Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 34 (2007) 3295–3313
computers & operations research
www.elsevier.com/locate/cor

An optimization model for the container pre-marshalling problem

Yusin Lee*, Nai-Yun Hsu

Department of Civil Engineering, National Cheng Kung University, Tainan 701, Taiwan

Available online 24 February 2006

Abstract

In most container yards around the world, containers are stacked high to utilize yard space more efficiently. In these yards, one major factor that affects their operational efficiency is the need to re-shuffle containers when accessing a container that is buried beneath other containers. One way to achieve higher loading efficiency is to pre-marshal the containers in such a way that it fits the loading sequence. In this research, we present a mathematical model for the container pre-marshalling problem. With respect to a given yard layout and a given sequence that containers are loaded onto a ship, the model yields a plan to re-position the export containers within the yard, so that no extra re-handles will be needed during the loading operation. The optimization goal is to minimize the number of container movements during pre-marshalling. The resulting model is an integer programming model composed of a multi-commodity flow problem and a set of side constraints. Several possible variations of the model as well as a solution heuristic are also discussed. Computation results are provided.

Keywords: Network flows; Container pre-marshalling; Integer programming; Optimization

1. Introduction

Containerization is the packing of cargo into large, designated boxes of standard dimensions, enabling multiple units of cargo to be handled simultaneously. Today, much of the world's international cargo moves in containers through major seaports. Because containerships are highly capitalized and their operating costs are significant, it is very important that their turn-around time in container terminals be as short as possible. Thus, in order to meet the demand for container traffic both effectively and efficiently, management of container terminal operations has become crucial.

One of the ways to measure port container terminal performance is by the berthing time at port. Most of the berthing time of a containership consists of the time it takes to load and unload containers. In order to reduce the loading time, a terminal operator must arrange storage locations for containers in the yard in such a manner that they can be loaded efficiently onto a containership.

Export containers arrive at the terminal in random order. Typically, containers start arriving at the terminal as early as 7 days in advance and containers scheduled for earlier ships are likely to arrive earlier than those that are scheduled for later ships. Therefore, by the time loading starts for a particular containership, it is often the case that many of the containers for this current ship are buried in the container stacks beneath other containers waiting to be loaded to a later ship. Containers can also be stacked in the wrong order due to lack of accurate information or other reasons [1].

* Corresponding author. Tel.: +886 6275 7575.
E-mail address: yusin@mail.ncku.edu.tw (Y. Lee).

0305-0548/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2005.12.006

Moreover, containers are stored on board according to the stowage plan, which specifies the location of each and every container on the ship. The stowage plan largely determines the order that the containers are to be loaded onto the ship. Because containers in the yard can be accessed only from the top of a stack, re-handles will be needed if the target container is not located at the top. A re-handle is the movement of a container, which is required to permit access to another container. Re-handles are very time consuming and have a major impact on the performance of the port. One way to reduce the number of re-handles while loading is through pre-marshalling. In a container yard, pre-marshalling means to re-position the export containers before the loading process starts, so that the containers can be loaded with few or no re-handles. Pre-marshalling requires additional cost, but is executed when the ship has to be loaded as fast as possible. In this research, we develop a model to optimize the pre-marshalling process.

There are not many research results published about optimizing the pre-marshalling process in a yard. Kim and Bae [2] discussed how to shift the containers efficiently to speed up the loading process that follows. However, Kim and Bae's research only takes into consideration the movement of the transfer cranes between bays, and does not consider the movements of individual containers within one bay, which is the main topic of this paper. Kim et al. [3] proposed a method to determine the storage location of containers in the yard according to their weight. Kim [4] developed a method to evaluate the expected number of re-handles required to reach a specific container, and also the total number of re-handles for picking up all the containers in a given yard configuration. Kozan [5] analyzed the operation of sea ports and used a network model to describe the process of containers in the port area and between various transportation modes. Yun and Choi [6] developed a simulation model to analyze the operation of container ports. Kim and Kim [7] proposed a model to optimize the loading operation for export containers, but the scope of this model is also limited to the routing of container carriers. Later, Kim and Kim [8] proposed an algorithm to determine the working route of a transfer crane for a similar loading operation. Again, the movements of individual containers within a bay is not considered in detail. Finally, Zhang et al. [9] studied the storage space allocation problem in a complex terminal yard with inbound, outbound and transit containers mixed together.

The operation strategy of any container yard has to be developed according to the equipment that particular container yard uses. In this research, we focus on yards that use rail mounted gantry crane (RMGC) as its major container handling equipment. RMGC is one of the most widely used container handling equipment worldwide. As illustrated in Fig. 1, when an RMGC lifts a container from the yard, the crane first positions itself at the bay where the target container resides, then positions its trolley over the right stack, then lowers its spreader to hold the container and lifts it up. After lifting the container, the RMGC can move the trolley to one end of the crane and lower the container to a waiting truck, or place the container on the top of another stack in the same bay. For safety reasons, it is usually prohibited to move the crane while carrying a container. If moving a container from one bay to another bay is necessary, the crane has to put the container on a truck first. Then both the crane and the truck move to the target bay. Finally, the crane picks up the container from the truck, and put it on top of the right stack. This type of operation is extremely time consuming and is avoided whenever possible.

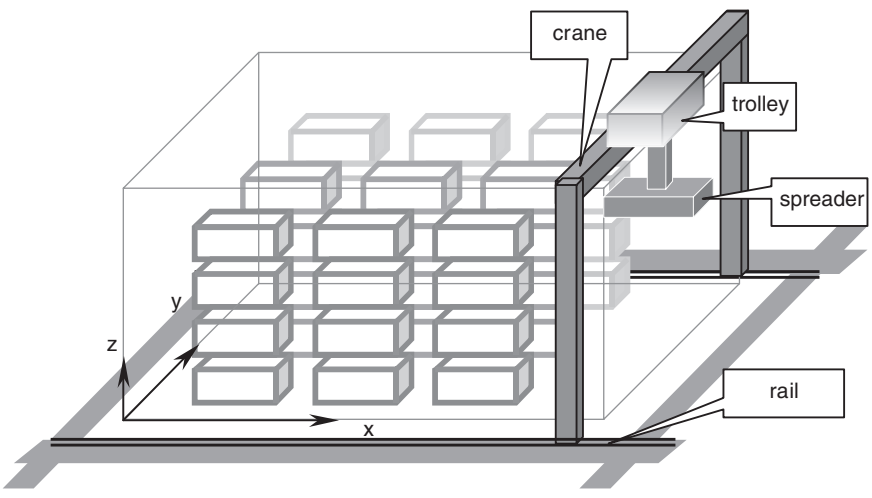


Fig. 1. Illustration of an RMGC.

In this research, we develop a mathematical model to solve for the optimal solution of the container re-marshalling problem. The movements of individual containers are investigated. Given a container yard configuration, the model yields a plan to relocate the containers within the yard, in order to eliminate the need of re-handling in the subsequent loading operation. The optimization goal is to minimize the number of container movements in the pre-marshalling process. The resulting model is a network flow-based integer program. The model can be solved with branch and bound or any other algorithm designed for the general integer programming problem. We also discuss several possible extensions of the model that can be obtained by modifying some of the constraints, and propose a heuristic to solve the model more efficiently.

This paper is divided into seven sections. Following this introduction, the second section will define the container pre-marshalling problem. The mathematical model will be developed in detail in the third section, followed by discussions of several possible extensions in the fourth section. Section 5 discusses how this large-scale problem can be solved. Computational examples are provided in the sixth section, and a brief conclusion is presented in the final section.

2. The container pre-marshalling problem

One of the ways to measure port container terminal performance is by the berthing time at port. Most of the berthing time of a containership consists of the time it takes to load and unload containers. In order to reduce loading time, a terminal operator must arrange storage locations for containers in the yard in such a manner that they can be loaded efficiently onto a containership.

Given an initial layout of a container yard, the container pre-marshalling problem solves for a working sequence to shuffle the containers around, so that the container yard will reach a final layout that satisfies certain criteria. Two major constraints to this problem are that the stacks can be accessed from the top only, and that all the pre-marshalling work has to be carried out within the yard. In other words, containers are not allowed to be moved out of the yard for temporary storage, then moved back to the yard at a later time.

The basic assumptions used in this research are listed below:

1. Pre-marshalling takes place in the same bay. This assumption follows from the fact that in practice, it is extremely time consuming to move containers between bays in yards equipped with RMGC. This assumption also implies that we need to consider only one RMGC.
2. Containers of different dimensions are stored in different stacks. The vast majority of containers are either 20- or 40-feet long. In principle, containers of different lengths can be mixed together in one single stack. For example, one 40-feet container can rest on two 20-feet containers (but not the other way round). However, doing this complicates yard operations and is avoided whenever possible in practice. For simplicity, in the model developed below, we will assume that all the containers are of the same length.
3. Only one ship is presumed to be present. Although it is possible for large ports to have two or more ships being loaded at the same time with containers taken from the same yard, served by the same RMGC, we do not consider this scenario.
4. The loading order of the containers is known. In practice, the loading sequence of a ship is determined well before loading starts (more than 6 h in most cases).

3. The model

The model developed in this research is an integer programming model which has a multi-commodity network flow problem embedded within. In this section, we first introduce the structure of the corresponding network, then we present a detailed description of the constraints, and finally the objective function of the mathematical model.

