Smart stacking for import containers using customer information at automated container terminals

Yuanjun Feng, Dong-Ping Song[*], *Dong Li*
School of Management, University of Liverpool, Chatham Street, Liverpool, L69 7ZH, UK
[Citation: Yuanjun Feng, Dong-Ping Song, Li Dong, (2021). Smart stacking for import containers using customer information at automated container terminals, European Journal of Operational Research, (in press). https://doi.org/10.1016/j.ejor.2021.10.044.]

Abstract: Motivated by the practical import free-flow program that aims to expedite the container retrieval process, we conceptualize a new container stacking strategy, termed Smart Stacking (SS) strategy. The SS strategy aims to create relocation-free stacks (smart stacks) by utilizing customer information. The Storage Location Assignment Problem (SLAP) under the SS strategy is addressed. The problem is to determine the smart customers/containers, and the number and locations of smart stacks, when assigning a batch of import containers to a yard block at an automated container terminal to minimize the total retrieval time. Two variants of SLAP are investigated under the non-split policy and the split policy, depending on whether the containers from the same customer are allowed to be split between smart stacks and non-smart stacks. For the non-split variant, a mixed-integer programming (MIP) model is formulated first. By analyzing the properties of the optimal solution, an improved formulation with enhanced computational performance is then proposed. Based on the structure of the model, a divide-and-conquer heuristic is designed to solve the non-split variant more efficiently. For the split variant, a MIP model under the optimal partitions of the non-split model is developed. We theoretically prove that the split variant yields better results than the non-split variant. Extensive experiments are carried out to illustrate the effectiveness of smart stacking. It is found that customer information and yard utilization rate have a significant influence on the effectiveness of smart stacking.
Keywords: OR in maritime industry, import container stacking problem, smart stacking strategy, value of customer information

1. Introduction

Maritime container terminals, where containers are transferred between seaborn transport and hinterland transport, provide crucial linkages in the global container shipping network. The handling productivities and efficiencies of containers terminals are essential to ensure efficient container supply chains, which is particularly imperative in the current era where port congestion has stretched out the globe and exacerbated the global supply chain delays.

A container terminal can be divided into three main areas: seaside, landside, and storage yard (de Melo da Silva, 2018). The storage yard serves as the buffer area for storing containers before their onward transportation and links the seaside and landside operations. Import containers are discharged from ships at the seaside, unloaded into the storage yard, and then loaded to external trucks or trains at the landside, whereas the export containers follow the reverse path (Kizilay and Eliiyi, 2020). Yard operations management is of paramount importance in determining the efficiency of a terminal (Caserta et al., 2020). Inefficient container unloading and loading operations at the yard can lead to port congestions and result in longer turnaround times for vessels and hinterland transport vehicles. This paper focuses on improving the retrieval efficiency of import containers loaded to external trucks at the interface of the storage yard and the terminal's landside.

Container stacking, the theme of this paper, addresses the assignment of storage positions in the yard to containers, which directly affects the container delivery efficiency and the truck waiting time. A major source of inefficiency when retrieving containers from yards is container relocation (Ku and Arthanari, 2016a). Due to limited space in the yard, containers are piled up vertically in stacks. If a target container to be retrieved is not on the topmost tier, those above it – that is, the blocking containers - need to be moved out of the way in order to access the target one. Such moves of blocking containers are called relocation, reshuffling, or rehandling. Relocation is an unproductive operation, which is costly to terminals and results in delivery delays to customers. A series of container stacking related problems

[*] Corresponding author. Email: Dongping.Song@liverpool.ac.uk

have been addressed to reduce relocations, for example, the container relocation problem that determines the positions of relocated containers (e.g., Bacci et al., 2020; Zhang et al., 2020; Azab and Morita, 2021; Tanaka and Voß, 2021), the container pre-marshalling problem that re-arranges the container stacking positions (e.g., Parreño-Torres et al., 2019; Tanaka et al., 2019; Boge et al., 2020), the container stacking problem that pre-plans the initial stacking positions of containers (e.g., Zhang et al., 2014; Gharehgozli and Zaerpour, 2018; Boge and Knust, 2020), and the joint planning of relocation and pre-marshalling (e.g., Zweers et al., 2020). In this paper, we address one of the container stacking problems - the Storage Location Assignment Problem (SLAP), where a batch of import containers are allocated to exact locations in a storage area at the yard to minimize their future retrieval times.

The main challenge of the SLAPs for import containers lies in the uncertainty regarding which container will be retrieved first since external trucks arrive at the terminal randomly to pick up a specific container (Saurí and Martín, 2011; Yu and Qi, 2013). A couple of studies attempt to reduce the number of relocations by stacking containers based on the information of retrieval times (e.g., Lee et al., 2008; Maldonado et al., 2019). However, in reality, in most cases, the retrieval times of containers are not yet known when stacking the containers, and thus containers are often randomly stacked, which can lead to a high relocation rate. For example, in busy ports such as Los Angeles-Long Beach, it takes on average two to three relocations to deliver one container to a truck (Mongelluzzo, 2015a).

