gantry is 3.5 s per bay and that the speed of the trolley is 1.2 s per container width. The speed of the spreader is 2.59 s per tier when it is empty and 5.18 s per tier when loaded. The height of the truck is assumed to be one half that of a tier. The combined acceleration and deceleration time loss for the gantry is 40 s. These parameters can be calibrated based on real-world scenarios without affecting the applicability of the proposed heuristic.

The names of all of the instances start with either ‘*R*’ or ‘*U*’ which correspond to random instances and upside-down instances, respectively. In a random instance, all of the containers are located randomly. In the upside-down instances, the stacks are arranged in a way that the containers retrieved earlier are always placed beneath those retrieved later, representing the most unfavorable configuration. Following the first letter are three numbers separated by two underscores. The first number shows the block dimensions. For example, 101606 means the block has 10 bays, 16 rows, and a maximum stacking height of 6. The second number is the total number of containers, and the third number is a serial number. As the proposed solution approach involves randomness, instead of presenting the results from individual instances, each problem instance was tested with five runs and the resulting performance indices were averaged across these runs.

To quantify the complexity of the problem instances, we introduce an index of the usage rate for a yard. The usage rate is calculated by dividing the number of containers by the number of slots in that block. For example, if a block has 10 bays, each bay contains 16 rows, and the maximum height of a stack is 6, then the total number of slots in this block is 960. Suppose that there are 720 containers; then, the usage rate is 75%. A higher usage rate implies less available space for relocation as well as increased chances of a container blocking others; thus, the resulting problem tends to be more difficult to solve.

Table 1 displays a comparison of the current heuristic with Lee and Lee (2010) for random instances. Each row represents one problem instance, and the data is the average of five runs as explained above. The current heuristic reached lower number of container movements in all instances. The CPU time was improved by approximately five orders of magnitude.

Table 2 displays a comparison of the current heuristic with Lee and Lee (2010) for upside-down instances. The number of container movements achieved by the current heuristic also outperformed the pervious results in all instances, and the CPU time was improved by approximately five orders of magnitude. Due to a slight difference in the way they are calculated in the two studies, the crane working times are not directly comparable. Of the 20 instances tested, in nine instances, the

Table 1
A comparison with Lee and Lee (2010) for random instances.

ID	B[a]	H[b]	C[c]	LB[d]	L&L[e]		Heuristic	
					M[f]	CPU[g]	M[f]	CPU[g]
R011606_0070	1	6	70	104.4	125.4	8204.324	110.2	0.090
R021606_0140	2	6	140	209.0	230.2	21579.552	213.4	0.084
R041606_0280	4	6	280	426.8	454.2	21522.112	433	0.085
R061606_0430	6	6	430	655.6	709.8	21358.796	657.4	0.087
R081606_0570	8	6	570	875.6	945.4	21200.966	876.2	0.084
R101606_0720	10	6	720	1092.0	1169.2	20844.760	1093	0.085
R011608_0090	1	8	90	142.0	191.4	13353.362	152	0.078
R021608_0190	2	8	190	305.8	367.8	21527.714	315.2	0.086
R041608_0380	4	8	380	610.6	768.6	21231.078	623.8	0.088
R061608_0570	6	8	570	906.0	1242.0	20874.034	926.2	0.087

[a] B: number of bays.
[b] H: maximum height of stacks.
[c] C: number of containers.
[d] LB: lower bound on the number of movements.
[e] L&L: results from Lee and Lee (2010).
[f] M: number of movements.
[g] CPU: CPU time, in seconds.

Table 2
A comparison with Lee and Lee (2010) for the upside-down instances.

ID	B[a]	H[b]	C[c]	LB[d]	L&L[e]		Heuristic	
					M[f]	CPU[g]	M[f]	CPU[g]
U011606_0070_001	1	6	70	125	125	17326.31	125	0.124
U011606_0070_002	1	6	70	124	130	11243.4	129	0.124
U021606_0140_001	2	6	140	248	255	21549.7	249	0.109
U021606_0140_002	2	6	140	248	259	21537.87	252	0.109
U041606_0280_001	4	6	280	497	505	21272.84	497	0.171
U041606_0280_002	4	6	280	496	510	21371.13	496	0.114
U061606_0430_001	6	6	430	764	789	20872.78	764	0.14
U061606_0430_002	6	6	430	764	797	20986.66	765	0.124
U081606_0570_001	8	6	570	1013	1059	20452.36	1013	0.124
U081606_0570_002	8	6	570	1014	1029	20555.97	1014	0.14
U101606_0720_001	10	6	720	1281	1368	19521.25	1281	0.109
U101606_0720_002	10	6	720	1281	1322	19923.64	1281	0.098
U011608_0090_001	1	8	90	164	175	21586.81	172	0.093
U011608_0090_002	1	8	90	164	180	8021.31	175	0.124
U021608_0190_001	2	8	190	348	387	21368.39	348	0.109
U021608_0190_002	2	8	190	348	414	21373.67	361	0.156
U041608_0380_001	4	8	380	696	848	21157.09	703	0.109
U041608_0380_002	4	8	380	696	752	20623.64	698	0.093
U061608_0570_001	6	8	570	1044	1230	20369.41	1052	0.093
U061608_0570_002	6	8	570	1044	1238	19763.89	1045	0.124