3.1. The network model

The network embedded in the model is a multi-commodity flow problem based on a time–space network. In the model, time is discretized into *time segments* separated by *time points*. The time points thus represent snapshots of the

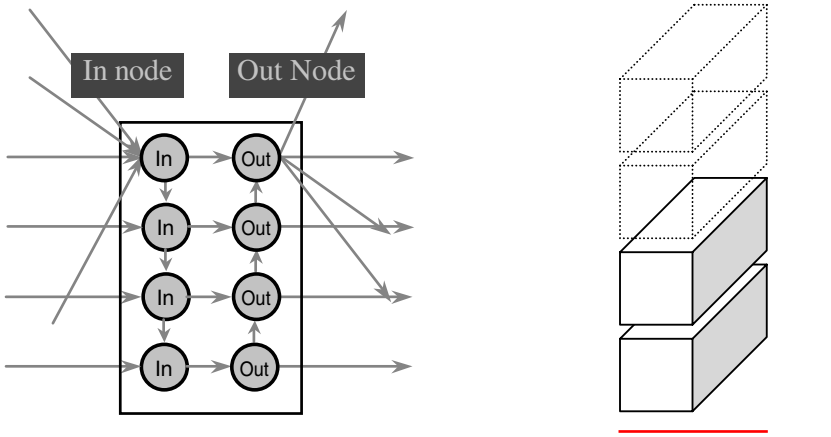
layout of the container yard, while the movements of containers take place in time segments. As we will introduce later, we measure the workload associated with a pre-marshalling sequence in terms of the number of movements in that sequence, and the optimization goal is to minimize the number of movements required to transform the container yard from its initial layout to its final layout. Under this concept, the actual time needed for each container movement as well as the actual length of a time segment is irrelevant for optimization purposes. The basic model allows one container movement to take place in each time segment, and we assume that the length of each time segment is equal to the time needed to perform the movement allocated to that time segment. An extension of the model allows for multiple movements to take place in each time segment. In that case we will assume that each time segment is long enough for the RMGC to finish all movements scheduled for that time segment. This variant of the model, as well as other possible extensions, will be discussed in a later section.

In practice, while all containers differ more or less from each other in shipping destination, weight, contents, priority, or other attributes, they are often classified into groups according to their shipping destination (and probably according to a few additional attributes) for pre-marshalling purposes. In our model, we make a similar assumption that the containers can be grouped into C types, numbered consecutively from 1 to C, with each type corresponding to one flow commodity in the network model. The number of types C is a parameter that can be set according to the user's choice without affecting basic properties or the structure of the network. However, this parameter has a significant influence on the problem size and thus the computational time needed to solve the problem instance.

Based on the concept described above, the network model captures the movements of the containers in time–space in the container yard. The nodes in the network correspond to the slots that hold the containers, the arcs correspond to container movement opportunities, and the flows in the network represent the movements of the containers in time–space.

Four parameters determine the size of the network. In addition to C as defined above, T represents the number of time points in the network. The first and last time points, which mark the start point and end point of the planning horizon, are numbered 1 and T, respectively. It follows from this definition that there are $T-1$ time segments in the model. The time segment between time points t and $t+1$ will be referred to as segment t. A large T will significantly increase the size of the network. However, the problem might become infeasible if T is too small. The third parameter is S, which is defined as the number of container stacks in the model. Since pre-marshalling takes place in one single bay, S is the width of the bay. Finally, H is the maximum height of each stack. The stacks will be numbered 1 to S in the model, and the slots in each stack will be numbered 1 to H, where 1 represents the slot at the bottom tier, and H is the slot at the top. A slot in the bay will be referred to as slot h of stack s, and a slot in the time–space model is frequently referred to as slot h of stack s at time t.

The network is composed of basic network units as illustrated in Fig. 2. The figure illustrates a basic network unit for a stack of maximum height $H=4$. Fig. 2(a) shows one basic network unit, which corresponds to one time point of one



(a) One basic unit of the network model, with 4 pairs of nodes and corresponding arcs

(b) The corresponding stack, with 4 slots holding 2 containers

Fig. 2. A basic network unit in the model.

stack in the container yard. Fig. 2(b) shows the corresponding container stack. The stack shown in the figure has four slots, and holds two containers, leaving the upper two slots empty. There are two types of nodes in each basic network unit, namely the *push in nodes* and the *pop out nodes*. In Fig. 2(a), the four nodes on the left-hand side are push in nodes and the other four nodes are pop out nodes. A pair of push in and pop out nodes at the same level corresponds to one slot. Therefore, there are $2H$ nodes in each unit. The push in node corresponding to slot h that belongs to stack s at time point t is referred to as node IN_{tsh}. Similarly, the pop out node for the same slot is referred to as node OUT_{tsh}.

There are five types of arcs in the network, also illustrated in Fig. 2(a). All arcs are directed and an arc connecting from node a to node b will be referred to as arc (a, b) in this paper. The first type of arc is called upward arc, which connects from the pop out node of one slot h to the pop out node of slot $h + 1$ in the same stack. An arc of this type is referred to as arc $(OUT_{tsh}, OUT_{ts(h+1)})$. This arc provides a mechanism for containers to be moved upward in a stack. Because no container will be removed from any stack at time point T (which is the final time point), there will be no such arc in the basic network units corresponding to the final time point.

The second type of arc is the *downward arc*. These arcs connect from the push in node of slot h to the push in node of slot $h - 1$ in the same stack, and are referred to as arc $(IN_{tsh}, IN_{ts(h-1)})$. The flows on these arcs correspond to containers moving downwards in a stack after being placed in from the top. For obvious reasons, basic network units corresponding to the first time point will not have this type of arcs. The third type of arc is called an internal arc. Each of these arcs connects one push in node and its corresponding pop out node in the same slot. An arc of this type corresponding to slot h at time t, stack s is referred to as (IN_{tsh}, OUT_{tsh}). This third type of arc does not represent any container movement. Instead, they provide connections between consecutive time segments of the same slot. Each arc of this type corresponds to one slot at one time point, and will carry flow if and only if a container occupies the slot at that time instance.

The fourth type is the stationary arc. A stationary arc connects one pop out node in one time point and another push in node in the following time point of the same slot. An arc corresponding to slot h of stack s, and spans time points t and $t + 1$ is referred to as arc $(OUT_{tsh}, IN_{(t+1)sh})$. The flows on these arcs correspond to containers that remain unmoved between time points t and $t + 1$. The final arc type is the *movement arc*. An arc of this type connects one pop out node of slot H, stack s, and time t, to one push in node of slot H, another stack z, and time $t + 1$, and is referred to as arc $(OUT_{tsH}, IN_{(t+1)zH})$. In order to provide enough opportunities for containers to move freely between stacks, movement arcs are established between all pairs of stacks across consecutive time points. Most of the arcs in the network are of this type.

In the more general (but rare) case that not all containers in the bay are of the same length, we assume that containers of different lengths are stored in different stacks. This assumption is consistent with reality because mixing containers of different length in the same stack will make the stack unstable and hard to handle. In this case movement arc $(OUT_{tsH}, IN_{(t+1)zH})$ exists only if stacks s and z hold containers of the same length. All other components of the model remain the same.

Fig. 3 illustrates a network model composed of S stacks, $T - 1$ time segments, and maximum height $H = 4$. It can be seen in this illustration that in addition to the nodes and arcs introduced above, there is one source node, one sink node, and a set of arcs connecting to these two nodes. From the source node there is one arc connecting to each push in node at the initial time point. These arcs carry flow from the source node into the network, which defines the initial condition of the container yard. From each of the pop out nodes at the final time point there is also one arc connecting to the sink node. These arcs simply provide paths for all containers to flow into the sink node upon termination of the planning horizon.

Each arc in the network can carry flows of different commodities. These flows correspond to the movements of the containers in time and space. We illustrate this with Fig. 4. This figure shows a yard whose parameters H, S, T, and C are all set to 3. Not all movement arcs are shown for simplicity. A feasible solution is also shown in this figure. The arcs that carry flow in the solution are shown in thick lines, and the numbers shown in the nodes correspond to the flow commodity that passes through that node in this particular solution. All these arcs carry just one unit of flow due to side constraints that will be introduced later in this section. Based on the flows originating from the source node, one can see that at the beginning of the planning horizon, there are three containers in the yard. Stack 1 has one container of type 3 and another container of type 1 placed above it. Stack 2 has one container of type 2, and stack 3 is empty. At time point 1, the flow-carrying arcs show that the type 1 container moves upward through an upward arc. Then it was lifted out of stack 1 and entered stack 3 at time point 2. Because the stack is empty, the container is lowered to the bottom of the stack through downward arcs. Between time points 2 and 3, the flow moves on a stationary arc, indicating

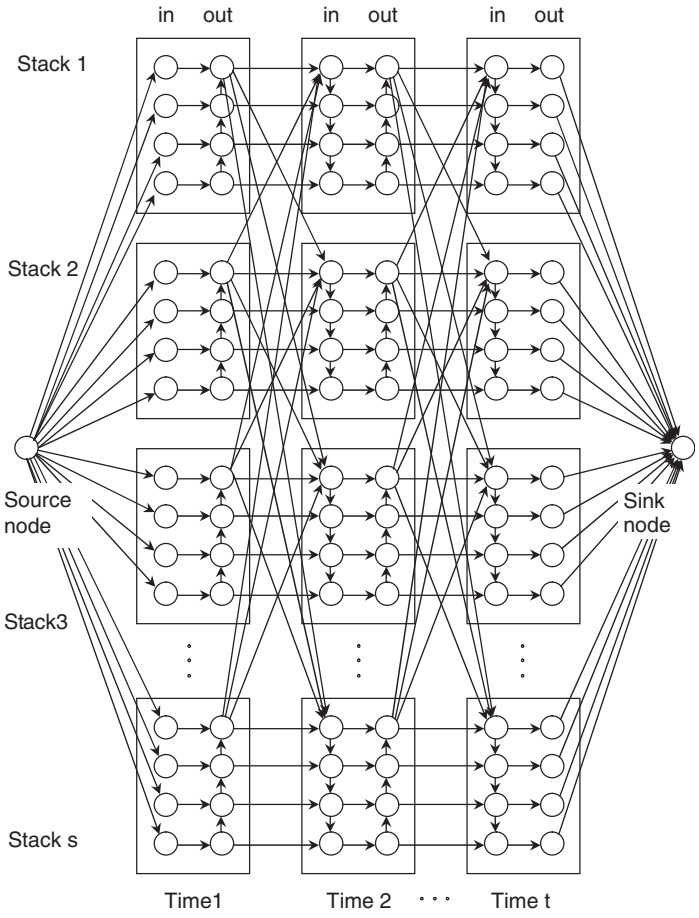


Fig. 3. Illustration of the network model.