With the development of port digitalization and the need to reduce truck turnaround times, an innovative container delivery and staging program — Import Free Flow (IFF) — has been initiated at Port of Los Angeles (Mongelluzzo, 2015b). The idea behind IFF is to eliminate the need for relocation and thus realize rapid retrieval flow through pre-staging large groups of containers to be picked up by the same customer. With IFF, high-volume customers can have all their containers stored in dedicated stacks when they are unloaded from a vessel. These containers are called free-flow containers. Free-flow containers can be picked up from the top of the stacks on a last-in-first-out basis since they belong to the same customer. As a result, no relocations are needed for retrieving free-flow containers.

The IFF program has resulted in significant improvements in truck turnaround times in practice. For example, free-flow containers reduced truck turnaround times by more than 50 percent at Port of Los Angeles in 2015 (Parker, 2015). However, the IFF program has not been widely adopted in practice. In the current practice, terminal operators usually do not utilize the customer information of containers when stacking the containers either because the information is not available to them, or probably more importantly, because they have not recognized the value of the customer information and do not know how to use the information to determine the container stacking positions. The current stacking strategy in the IFF program is rather heuristic and is inadequate for its mass application. For example, the free-flow service is only available to high-volume customers who own at least 50 containers, and the free-flow containers are simply pre-staged in specific stacks that are separated from the traditional containers (i.e., non-free-flow containers) (Dupin, 2015; Parker, 2015). Expanding the free-flow service to smaller customers could dramatically improve container retrieval efficiency, which is the next goal of the practitioners (Parker, 2015). However, academic research in this regard has rarely been seen. There is no method for determining which customers or containers should be free-flowed, how many free-flow stacks should be selected, and where these stacks should be located in the yard.

Motivated by the IFF program, we conceptualized a new import container stacking strategy - Smart Stacking (SS) strategy - where import containers are grouped based on customer information, and they are classified into either smart (free-flow) or non-smart (non-free-flow) containers to be allocated to smart stacks and non-smart stacks respectively in a yard block in an optimal way. The smart containers of a customer do not share stacks with the containers from any other customers to guarantee zero relocation. The non-smart containers share stacks as normal, which still need relocations during the future retrieval process. This paper aims to investigate how customer information can be utilized to better plan the exact stacking positions for import containers so as to improve retrieval efficiency. The research objectives are: i) to seek the optimal solution for stacking a batch of import containers into a limited (pre-defined) storage area in a yard block under the proposed SS strategy; ii) to quantify the reduction in the total retrieval time by applying the SS strategy; iii) and to evaluate the impacts of relevant parameters (customer information and yard utilization rate) on the effectiveness of the SS strategy.

Our contributions to the existing literature and practice can be summarized as follows: (i) We propose a new stacking strategy – Smart Stacking (SS) strategy - to improve the import container retrieval

efficiency at container terminals. (ii) We introduce two forms of stacking policies under the framework of the SS strategy, depending on whether the containers from the same customer are allowed to be split between smart stacks and non-smart stacks or not. Correspondingly, we develop two variants of mathematical models for the Storage Location Assignment Problem (SLAP) at an Automated Container Terminal (ACT), that is, the non-split model and the split model. The proposed models enable terminal operators to determine which customers or containers should be selected to be free-flowed and to quantify the additional benefits of the splitting policy. (iii) We establish structural properties of the optimal solution to the non-split model and leverage these properties to improve the computational efficiency of the non-split model. (iv) To overcome the computational complexity, we develop a heuristic algorithm to solve the non-split variant (which is the focus of this paper) based on the structure of the model. The heuristic algorithm can obtain near-optimal solutions in several seconds. (v) We conduct extensive experiments to demonstrate the effectiveness of the SS strategy, and the impact of the customer information and the yard utilization rate on the results. The findings can help to understand the effectiveness of the SS strategy under a variety of scenarios and assess the value of customer information to container retrieval efficiency, which could promote the vertical collaboration between terminal operators, trucking companies, and cargo owners to improve the container supply chain performance.

The remainder of the paper is organized as follows. In Section 2, we review existing stacking strategies, discuss the previous work related to the container stacking problems, and summarize the research gap. Section 3 describes the problem under consideration and presents two forms of smart stacking policies. Section 4 formulates the SLAPs under the two policies by using mixed-integer programming and presents some theoretical analyses. Section 5 proposes a heuristic algorithm for the non-split model. In section 6, we conduct computational experiments to evaluate the effectiveness of the proposed strategies and generate managerial insights. Section 7 concludes the paper, discusses several extensions of this study, and envisages further research directions.

2. Literature review

Container stacking has attracted extensive attention over the last two decades (see reviews from Zhen et al., 2013; Carlo et al., 2014a, b; Lehnfeld and Knust, 2014). The problems related to container stacking include container stacking strategies, storage space allocation, storage location assignment, container relocation, and container pre-marshalling. This paper focuses on the short-term operational decision to assign a batch of import containers to exact storage locations in a yard block, which belongs to the storage location assignment problem (SLAP). Relevant literature is organized into three topics: container stacking strategies, storage location assignment, and container relocation estimation.