[a] B: number of bays.
[b] H: maximum height of stacks.
[c] C: number of containers.
[d] LB: lower bound on the number of movements.
[e] L&L: results from Lee and Lee (2010).
[f] M: number of movements.
[g] CPU: CPU time, in seconds.

number of movements reached the lower bound, and in another three instances, the solutions only possess one move above the lower bound.

Table 3 tests the influence of the row penalty factor and the bay penalty factor. Five different combinations of both penalty factors were tested on 40 different instances. The test instances are divided into eight categories, covering the combinations of random instances and upside-down instances, 90% and 70% usage rates, and different group sizes. In instances

Table 3
The results under different penalty factors.

Instance	LB[a]	(P_r, P_b)									
		(0,0)		(15,150)		(20,200)		(25,250)		(30,300)	
		M[b]	WT[c]	M[b]	WT[c]	M[b]	WT[c]	M[b]	WT[c]	M[b]	WT[c]
Random instances, 90% usage, group size = 1											
R601609_7800	13017.6	13457.2	3596045.2	13765.0	3087236.0	13773.4	3077087.8	13817.8	3083005.6	13853.2	3093073.6
Random instances, 70% usage, group size = 1											
R601609_6100	9853.0	9869.0	2634949.8	9920.6	2201552.4	9930.0	2196007.4	9935.4	2190591.4	9946.2	2189046.2
Random instances, 90% usage, random group size											
R601609_7800	12966.0	13821.6	3296280.2	13920.6	2958727.4	13921.8	2952830.6	13925.2	2958854.2	13925.8	2959819.0
Random instances, 70% usage, random group size											
R601609_6100	9835.8	9973.8	2412059.6	10088.0	2158531.0	10091.0	2164255.6	10093.8	2164758.2	10100.2	2165135.6
Upside-down instances, 90% usage, group size = 1											
U601609_7800	14640.0	14643.2	3822518.4	14643.4	3060584.4	14645.2	3038712.6	14645.6	3023891.4	14646.2	3010270.4
Upside-down instances, 70% usage, group size = 1											
U601609_6100	11241.4	11242.4	2960363.6	11242.4	2370676.8	11242.4	2353212.6	11242.4	2343136.8	11242.4	2335201.8
Upside-down instances, 90% usage, random group size											
U601609_7800	14629.6	15114.4	3665089.8	15211.4	3163958.8	15220.6	3178685.6	15216.0	3183673.2	15235.8	3194531.4
Upside-down instances, 70% usage, random group size											
U601609_6100	11231.0	11240.2	2794343.8	11302.6	2360642.6	11308.4	2362728.8	11306.6	2362533.8	11306.6	2367634.6

[a] LB: lower bound on the number of movements.
[b] M: number of movements.
[c] WT: working time.

with a group size of one, every container belongs to a different group. In other instances, the number of containers belonging to each group is randomly chosen, where the average is fixed at ten containers per group. The CPU time is not presented because all instances required less than one second to compute.

The test results provide important insights. First, the $(P_r, P_b) = (0, 0)$ setting resulted in the least number of movements, which is not surprising. However, this setting also resulted in the highest working time. This seemingly counter-intuitive result is in fact reasonable, because the movement count does not fully reflect the time taken for the crane to reposition itself. The test results show that higher penalty factors result in a larger number of movements and shorter total working time, but only to some extent. Beyond that, higher penalty factors increase both the number of movements and the working time.

Table 4 compares the average working time for each movement (in seconds) for the same test displayed in Table 3. It is interesting to note that, for instances of the same dimension, usage rate, and grouping style, the average working time required by each movement is lower for the upside-down instances, except when the penalty factors are set to $(P_r, P_b) = (0, 0)$. This is probably caused by the fact that, when the penalty factors are greater than zero, the heuristic tends to relocate the containers to stacks that are closer to the truck. In the upside-down instances, more containers are relocated with simultaneous improvements in their locations, which results in the observed phenomenon.