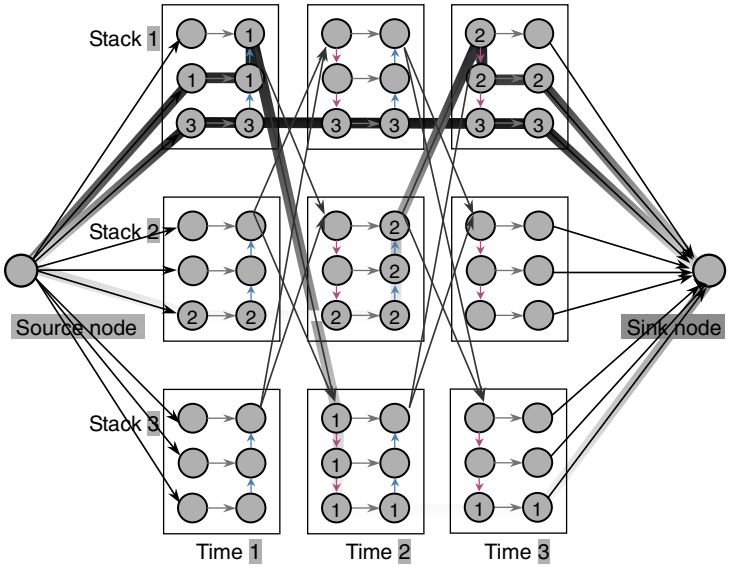


Fig. 4. Illustration of a model and a feasible solution.

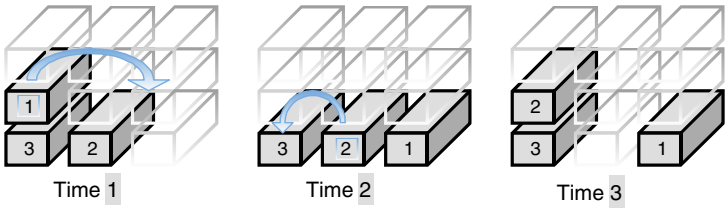


Fig. 5. The layouts of the yard corresponding to Fig. 4.

that the container stays unmoved. Finally, the flow enters the sink node. The movements of the other two containers in time and space can also be traced in a similar way. The container of type 3 remains unmoved throughout the planning period, and finally the type 2 container was lifted from stack 2 to stack 1 at time segment 2. Fig. 5 shows figures of the yard at the three time points.

3.2. The mathematical model

We are now ready to introduce the components that make up the mathematical model. The constraints describe the multi-commodity flow network defined above, plus additional side constraints. Altogether, the model captures the topological layout of the container yard, as well as the physical rules that the containers have to follow while being moved around the yard. The optimization goal is to minimize the number of movements required to transform the container yard from its initial layout to its final layout. The resulting model is a linear integer programming problem.

Several sets and parameters are defined and used in the model. The first set $COMS = \{1, \ldots, C\}$ is the set of all container types. As discussed above, this is also the set of commodities in the network. Set $TIME = \{1, \ldots, T\}$ is the set of all time points, $STACK = \{1, \ldots, S\}$ is the set of all stacks, and $HEIGHT = \{1, \ldots, H\}$ is the set of all possible slot numbers. For the initial time point, we define a parameter ${}^{c}SUPPLY_{sh}$. At the initial layout, if slot h at stack s holds a container of type c, this parameter will be set to 1. Otherwise, it will be set to 0. Table 1 shows a list of the parameters and sets used in this paper.

There are seven different types of decision variables in the model, all of them are binary integer variables, and all of them correspond to flows in the network. The definitions of these variables, together with some additional variables that we will introduce later when we extend the model, are listed in Table 2.

Of the variables defined in Table 2, the value of the ${}^{c}sp_{sh}$ variables can be derived directly from the initial layout of the yard, therefore, these variables can be treated as a known constant. The last type of variables, ${}^{c}dm_{sh}$, is also known if the final layout of the yard is given and required, as we will see in an extended version of the model. For the time being, we will regard the final layout as unknown and treat ${}^{c}dm_{sh}$ as a variable.

Since the objective function is relatively simple as compared to the constraints, we will introduce the constraints first and discuss the objective function afterwards. There are four sets of constraints in the model. The first set expresses the basic physical rules that the containers have to follow. Constraint (1) states that any container must be placed at the bottom of a stack or on top of another container. In network flow terminology, this constraint states that for any given stack s at time point t, if an internal arc carries flow, then the internal arc directly beneath it (if there is one) has to carry flow as well

$$\sum_{c \in COMS} {}^{c}i^{t}_{sh} \geqslant \sum_{c \in COMS} {}^{c}i^{t}_{s(h+1)} \quad \forall t \in TIME, \quad s \in STACK, \quad h \in HEIGHT \backslash \{H\}. \tag{1}$$

Constraint (2) states that when an upward arc carries flow, the upward arc directly above it should carry flow as well. By setting this constraint, whenever a container moves upward in a stack, it will move all the way up to the top and will never stop at an intermediate slot

$${}^{c}u^{t}_{sh} \leqslant {}^{c}u^{t}_{s(h+1)} \quad \forall c \in COMS, \quad t \in TIME \backslash \{T\}, \quad s \in STACK, \quad h \in HEIGHT \backslash \{H, H-1\}. \tag{2}$$

Constraint (3) states that containers can be accessed from the top of a stack only. Consider stack s at time t. The first item on the left-hand side of the constraint is the amount of flow carried on the internal arc $(IN_{ts(h+1)}, OUT_{ts(h+1)})$ corresponding to the slot $h+1$, and the second item is the flow on the upward arc $(OUT_{tsh}, OUT_{ts(h+1)})$ beneath it.

Table 1
Definitions of parameters and sets

Name	Definition
C	The number of container types
T	The number of time points
S	The number of stacks
H	The maximum height of each stack
${}^{c}SUPPLY_{sh}$	Set to 1 if slot h at stack s holds a container of type c in the initial layout, and set to 0 otherwise
${}^{c}DEMAND_{sh}$	Set to 1 if slot h at stack s has to hold a container of type c in the final layout, and set to 0 otherwise
${}^{c}LEAVE^{t}$	The total number of type c containers that are due to be removed from the yard at time t
$COMS$	The set of all container types $\{1, \ldots, C\}$
$TIME$	The set of all time points $\{1, \ldots, T\}$
$STACK$	The set of all stacks $\{1, \ldots, S\}$
$HEIGHT$	The set of all possible slot numbers $\{1, \ldots, H\}$

Table 2
Definitions of decision variables

Variable name	Definition
${}^{c}u^{t}_{sh}$	Type c flow on the upward arc $(OUT_{tsh}, OUT_{ts(h+1)})$
${}^{c}d^{t}_{sh}$	Type c flow on the downward arc $(IN_{tsh}, IN_{ts(h-1)})$
${}^{c}i^{t}_{sh}$	Type c on the internal arc (IN_{tsh}, OUT_{tsh})
${}^{c}o^{t}_{sh}$	Type c flow on the stationary arc $(OUT_{tsh}, IN_{(t+1)sh})$
${}^{c}r^{t}_{sz}$	Type c flow on the movement arc $(OUT_{tsH}, IN_{(t+1)zH})$
${}^{c}sp_{sh}$	Type c flow on the arc $(Source\ node, IN_{1sh})$
${}^{c}dm_{sh}$	Type c flow on the arc$(OUT_{Tsh}, Sink\ node)$
${}^{c}v^{t}_{s}$	Type c containers moved out of the yard from stack s at time t. Used in extended model

Because $\sum_{c\in COMS} {}^{c}i^{t}_{s(h+1)}=1$ if and only if slot $h+1$ of stack s is occupied by a container at time t, and $\sum_{c\in COMS} {}^{c}u^{t}_{sh}=1$ if and only if a container is moving upward from slot h to slot $h+1$, this constraint will prevent any container from moving upward as long as a container is present at a higher slot

$$\sum_{c\in COMS} {}^{c}i^{t}_{s(h+1)} + \sum_{c\in COMS} {}^{c}u^{t}_{sh} \leqslant 1 \quad \forall t \in TIME\backslash\{T\},\ \ s \in STACK,\ \ h \in HEIGHT\backslash\{H\}. \tag{3}$$

Constraint (4) states that only one container is allowed to move during each time segment. Since each movement corresponds to one unit of flow in one movement arc, this constraint states that for each time segment, the sum of flow on all the movement arcs must not exceed 1

$$\sum_{s\in STACK} \sum_{\substack{z\in STACK \\ s\neq z}} \sum_{c\in COMS} {}^{c}r^{t}_{sz} \leqslant 1 \quad \forall t \in TIME\backslash\{T\}. \tag{4}$$

The following set of constraints (5)–(9) state that only one unit of flow, which corresponds to one container, can be carried on each arc. The left-hand side of each constraint is the sum of all flows corresponding to all container types, which is exactly the total flow on each corresponding arc. Note that constraint (4) implies constraint (9). In the next section we will discuss how to relax constraint (4) in order to gain better computation efficiency. Therefore, we keep constraint (9) in this model

$$\sum_{c\in COMS} {}^{c}u^{t}_{sh} \leqslant 1 \quad \forall t \in TIME\backslash\{T\},\ \ s \in STACK,\ \ h \in HEIGHT\backslash\{H\}, \tag{5}$$

$$\sum_{c\in COMS} {}^{c}i^{t}_{sh} \leqslant 1 \quad \forall t \in TIME,\ \ s \in STACK,\ \ h \in HEIGHT, \tag{6}$$

$$\sum_{c\in COMS} {}^{c}d_{sh}^{t} \leqslant 1 \quad \forall t \in TIME\backslash\{1\},\ \ s \in STACK,\ \ h \in HEIGHT\backslash\{1\}, \tag{7}$$

$$\sum_{c\in COMS} {}^{c}o_{sh}^{t} \leqslant 1 \quad \forall t \in TIME\backslash\{T\},\ \ s \in STACK,\ \ h \in HEIGHT, \tag{8}$$

$$\sum_{c\in COMS} {}^{c}r_{sz}^{t} \leqslant 1 \quad \forall t \in TIME\backslash\{T\},\ \ s, z \in STACK,\ \ s \neq z. \tag{9}$$

The following pair of constraints corresponds to the initial and final layouts of the yard. Constraint (10) states that flows of the correct commodity, as specified by the given initial layout, have to enter the push in nodes of time point 1 from the source node. Next, constraint (11) states that these flows must move into the corresponding internal arcs. Note that while constraint (10) sets the exact location of every container in the initial layout, the model in its current form does not have another similar constraint to spell out the final layout in detail. This will be discussed later when we introduce several possible model extensions

$${}^{c}sp_{sh} = {}^{c}SUPPLY_{sh} \quad \forall c \in COMS,\ \ s \in STACK,\ \ h \in HEIGHT, \tag{10}$$

$${}^{c}sp_{sh} = {}^{c}i_{sh}^{1} \quad \forall c \in COMS,\ \ s \in STACK,\ \ h \in HEIGHT. \tag{11}$$

Constraint (12) makes the flow enter the sink node from the pop out nodes that belong to the final time point

$${}^{c}dm_{sh} = {}^{c}i_{sh}^{T} \quad \forall c \in COMS,\ \ s \in STACK,\ \ h \in HEIGHT. \tag{12}$$

The next set of constraints is the flow conservation constraints for each node. Flow conservation for push in nodes is maintained by constraints (13)–(15). Consider the push in node corresponding to the bottom slot at stack s, time t. Flow can enter this node from the pop out node of the same slot at the previous time point or from the push in node of the slot above it at this time point. Since there is no downward arc attached to this node, the flow that enters has to exit from its internal arc. This requirement is expressed in constraint (13), where the sum of the entering flow is represented on the left-hand side of the equation, and the exiting flow is on the right-hand side

$${}^{c}o_{s1}^{(t-1)} + {}^{c}d_{s2}^{t} = {}^{c}i_{s1}^{t} \quad \forall c \in COMS,\ \ t \in TIME\backslash\{1\},\ \ s \in STACK. \tag{13}$$

Constraint (14) is the flow conservation constraint for push in nodes corresponding to the slots that are not at the top, nor at the bottom of stacks. For such nodes, flow can enter from the pop out node of the same slot at the previous time point or from the push in node of the slot above at the same time point. These flows are summed up on the left-hand side of constraint (14). To exit the node, flow can take the downward arc that leads to the push in node of the slot beneath, or the internal arc that leads to the pop out node of the same slot. These exiting flows are listed on the right-hand side of constraint (14)

$${}^{c}o_{sh}^{(t-1)} + {}^{c}d_{s(h+1)}^{t} = {}^{c}d_{sh}^{t} + {}^{c}i_{sh}^{t} \quad \forall c \in COMS,\ \ t \in TIME\backslash\{1\},\ \ s \in STACK,\ \ h \in HEIGHT\backslash\{1, H\}. \tag{14}$$

Equivalently, constraint (14) can be interpreted intuitively as follows. Consider a slot that is not at the top, nor at the bottom of a stack. At a certain time t, a container can enter the slot from the top of the stack or can be carried over from the previous time point. In either case, the container has to remain at this slot or move downwards to the slot beneath it. Because of constraint (1), any container entering from the top will not remain at a slot unless there is already another container beneath it.

Constraint (15) sets the flow conservation requirement for the push in nodes located at the top of stacks. The only difference between constraints (14) and (15) is the second item on the left-hand side. Instead of having the possibility that containers might enter the slot from the one above, this item in constraint (15) is the sum of the flows that enter the corresponding downward node from all movement arcs ending at this node. These flows correspond to the container(s) being moved into this stack from other stacks

$${}^{c}o_{sH}^{(t-1)} + \sum_{\substack{z\in STACK \\ z\neq s}} {}^{c}r_{zs}^{(t-1)} = {}^{c}d_{sH}^{t} + {}^{c}i_{sH}^{t} \quad \forall c \in COMS,\ \ t \in TIME\backslash\{1\},\ \ s \in STACK. \tag{15}$$

Constraint (16) ensures that no more than one unit of flow enters each stack at any time

$$\sum_{c\in COMS} {}^{c}o_{sH}^{(t-1)} + \sum_{c\in COMS} \sum_{\substack{z\in STACK \\ z\neq s}} {}^{c}r_{zs}^{(t-1)} \leqslant 1 \quad \forall t \in TIME\backslash\{1\},\ \ s \in STACK. \tag{16}$$

Constraints (17)–(19) are flow conservation constraints for pop out nodes. These three constraints correspond to pop out nodes located at the bottom of a stack, intermediate slots of a stack, and at the top of a stack, respectively. Details of these constraints are similar to those of constraints (13)–(15) and will be omitted from the paper

$${}^{c}i_{s1}^{t} = {}^{c}u_{s1}^{t} + {}^{c}o_{s1}^{t} \quad \forall c \in COMS,\ \ t \in TIME\backslash\{T\},\ \ s \in STACK, \tag{17}$$

$${}^{c}i_{sh}^{t} + {}^{c}u_{s(h-1)}^{t} = {}^{c}u_{sh}^{t} + {}^{c}o_{sh}^{t} \quad \forall c \in COMS,\ \ t \in TIME\backslash\{T\},\ \ s \in STACK,\ \ h \in HEIGHT\backslash\{1, H\}, \tag{18}$$

$${}^{c}i_{sH}^{t} + {}^{c}u_{s(H-1)}^{t} = {}^{c}o_{sH}^{t} + \sum_{\substack{z\in STACK \\ z\neq s}} {}^{c}r_{sz}^{t} \quad \forall c \in COMS,\ \ t \in TIME\backslash\{T\},\ \ s \in STACK. \tag{19}$$

The purpose of container yard pre-marshalling is to reduce or eliminate the situation where a container is being placed on top of another container that is scheduled to be moved out of the yard at an earlier time. Here, we assume without loss of generality that containers of smaller type numbers are always moved out earlier than those of larger type numbers. For example, a container of type 2 leaves the yard earlier than another container of type 3. It follows that when the pre-marshalling operation is completed, no container should be placed above another container of a smaller type. For example, after pre-marshalling, a stack should not have containers of types 3, 3, 1, 1, 2 (listed from the bottom up), because a container of type 2 should not be placed on top of another container of type 1. This requirement is stated mathematically in constraint (20)

$$\sum_{c1\in COMS} c_1 \times {}^{c1}dm_{sh} \geqslant \sum_{c2\in COMS} c_2 \times {}^{c2}dm_{s(h+1)} \quad \forall s \in STACK,\ \ h \in HEIGHT\backslash\{H\}. \tag{20}$$

By definition, ${}^{c1}dm_{sh} = 1$ if a container of type c_1 resides at slot h of stack s in the final layout, and ${}^{c1}dm_{sh} = 0$ if the slot is empty. Since by constraint (6) each slot can hold at most one container at a time, at most one item in the summation on either side of constraint (20) can be non-zero. Therefore, the left-hand side of constraint (20) will equal to the type number of the container if the slot is occupied, or equal to 0 if the slot is empty. For example, if a container of type 2 resides at slot h of stack s in the final layout, the left-hand side of the constraint will have the value 2. The same holds for slot $h + 1$ as well, which is represented on the right-hand side of the constraint. Together, constraint (20) states that slot $h + 1$ has to hold a container of type number smaller or equal to the one in slot h, or be empty.