Table 5 tests the effect of multiple spreaders. The row and bay penalty factors are set to $(P_r, P_b) = (25, 250)$ in these experiments. The number of movements performed by one spreader exceeds the lower bound by less than 10% in all of the instances tested, and most are less than 1% when the usage rate is 70%. The number of movements for a few instances is lower than the lower bound when multiple spreaders are available, which is normal because the lower bound is estimated based on one spreader. The number of movements decreases with increasing spreader availability, as expected. However, it can be seen that the difference is less than 1%. Because the lower bound for multiple spreaders is unclear, whether this can be further improved remains to be studied in the future.

Table 4
The average working time per movement under the different penalty factors.

Instance	(P_r, P_b)				
	(0,0)	(15,150)	(20,200)	(25,250)	(30,300)
Random instances, 90% usage, group size = 1					
R601609_7800	267.2	224.4	223.2	223.2	223.2
Random instances, 70% usage, group size = 1					
R601609_6100	266.8	222.2	221.4	220.4	220.0
Random instances, 90% usage, random group size					
R601609_7800	238.6	212.2	212.4	212.6	212.6
Random instances, 70% usage, random group size					
R601609_6100	241.8	214.0	214.4	214.6	214.4
Upside-down instances, 90% usage, group size = 1					
U601609_7800	261.4	209.2	207.4	206.4	205.4
Upside-down instances, 70% usage, group size = 1					
U601609_6100	263.2	211.0	209.0	208.4	207.6
Upside-down instances, 90% usage, random group size					
U601609_7800	242.4	208.0	208.8	209.2	209.6
Upside-down instances, 70% usage, random group size					
U601609_6100	248.6	208.8	209.0	209.0	209.6

Table 5
The results with different numbers of spreaders.

Instance	LB[a]	The number of spreaders									
		1		2		3		4		5	
		M[b]	WT[c]	M[b]	WT[c]	M[b]	WT[c]	M[b]	WT[c]	M[b]	WT[c]
R601609_7800	13017.6	13817.8	3083005.6	13759.2	3069790.2	13742.0	3063924.4	13733.8	3064087.8	13727.8	3060239.0
R601609_6100	9853.0	9935.4	2190591.4	9891.4	2180498.0	9876.4	2176151.8	9870.2	2174840.8	9867.4	2173946.8
R601609_7800	12966.0	13925.2	2958854.2	13867.0	2941948.2	13862.0	2941725.8	13835.4	2932394.4	13814.2	2922264.2
R601609_6100	9835.8	10093.8	2164758.2	10055.6	2155756.4	10041.6	2149219.8	10030.2	2151030.6	10021.4	2146404.6
U601609_7800	14640.0	14645.6	3023891.4	14615.4	3017810.0	14606.4	3012446.2	14603.4	3015725.8	14602.8	3016043.2
U601609_6100	11241.4	11242.4	2343136.8	11205.0	2335633.0	11191.8	2334544.6	11187.6	2332063.0	11186.8	2332407.6
U601609_7800	14629.6	15216.0	3183673.2	15170.8	3170054.8	15159.4	3165046.8	15130.8	3143350.0	15130.4	3148371.2
U601609_6100	11231.0	11306.6	2362533.8	11276.4	2357447.0	11257.6	2353370.8	11245.8	2353174.4	11240.4	2344733.6

[a] LB: lower bound on the number of movements.
[b] M: number of movements.
[c] WT: working time.

5. Conclusions and directions for future research

In this research, we have developed heuristics to solve the container retrieval problem, yielding a working plan for RMGCs to retrieve all of the containers from a given export container yard in a given sequence. The containers are divided into groups, and the containers with smaller group indexes should be retrieved earlier. The optimization goal is to minimize both the number of container movements and the RMGC working time.

This research also investigated variations in the container retrieval problem. We first proposed a heuristic to generate a retrieving plan for the single-lift crane. The retrieving plan can be further modified to utilize additional spreaders and to reduce the number of movements. The heuristic is designed for grouped instances and performs equally well when each group has exactly one container.

The proposed heuristic is tested with instances possessing more than 7000 containers, and it demonstrates its applicability in practice. In practice, the real retrieval sequence can be affected by other operations taking place in the yard at the same time retrieval for a vessel is performed. Because our algorithm solves real-sized instances in a fraction of a second, the computer will be able to adjust the plan in a response time that can be considered as real-time for practical purposes. It was also observed that using multiple spreaders can reduce both the RMGC working time and the number of container movements.