The last set of constraints is the binary constraints for all variables

$${}^{c}u_{sh}^{t} \in \{0, 1\} \quad \forall c \in COMS,\ \ t \in TIME\backslash\{T\},\ \ s \in STACK,\ \ h \in HEIGHT\backslash\{H\}, \tag{21}$$

$${}^{c}d_{sh}^{t} \in \{0, 1\} \quad \forall c \in COMS,\ \ t \in TIME\backslash\{1\},\ \ s \in STACK,\ \ h \in HEIGHT\backslash\{1\}, \tag{22}$$

$${}^{c}i_{sh}^{t} \in \{0, 1\} \quad \forall c \in COMS,\ \ t \in TIME,\ \ s \in STACK,\ \ h \in HEIGHT, \tag{23}$$

$${}^{c}o_{sh}^{t} \in \{0, 1\} \quad \forall c \in COMS,\ \ t \in TIME\backslash\{T\},\ \ s \in STACK,\ \ h \in HEIGHT, \tag{24}$$

$${}^{c}r_{sz}^{t} \in \{0, 1\} \quad \forall c \in COMS,\ \ t \in TIME\backslash\{T\},\ \ s \in STACK,\ \ z \in STACK,\ \ s \neq z. \tag{25}$$

The optimization goal is to minimize the total number of container movements for the pre-marshalling operation. Since each container movement contributes to one unit of flow in one of the movement arcs, the objective function is to minimize the total flow in these arcs, as stated

$$\text{Min} \sum_{\substack{t\in TIME \\ t\neq T}} \sum_{s\in STACK} \sum_{\substack{z\in STACK \\ z\neq s}} \sum_{c\in COMS} {}^{c}r_{sz}^{t}. \tag{26}$$

4. Model extensions

In this section, we will discuss three ways to extend the model so that it can be used for various requirements. We shall refer to the model (1)–(26) as the basic model.

The three different extensions of the model accommodate for different requirements on the final layout of the container yard. First, consider the case that the exact locations of all containers at the final stage are specified (to the extent that the container type for each slot is given). For this situation, one needs to introduce a set of parameters ${}^{c}DEMAND_{sh}$. Similar to ${}^{c}SUPPLY_{sh}$, ${}^{c}DEMAND_{sh}$ is set to 1 if at the final layout, slot h at stack s is required to hold a container of type c, and is set to 0, if otherwise. This set of new parameters appears in constraint (27), which sets the final layout in a way similar to how constraint (10) sets the initial layout of the yard

$$ {}^{c}dm_{sh} = DEMAND_{sh} \quad \forall c \in COMS, \quad s \in STACK, \quad h \in HEIGHT. \tag{27} $$

With constraint (27) in the model, constraint (20) is no longer needed. The purpose of the latter is to ensure that containers are stacked in the proper order at the final layout. This requirement becomes redundant once the type of container for each slot is specified in detail.

Sometimes yard managers prefer that containers be sorted in such a way that each stack holds just one container type. The basic model can be extended to satisfy this requirement by adding constraint (28). This constraint functions by stating that for a given stack s, if there is a container of type c_1 at slot $h+1$, then slot h cannot have any container that is of a different type. Recall that ${}^{c}dm_{sh}$ is the flow of type c on the arc $(OUT_{tsh}, Sink\ node)$. Therefore, the first item in constraint (28) ${}^{c1}dm_{s(h+1)} = 1$ implies that at the final layout, slot $h+1$ in stack s is occupied by a container of type c_1. In this case, the constraint requires that slot h of the same stack should not be occupied by any container of type $c_2 \neq c_1$. Since the slot beneath an occupied slot cannot be empty, constraint (28) thus implies that if slot $h+1$ holds a container, slot h has to hold a container of the same type

$$ {}^{c1}dm_{s(h+1)} + \sum_{c2 \in COMS \setminus \{c1\}} {}^{c2}dm_{sh} \leqslant 1 \quad c_1 \in COMS, \quad s \in STACK, \quad h \in HEIGHT \setminus \{H\}. \tag{28} $$

When constraint (28) is present, each stack will be either empty or have exactly one container type in the final layout. Therefore, constraint (20) will no longer be needed and thus can be removed from the basic model.

Finally, we discuss an extension of the model that allows containers to leave the yard (for example, be loaded onto the ship) at the same time when pre-marshalling takes place. This type of yard operation is common in practice, and the loading and pre-marshalling operations are mostly done by the same RMGC. For this purpose, we introduce one extra sink node for each time segment into the model to represent the ship being loaded. This extra sink node is connected to the pop out node at the top (i.e., height H) of each stack at the corresponding time point. Fig. 6 illustrates one basic unit of the network model with the extra sink node and its connecting arc shown. In the multi-commodity flow model, the flow on this arc corresponds to an integer variable ${}^{c}v_{s}^{t}$, which is defined as the number of type

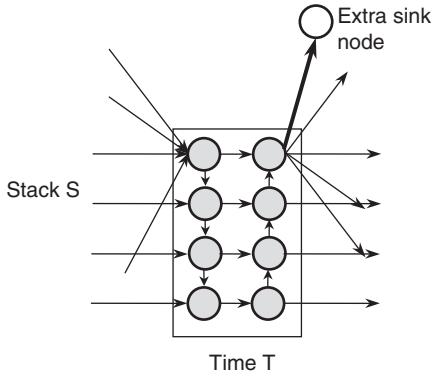


Fig. 6. Illustration of basic network unit with extra sink node shown.

c containers being lifted out of stack s at time t to leave the yard. We also define a parameter ${}^{c}LEAVE^{t}$, which is the total number of type c containers that are due to be removed from the yard at time t. Similar to the previous assumption that the loading sequence of containers is given, we also assume that ${}^{c}LEAVE^{t}$ is known. Constraint (29) requires that the correct number of containers of each type be removed from the yard at the required time

$$\sum_{s \in STACK} {}^{c}v_{s}^{t} = {}^{c}LEAVE^{t} \quad \forall c \in COMS, \quad t \in TIME \backslash \{1\}. \tag{29}$$

The left-hand side of constraint (29) is the total number of containers of type c being lifted out of all stacks at time t. The constraint requires that this summation have to be equal to the required amount ${}^{c}LEAVE^{t}$. Because of the introduction of constraint (29), we need to replace the flow conservation constraint for the pop out nodes at the highest tier (19) with constraint (30). This new constraint differs from constraint (19) in that it allows containers be removed from the yard by adding the last item on the right-hand side

$${}^{c}i_{sH}^{t} + {}^{c}u_{s(H-1)}^{t} = {}^{c}o_{sH}^{t} + \sum_{\substack{z \in STACK \\ z \neq s}} {}^{c}r_{szH}^{t} + {}^{c}v_{s}^{t} \quad \forall c \in COMS, \quad t \in TIME \backslash \{1\}, \quad s \in STACK. \tag{30}$$

An illustration of a full network for this extension is shown in Fig. 7. Each time point except for the initial one corresponds to one extra sink node together with the connecting arcs as defined above.

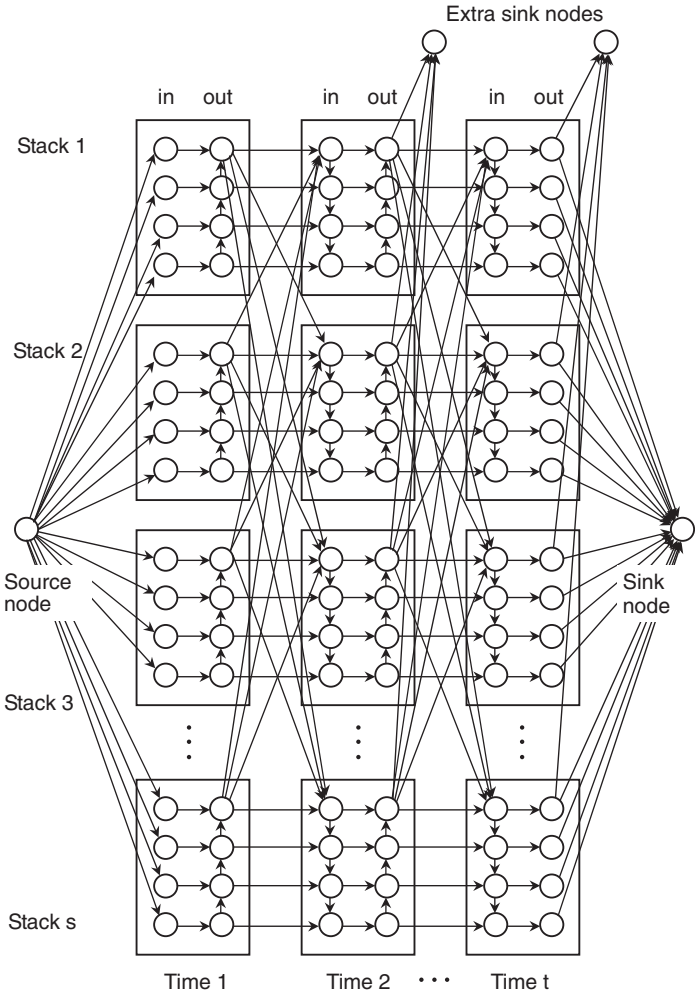


Fig. 7. Illustration of an extended full network model.

5. Model solving

The model developed in this research, either in its basic form or in one of the three extended forms, is a large-scale integer program, which can take significant time for the computer to solve. Here, we discuss some strategies to simplify the model in order to gain better computation efficiency, and propose a heuristic approach for the model.