Numerical experiments provided useful insights that pointed to promising future studies. It was observed that the number of movements increased when all of the stacks were arranged in an upside-down fashion, but the average working time for each movement decreased, pointing to possible new strategies for yard planning. The test data also demonstrated the tradeoff between the number of movements and the working time. While the former has been the focus of many related studies, the latter must also be concerned in practice. With a heuristic that can generate large number of real-sized solutions quickly, one might be able to identify more significant findings with data mining or similar methods in the future.

The current research can be extended in several future directions. Only RMGCs were considered in this work, but it would be interesting to investigate scenarios in which different types of lifting equipment are used to execute the retrieving plan generated by our proposed method. Next, because pre-marshaling export containers before loading commences is another approach that reduces the loading time (Lee and Hsu, 2007), the current approach might allow the simultaneous investigation of the retrieving problem and the pre-marshaling problem. Taking advantage of the fast-solving ability of this heuristic, one can also go one step further to integrate the container retrieval problem with the ship planning problem, so that yard layout can be taken into consideration while assigning the location of containers on the vessel. Another important extension is to explore the optimal coordination of multiple RMGCs in the same yard, which typically share the same rail. Finally, the method developed in this work might be very useful to floating container terminals (Baird and Rother, 2013), which is still at its very early stage of development, since they are designed to perform loading, unloading, and transshipment of containers on very limited space.

Acknowledgments

The authors acknowledge the National Research Council, Taiwan, for providing partial funding support under contract number NSC98-2221-E-006-151-MY2 and MOST 103-2410-H-006-079-MY3. The contents of the article remain the sole responsibility of the authors.

References

Baird, A.J., Rother, D., 2013. Technical and economic evaluation of the floating container storage and transhipment terminal (FCSTT). Transport. Res. Part C: Emerg. Technol. 30, 178–192.
Caserta, M., Voß, S., Sniedovich, M., 2011. Applying the corridor method to a blocks relocation problem. OR Spectrum 33 (4), 915–929.
Choe, R., Park, T., Oh, M.-S., Kang, J., Ryu, K.R., 2011. Generating a rehandling-free intra-block remarshaling plan for an automated container yard. J. Intell. Manuf. 22 (2), 201–217.
Dekker, R., Voogd, P., Van Asperen, E., 2006. Advanced methods for container stacking. OR Spektrum 28 (4), 563–586.
Forster, F., Bortfeldt, A., 2011. A tree search heuristic for the container retrieval problem. In: Operations Research Proceedings 2011. Springer, pp. 257–262.
Forster, F., Bortfeldt, A., 2012. A tree search procedure for the container relocation problem. Comput. Oper. Res. 39 (2), 299–309.
Han, Y., Lee, L.H., Chew, E.P., Tan, K.C., 2008. A yard storage strategy for minimizing traffic congestion in a marine container transshipment hub. OR Spectrum 30 (4), 697–720.
Kim, K.H., Hong, G.-P., 2006. A heuristic rule for relocating blocks. Comput. Oper. Res. 33 (4), 940–954.
Kim, K.H., Park, Y.M., Ryu, K.-R., 2000. Deriving decision rules to locate export containers in container yards. Eur. J. Oper. Res. 124 (1), 89–101.
Lee, Y., Hsu, N.-Y., 2007. An optimization model for the container pre-marshalling problem. Comput. Oper. Res. 34 (11), 3295–3313.
Lee, Y., Lee, Y.-J., 2010. A heuristic for retrieving containers from a yard. Comput. Oper. Res. 37 (6), 1139–1147.
Meisel, F., Wichmann, M., 2010. Container sequencing for quay cranes with internal reshuffles. OR Spectrum 32 (3), 569–591.
Meng, Q., Wang, S., Andersson, H., Thun, K., 2014. Containership routing and scheduling in liner shipping: overview and future research directions. Transport. Sci. 48 (2), 265–280.
Stahlbock, R., Voss, S., 2008. Operations research at container terminals: a literature update. OR Spectrum 30 (1), 1–52.
Tran, N.K., Haasis, H.-D., 2015. An empirical study of fleet expansion and growth of ship size in container liner shipping. Int. J. Prod. Econ. 159.
Vis, I.F.A., de Koster, R., 2003. Transshipment of containers at a container terminal: an overview. Eur. J. Oper. Res. 147 (1), 1–16.
Wan, Y.-W., Liu, J., Tsai, P.-C., 2009. The assignment of storage locations to containers for a container stack. Naval Res. Logist. 56 (8), 699–713.
Zhang, C., Chen, W., Shi, L., Zheng, L., 2010. A note on deriving decision rules to locate export containers in container yards. Eur. J. Oper. Res. 205 (2), 483–485.