In the basic model, the parameter T determines the number of time segments, and the size of the problem grows proportionally with T. However, if T is less than the minimum number of movements required to transform the container yard from its initial layout to the required condition (i.e., no containers of larger type number be put on top of another container of smaller type number), the problem will become infeasible. One way to reduce T without overly compromising the quality of the solution is to allow for multiple container movements to occur in each time segment. If the model allows multiple containers be moved in one time segment, T can be reduced considerably compared to the basic model, which allows only one container movement per each time segment. The requirement that only one container can be moved in any time segment is imposed by constraint (4). Thus, relaxing this constraint will remove this requirement completely. However, doing so allows cycles to appear in the solution as explained below. Let (i, j) be a movement that carries a container from stack i to stack j. A cyclic movement is a set of movements (i_1, i_2), $(i_2, i_3), \ldots, (i_k, i_1)$ that forms a cycle and all the movements occur in the same time period as illustrated in Fig. 8. In this example, movements (1, 2) and (2, 1) form a cycle of length 2 since they both occur in the same time period. Multiple movements that all occur in the same time segment can be completed if and only if there are no cyclic movements embedded. To avoid cycles of length 2, one can introduce constraint (31), which states that at any time segment t, containers cannot be moved simultaneously from stack s to another stack z, and also from stack z back to stack s

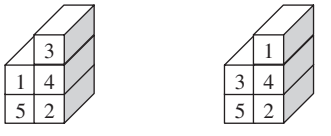
$$\sum_{c\in COMS} {}^{c}r^{t}_{sz} + \sum_{c\in COMS} {}^{c}r^{t}_{zs} \leqslant 1 \quad \forall t \in TIME\backslash\{T\}, \quad s, z \in STACK, \quad z \neq s. \tag{31}$$

Similarly, constraint (32) prevents cycles of length 3 from forming in the solution. For any combination of three different stacks q, s, and z, this constraint states that containers cannot be moved from stack s to z, from z to q, and from q to s during the same time segment, thus preventing the cycle from forming

$$\sum_{c\in COMS} {}^{c}r^{t}_{sz} + \sum_{c\in COMS} {}^{c}r^{t}_{zq} + \sum_{c\in COMS} {}^{c}r^{t}_{qs} \leqslant 2 \quad \forall t \in TIME\backslash\{T\}, \quad s, z, q \in STACK, \quad z \neq s, \quad s \neq q, \quad q \neq z. \tag{32}$$

Longer cycles can be prevented by adding similar constraints. However, the number of these constraints grows exponentially with the cycle length. In the case that a cycle of length 4 or longer does appear in the solution, it can be easily broken by replacing one of the movements in the cycle with two other movements. For example, a cycle (1, 2), (2, 3), (3, 1) can be broken by replacing (1, 2) with (1, 4) and (4, 2). It is possible that doing so might result in a sub-optimal solution, but the effect is very small and should be acceptable for practical purposes. Another possibility is to include the cycle breaking constraints (31) and (32) in the model while keeping constraint (4) at the same time, but relax constraint (4) by replacing the right-hand side with 3. In this configuration, the model allows up to three movements in each time segment, and cycles will not appear in any feasible solution.

When multiple movements are allowed in one time segment, the container movements still have to be ordered carefully even when cycles do not exist. For example, consider the case that the solution calls for moving container A from stack 1 to stack 2, and container B from stack 2 to stack 3 in one time segment. This pair of movements does not form a cycle, but clearly container B has to be lifted out of stack 2 before container A is placed in. Given a set



(a) Before movement. (b) After movement.

Fig. 8. Illustration of a cyclic movement of length 2.

of movements that all take place in the same time segment, one can order them correctly according to the yard layout before and after this time segment. Yard layout at any time point t can be obtained from the status of the internal arcs (which correspond to variables ${}^{c}i^{t}_{sh}$ in the model). The following strategy can be used to determine the movement sequence within one time segment. The strategy is simple and we state it without proof of correctness.

Consider the set of movements $M = \{(s_1, t_1), (s_2, t_2), \ldots, (s_p, t_p)\}$ that belong to a time segment T^*. The time points before and after this time segment are T^* and $T^* + 1$, respectively. Starting from the yard layout at time point T^*, identify a movement in M that can be performed without conflict. (Conflict occurs when the movement is blocked by another container at the pop-out stack, or blocks; a later movement at the push-in stack.) Then, we update the yard layout accordingly, and remove this movement from M. If M is not empty, yet no such movement can be found, cycles exist. In this case, replace one of the movements in the cycle with two new movements to break the cycle as discussed above. The process repeats until M is empty.

Another possible modification of the basic model that can also reduce T is to relax constraints (3), (4), (5), and (21), and introduce constraints (33) and (34). This replacement of constraints allows multiple containers be moved out of a stack in one time segment. Doing so will enable the model to be solved more efficiently by using less time segments. The role of constraint (33) is to prevent too many containers from entering and exiting a stack at the same time point

$$\sum_{c \in COMS} {}^{c}u^{t}_{sh} \leqslant h \quad \forall t \in TIME, \quad s \in STACK, \quad h \in HEIGHT \backslash \{H\}. \tag{33}$$

The purpose of constraint (34) is to replace constraint (3). In the basic model, the purpose of constraint (3) is to prevent any container from moving upward when there is another container stacked above it. In network flow terminology, this constraint prevents flow from entering an upward arc when the internal arc above it carries flow. This constraint also prevents more than one container be lifted from one stack in a given time segment. Removing constraint (3) allows one to lift more than one container from a stack at a time segment, but also allows one to lift a container out of a stack without first removing those stacked above it. In constraint (34) M is a constant greater than H. This constraint relates the upward arc of a slot with the stationary arc of the slot above it. If a container occupies slot $h + 1$ of stack s at time segment t, then $\sum_{c \in COMS} {}^{c}o^{t}_{s(h+1)} = 1$. It follows that the second term on the left-hand side has to be equal to 0, which prohibits the container at slot h from moving upward. On the other hand, if $\sum_{c \in COMS} {}^{c}o^{t}_{s(h+1)} = 0$, then this constraint will be non-binding

$$M \sum_{c \in COMS} {}^{c}o^{t}_{s(h+1)} + \sum_{c \in COMS} {}^{c}u^{t}_{sh} \leqslant M \quad \forall t \in TIME \backslash \{T\}, \quad s \in STACK, \quad h \in HEIGHT \backslash \{H\}. \tag{34}$$

The strategy introduced above is able to speed up the solution process considerably. However, due to the complex nature of mixed integer problems, applications are still limited to small-scale problems with present-day personal computers and solution engines. As we will see in the examples provided in the next section, an instance with 14 containers, 6 stacks, and maximum height of 4 can take significant amount of time to solve. In practice, an average bay has 12 stacks, maximum height 4, and holds approximately 35 containers [1]. Next we present a simple heuristic that enables the computer to handle problems that exceed this size.

The heuristic solves the problem in two phases. In the first phase, the heuristic attempts to solve for a set of movements that leads to a good final layout. This phase runs quickly, but the solution quality is low in that it can contain multiple cycles. In the second phase, the heuristic sorts the movements and breaks all cycles by including additional movements. Both phases work iteratively by repeatedly solving variants of the basic model.

The basic idea of the first phase is as follows. Observe that most of the arcs in the basic model are movement arcs, but only a small portion of them are used in the final solution. If one could include only a small number of movement arcs in the model, and if those arcs are chosen properly, then the problem can be solved much faster yet still yield a good solution. Another efficient method that eases the computation burden is to further reduce the T value. To solve the problem with a small number of time segments, one can replace constraints (3), (4), (5), and (21), with constraints (33) and (34) in the basic model as explained above, and further relax constraint (7) to allow multiple number of containers to enter each stack during each time segment. After these modifications, the model can be solved with $T = 2$, in which all movements take place in one time segment. However, these movements will have no particular order, and cycles will be present in most cases. Therefore, in the second phase the heuristic orders the movements and breaks any cycles that exist. Details of the heuristic are explained below.

First, we introduce how the first phase of the heuristic works. In this phase the model is generated with only some of the movements. Then the model is solved without constraint (20). This constraint requires all containers be stacked in the correct order in the final layout. Instead of prohibiting, we penalize those that are stacked in the wrong order in the final layout. After solving the model with the branch-and-bound method, we keep the movement arcs that are used, replace those that are not used with a set of randomly selected movement arcs, and solve again. This process is repeated until the final layout satisfies constraint (20), and the best solution (one that uses the least number of movements) is used as the input for the second phase. The heuristic uses the following rule to generate the initial set of movement arcs. For each stack s_0 that has a type c container, randomly pick two different stacks $s_1 \neq s_0$ and $s_2 \neq s_0$, generate the movement arc that connects from stack s_1 to s_1, and also the movement arc that connects from stack s_1 to s_2. Only containers of type c are allowed to flow on these arcs. In addition to these arcs, we also include a few randomly selected movement arcs.

The movements obtained in the first phase almost always contain multiple cycles. In the second phase, we attempt to obtain a feasible ordering of these movements and break all the cycles by introducing new movements. This phase uses a network model similar to that of the first phase. The movement arcs used in the model are those chosen in the first phase, together with some randomly selected movement arcs. Unlike the first phase, in the second phase we set $T = 3$, which corresponds to two time segments. The two time segments use the same set of movement arcs, but in the first time segment only three movements are allowed, and cycle breaking constraints (31) and (32) are deployed. In the second time segment no cycle breaking constraints are used, and there is no limit on the number of movement arcs selected. After the problem is solved, the first time segment will contain three cycle-free movements, which can be easily ordered and recorded. The movements that remain in the second time segment are then used in the next iteration. The second phase terminates when the second time segment has no movements.

An example demonstrating the performance of this heuristic will be provided in the next section.

6. Computational results

In this section, we provide computational examples for the model and its extensions introduced earlier. They are solved on a personal computer equipped with a 3.2 GHz CPU and 1024 M memory space. The solver is written in C++, combined with CPLEX 9.0 as its optimization engine, which in turn uses the branch-and-bound method to solve the problem. In all cases we require that in the final layout, containers with a larger type number should never be stacked on top of another container of a smaller type number.

The initial layout used in the examples is shown in Fig. 9. There are 6 stacks and 14 containers in the yard. In Fig. 9, stack 1 is the stack on the far left and stack 6 is on the far right. The containers belong to 3 types, and the type numbers of containers are shown in the figure. The maximum height of each stack is 4. In the following, we will use an ordered pair (a, b) to represent a movement of one container from stack a to stack b, and use a sequence of ordered pairs to represent a solution of the model, which is an ordered list of container movements.

The computational results of 36 examples are listed in Table 3. The basic model is used in the first two examples. These two examples differ only in parameter T, which specifies the number of time points in the network model. The first example uses $T = 11$ and took more than 70,000 s to solve. The second example uses $T = 10$, and was solved in < 7000 s. The two examples resulted in different solutions (with the same objective function value), but the final layout is the same, as shown in Fig. 10. The problem becomes infeasible when $T < 10$.

Examples 3–5 demonstrate the effect of replacing constraint (20) with (27), which requires the final layout to be as shown in Fig. 11. Example 3 solves with $T = 12$ and took 230 s to complete. Example 4 is set with$T = 11$, and the CPU time is reduced to 42 s. When T is further reduced to 10 as in example 5, the computation time increased to 102 s. Again, the model becomes infeasible for $T < 10$. By comparing the computational results one can observe that the problem can be solved a lot faster when the final layout is given.

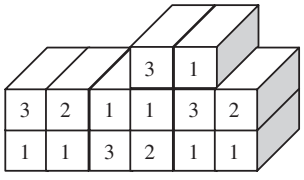


Fig. 9. Initial layout of the test cases.

Table 3
Testing results

Example	T	CPU	Solution
Basic model			
1	11	70582	{(2, 6), (2, 3), (1, 2), (5, 1), (5, 2), (5, 1), (4, 3), (6, 2), (6, 2)}
2	10	6802	{(2, 6), (3, 2), (4, 3), (1, 3), (4, 2), (6, 4), (6, 4), (5, 6), (5, 3)}
Target final layout specified			
3	12	230	{(6, 2), (5, 6), (3, 6), (1, 3), (4, 3), (1, 4), (5, 1), (2, 1), (2, 1)}
4	11	42	{(6, 2), (3, 6), (4, 3), (1, 3), (5, 4), (1, 6), (5, 1), (2, 1), (2, 1)}
5	10	102	{(6, 2), (3, 6), (4, 3), (5, 4), (1, 3), (1, 6), (5, 1), (2, 1), (2, 1)}
Containers moved out of the yard			
6	12	24974	Time 6: (5, 3), time 7: (4, 3), time 9: (1, 3)
7	11	3195	Time 5: (4, 3), time 7: (1, 3), time 8: (5, 3)
8	10	2930	Time 6: (5, 3), time 7: (1, 3), time 8: (4, 3)
9	9	364	Time 4: (5, 3), time 6: (1, 3), time 7: (4, 3)
10	8	576	Time 3: (4, 3), time 4: (5, 3), time 5: (1, 3)
11	7	146	Time 3: (4, 3), time 4: (5, 3), time 5: (1, 3)
Multiple containers can move in the same time segment, no cycle breaking			
12	10	13	Time 2: {(1, 3), (3, 1)} Time 4: {(4, 3)} Time 5: {(4, 1)} Time 8: {(2, 4), (5, 2)} Time 9: {(5, 3), (6, 4)}
13	9	18	Time 4: {(1, 3), (3, 1)} Time 5: {(4, 3)} Time 6: {(5, 1)} Time 7: {(2, 4), (4, 2)} Time 8: {(5, 3), (6, 4)}
14	8	7	Time 2: {(1, 3), (3, 1)} Time 3: {(5, 1)} Time 5: {(4, 3)} Time 6: {(2, 4), (4, 2)} Time 7: {(5, 3), (6, 4)}
15	7	10	Time 1: {(1, 3), (3, 1)} Time 2: {(5, 1)} Time 4: {(4, 3)} Time 5: {(2, 4), (4, 2)} Time 6: {(5, 3), (6, 4)}
16	6	4	Time 1: {(1, 3), (3, 1)} Time 2: {(4, 3)} Time 3: {(4, 6), (5, 1), (6, 4)} Time 5: {(2, 4), (5, 3)}
17	5	3	Time 1: {(1, 3), (3, 1)} Time 2: {(4, 3), (5, 1)} Time 3: {(2, 4), (4, 2)} Time 4: {(6, 4), (5, 3)}
18	4	1	Time 1: {(6, 2), (3, 6), (4, 3)}Time 2: {(1, 3), (4, 6), (2, 4), (5, 1)} Time 3: {(2, 4), (5, 3)}
Same as 12–18, 2-*cycle breaking constraint applied*			
19	10	224	Time 3: {(3, 2), (1, 3)} Time 5: {(5, 1)} Time 6: {(4, 3)} Time 7: {(4, 1)} Time 8: {(6, 4), (2, 6)} Time 9: {(2, 4), (5, 3)}
20	9	58	Time 2: {(3, 5), (4, 3)} Time 3: {(1, 3), (5, 1)} Time 6: {(4, 1), (6, 4)} Time 7: {(5, 6)} Time 8: {(2, 4), (5, 3)}
21	8	47	Time 1: {(3, 6), (4, 3)} Time 2: {(1, 3), (4, 1)} Time 3: {(2, 4), (5, 2), (6, 1)} Time 4: {(6, 4)} Time 7: {(5, 3)}
22	7	19	Time 1: {(3, 6), (4, 3), (6, 1)} Time 2: {(4, 6), (2, 4), (5, 2)} Time 3: {(1, 4), (5, 3)} Time 6: {(1, 3)}
23	6	6	Time 1: {(3, 2), (4, 3)} Time 2: {(1, 3), (4, 1)} Time 3: {(5, 1)} Time 4: {(6, 4), (2, 6)} Time 5: {(2, 4), (5, 3)}
24	5	3	Time 1: {(1, 3), (2, 6), (3, 2)} Time 2: {(4, 3)} Time 3: {(4, 1), (5, 2), (6, 4)} Time 4: {(5, 3), (6, 4)}
25	4	2	Time 1: {(6, 2), (3, 6), (4, 3)} Time 2: {(1, 3), (2, 4), (4, 1), (5, 6)} Time 2: {(2, 4), (5, 3)}
Same as 12–18, 2- *and* 3-*cycle breaking constraints applied*			
26	10	181	Time 1: {(3, 5)} Time 2: {(1, 3), (5, 1)} Time 3: {(4, 3)} Time 4: {(4, 1), (6, 4)} Time 5: {(2, 4)} Time 6: {(5, 6)} Time 9: {(5, 3)}
27	9	146	Time 1: {(5, 6), (6, 2)} Time 2: {(3, 6), (4, 3)} Time 3: {(4, 5), (2, 4), (5, 3)} Time 8: {(1, 3), (2, 4)}
28	8	81	Time 1: {(3, 2), (1, 3)} Time 2: {(4, 3)} Time 3: {(4, 1), (6, 4)} Time 5: {(5, 6)} Time 6: {(2, 1)} Time 7: {(2, 4), (5, 3)}
29	7	28	Time 1: {(6, 2), (3, 6), (1, 3)} Time 3: {(5, 1)} Time 4: {(4, 3)} Time 5: {(4, 1), (2, 4)} Time 6: {(2, 4), (5, 3)}
30	6	14	Time 2: {(3, 6), (1, 3), (5, 1)} Time 3: {(4, 3)} Time 4: {(4, 1), (2, 4), (6, 2)} Time 5: {(5, 3), (6, 4)}
31	5	5	Time 1: {(2, 6), (3, 2), (1, 3), (5, 1)} Time 2: {(4, 3)} Time 3: {(4, 1), (6, 4)} Time 4: {(5, 3), (6, 4)}
32	4	3	Time 1: {(1, 6), (3, 1), (4, 3)} Time 2: {(4, 1), (2, 4), (5, 2), (6, 3)} Time 3: {(5, 3), (6, 4)}
One container type in each stack			
33	13	26847	{(1, 2), (3, 1), (2, 3), (4, 3), (4, 1), (6, 4), (2, 4), (5, 2), (5, 3)}
34	12	17725	{(1, 2), (3, 1), (2, 3), (4, 3), (4, 1), (6, 4), (2, 4), (5, 2), (5, 3)}
35	11	4876	{(2, 6), (3, 2), (5, 2), (4, 3), (5, 3), (4, 5), (6, 4), (6, 4), (1, 3)}
36	10	2096	{(3, 2), (4, 3), (1, 3), (2, 1), (4, 1), (2, 4), (6, 4), (5, 2), (5, 3)}

Examples 6–11 test the model extension that some containers have to be moved out of the yard at the same time pre-marshalling takes place. We assume that two containers of type 1 and one container of type 2 have to be moved out at time 3, and one type 2 container needs to be moved out at time 5. In addition to these demands, no mis-overlays are allowed in the final layout. When $T = 12$ as in example 6, it took the computer almost 25,000 s to solve. The computing time decreases quickly as T is reduced to 7, as demonstrated in examples 7–11 that follow. The results of example 6 are shown in Fig. 12.

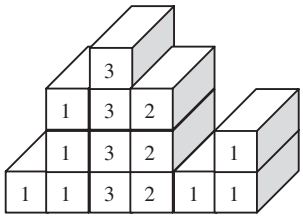


Fig. 10. Final layout of examples 1 and 2.

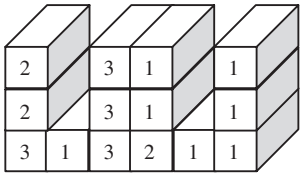
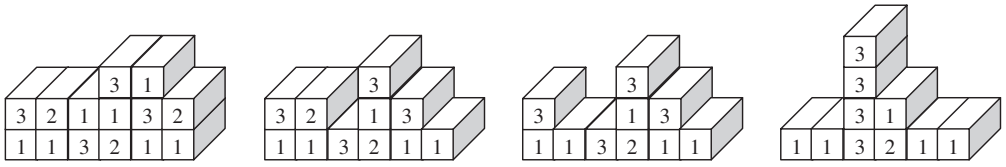


Fig. 11. Required final layout of tests 3–5.



(a) Initial layout. (b) After time 3 (c) After time 5 (d) Final layout

Fig. 12. Results of example 6.

Next, examples 12–18 test the effect of allowing multiple containers to be moved in each time segment. This is done by relaxing constraint (4). No cycle breaking constraints are applied. These examples are solved very quickly as shown in Table 3. From these results one can observe that the CPU time is less than those of the basic model by several orders of magnitude. One can also see that cycles of length 2 appear in almost every solution. Finally, by comparing the solutions one can see that as T decreases, more containers are moved in each time segment. However, the number of movement steps (which is the minimization goal) remains the same except for example 18, which is also the only result in this group that does not contain a cycle. Figs. 13 and 14 show the final layout of examples 12 and 18, respectively.

Examples 19–25 solve the same models as those in 12–18 with the same set of T values and other parameters, except that the 2-cycle breaking constraint (31) is applied. As a result, the computation times increased significantly. Although the 3-cycle breaking constraint (32) is absent from the models, no cycles appeared in the solutions. Fig. 15 shows the results of example 25, which has only 3 time segments in the model. When constraint (32) is added to the model as in examples 26–32, the CPU times increase further.

In examples 32–36 we require that there is only one container type in each stack in the final layout. All five testing resulted in the same final layout as shown in Fig. 16.

Finally, we provide example 37 to demonstrate the proposed heuristic. The problem has 45 containers grouped into 6 types. Fig. 17 shows the initial layout of the containers, which are arranged into 12 stacks. The maximum height of each stack is 5. The first phase took 306 s and yielded a set of 31 movements: (6, 8), (7, 8), (8, 5), (9, 11), (1, 5), (1, 10), (3, 10), (7, 10), (8, 10), (3, 4), (5, 4), (9, 12), (10, 4), (12, 4), (3, 1), (5, 1), (11, 12), (12, 1), (2, 3), (4, 9), (7, 3), (7, 9), (9, 3), (12, 9), (3, 12), (5, 7), (6, 12), (8, 3), (9, 7), (10, 7), and (11, 7). One can observe that there are many cycles embedded, for example (7, 10) and (10, 7) is a cycle of length 2, and (3, 10), (10, 4), (4, 9), (9, 7), (7, 3) is a cycle of length 5.

The second phase sorted the movements and broke all cycles. The final solution has 47 movements: (4, 1), (5, 4), (12, 4), (5, 2), (10, 12), (10, 4), (6, 8), (1, 12), (1, 10), (1, 10), (3, 1), (3, 10), (11, 3), (11, 1), (8, 11), (8, 4), (8, 10),

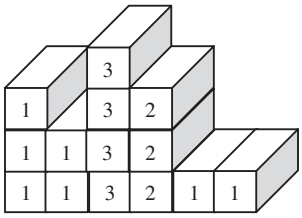
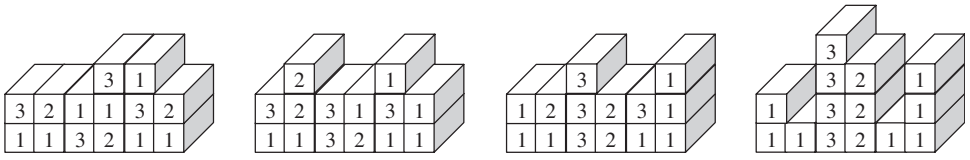
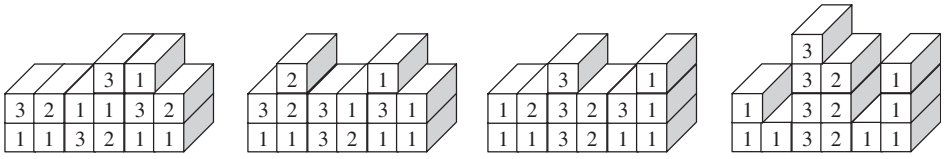


Fig. 13. Final layout of example 12.



(a) Initial layout (b) After time 0 (c) After time 1 (d) Final layout

Fig. 14. Final layout of example 18.



(a) Initial layout (b) After time 0 (c) After time 1 (d) Final layout

Fig. 15. Results of example 25.

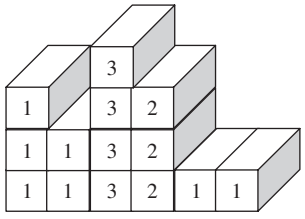


Fig. 16. Final layout of examples 33–36.

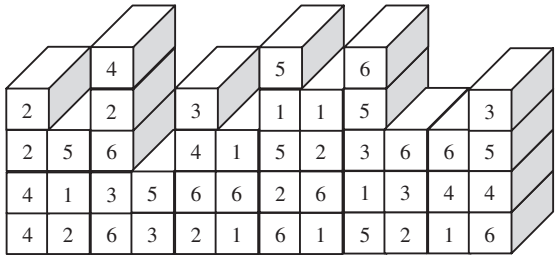


Fig. 17. Initial layout of example 37.

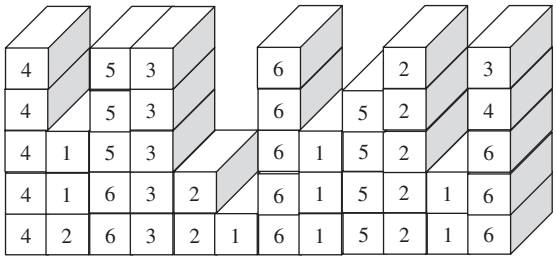


Fig. 18. Final layout of example 37.

(8, 1), (4, 8), (7, 11), (7, 8), (7, 4), (7, 6), (5, 7), (6, 5), (3, 7), (9, 7), (3, 7), (12, 3), (12, 9), (12, 5), (1, 5), (12, 1), (6, 12), (9, 12), (2, 12), (3, 6), (3, 12), (5, 3), (2, 3), (4, 3), (9, 3), (9, 4), (9, 2), (6, 9), (11, 9), and (5, 9). The final layout of the yard is shown in Fig. 18. The total CPU time is 318 s, including time used by both phases.

Solving the same problem of example 1 with this heuristic yields the same final layout as shown in Fig. 10. The movements obtained after phase 1 are (3, 6), (4, 2), (5, 2), (2, 4), (6, 4), (1, 3), (4, 3), (5, 3). The final solution has 10 movements (the optimal solution has 9). The movements are (6, 1), (3, 6), (4, 3), (2, 6), (5, 2), (5, 3), (4, 2), (1, 4), (6, 4), and (1, 3). The CPU time is 0.265 s.

7. Conclusion

The container pre-marshalling problem is very important in that pre-marshalling containers in the yard prior to ship loading can help in reducing the berthing time. In this paper, we developed a mathematical model for the container pre-marshalling problem. The model is based on a multi-commodity network flow model. The nodes and arcs correspond to the time–space structure of the container yard, and containers moving in time–space in the yard are represented with flows. Physical laws that containers have to comply with are specified with constraints. We also provided several extensions of the model. Some methods are developed to simplify the model to gain better computation efficiency. A simple but effective heuristic enables the computer to solve problems close to real-world problems in size. The model and solution methods are demonstrated with 37 examples.

In this paper, we investigated the container pre-marshalling problem for a single ship. The extension to the case of multiple ships is a promising topic for future research.

References

[1] Steenken D, Voß S, Stahlbock R. Container terminal operation and operations research—a classification and literature review. OR Spectrum 2004;26:3–49.
[2] Kim KH, Bae JW. Re-marshalling export containers in port container terminals. Computers and Industry Engineering 1998;35:655–8.
[3] Kim KH, Park YM, Ryu K-R. Deriving decision rules to locate export containers in container yards. European Journal of Operational Research 2000;124:89–101.
[4] Kim KH. Evaluation of the number of rehandles in container yards. Computers and Industry Engineering 1997;32:701–11.
[5] Kozan E. Optimising container transfers at multimodal terminals. Mathematical and Computer Modelling 2000;31:235–43.
[6] Yun WY, Choi YS. A simulation model for container-terminal operation analysis using an object-oriented approach. International Journal of Production Economics 1999;59:221–30.
[7] Kim KY, Kim KH. A routing algorithm for a single transfer crane to load export containers onto a containership. Computers and Industry Engineering 1997;33:673–6.
[8] Kim KH, Kim KY. An optimal routing algorithm for a transfer crane in port container terminals. Transportation Science 1999;33:17–33.
[9] Zhang C, Liu J, Wan Y-W, Murty KG, Linn RJ. Storage space allocation in container terminals. Transportation Research B 2003;37:883–903.