2.1 Container stacking strategies

Container stacking strategies are a set of stacking rules or criteria that should be adhered to when determining the storage position of each container or the storage space of a group of containers. Container stacking strategies are tactical level decisions of container terminals (Maldonado et al., 2019), which influence the allocation of stacking positions at the operational level. Several types of stacking strategies have been applied in practice and studied in the literature, which usually differ between export containers and import containers due to their different arrival and departure characteristics.

Export containers usually arrive at terminals individually and are loaded onto vessels in large batches. Their arrival times are uncertain but their departure times are relatively fixed by the destination vessels. The stacking strategies for export containers are usually based on the containers' categories or departure times, which include the residence time stacking strategy (Borgman et al., 2010), the category stacking strategy (Dekker et al., 2006), the dedicated stacking strategy and the shared stacking strategy (Gharehgozli et al., 2014; Gharehgozli and Zaerpour, 2018).

Import containers are unloaded from vessels in large volumes and then picked up by customers individually and randomly. Due to the high uncertainty in the retrieval sequences of import containers, it is difficult to categorize import containers according to their departure times. Two types of stacking strategies for import containers are commonly used in practice and have been investigated in academic research: segregation strategy and non-segregation strategy. Under the segregation strategy, containers from different ships are stacked separately in the container yard. Under the non-segregation strategy, containers from different ships are mixed in the storage area such that newly discharged containers are

stacked on top of old ones. The segregation strategy may have the advantage of reducing the number of relocations during the container retrieval process because earlier-arrived containers are likely to be retrieved earlier, but it requires additional clearing moves before each ship's arrival to create enough space for the new containers. On the other hand, the non-segregation strategy would increase relocation moves because the containers that have stayed for a longer time and thus tend to be picked up soon will be buried under recently arrived ones (De Castilho and Daganzo, 1993).

The segregation strategy and non-segregation strategy are first investigated by De Castilho and Daganzo (1993) and then are further developed by Saurí and Martín (2011). Mathematical models have been developed to optimize the stacking height under the segregation strategy (Kim and Kim, 1999) and to optimize the number of import containers allocated to each bay under both segregation and non-segregation strategies (Yu and Qi, 2013). The segregation strategy separates import containers roughly by the arriving vessels but does not specify how the containers from the same vessel are stacked. A few studies develop more detailed stacking strategies based on the container departure dates (e.g., Guldogan, 2011) and the estimated dwell times of import containers (e.g., Lee et al., 2008; Gaete et al., 2017; Maldonado et al., 2019) where containers with longer dwell times are stored under those with smaller values to reduce the number of future relocations. However, in reality, the departure dates are often not available in advance, and accurate prediction of dwell times is difficult. Besides, the containers with the same dwell time will be picked up randomly, which still incur relocations.

Different from the previous studies, in this paper, the smart stacking strategy utilizes the customer information of import containers to group containers so as to create relocation-free stacks. This can avoid the difficulty to predict the container departure time. The smart stacking strategy resembles category-based stacking or grouped-storage. Similar concepts and practice can also be identified in the storage systems of other relevant industries, such as containership stowage systems (Monaco et al., 2014; Iris et al., 2018), warehousing systems (Zaerpour et al., 2015) and generic block stacking systems (Yang and Kim, 2006; Jang et. al, 2013). However, there is a fundamental difference between our smart stacking strategy and the existing category-based stacking. The category-based stacking relies on simple criteria (e.g. container attributes, departure time, and destinations) to categorize containers into groups, which is treated as pre-determined before stacking. It does not differentiate between smart and non-smart groups/containers because each stack is allowed to store multiple groups of containers. In contrast, the smart stacking strategy incorporates intelligence into the stacking decision-making by simultaneously determining which groups/containers should be smart and optimizing the locations of the smart and non-smart stacks and the containers.

2.2 Storage location assignment problem

The determination of container storage locations is usually addressed hierarchically in two decision problems: the Storage Space Allocation Problem (SSAP) and the Storage Location Assignment Problem (SLAP) (Kim and Park, 2003; Zhang et al., 2003). The SSAP determines the amount of yard storage space allocated to each vessel for their containers, which can be addressed at various levels according to the storage space unit considered: yard section, yard block, yard sub-block, and yard bay (Jin et al., 2016). The SSAPs mainly aim to improve the efficiency of the container stacking process with efficient use of the terminal resources (e.g., Zhang et al., 2003; Lee et al., 2007; Zhen, 2016; Jiang et al., 2012, 2013; Zhou et al., 2020). The SLAP deals with the assignment of individual containers to exact storage locations – which is specified by a bay number, a row number, and a tier - in blocks. The number of relocations during the future retrieval process is an important performance measure in the SLAPs when the container retrieval efficiency is the focus (e.g., Kim et al., 2000; Zhang et al., 2010; Saurí and Martín, 2011; Zhu et al., 2020). This paper falls into the SLAP, in which we determine the exact storage location of each import container in a given storage area of a block.

The SLAP may be classified into two broad categories according to the planning approaches: online planning and offline planning. The online planning approach allocates containers to slots in a real-time way by considering the dynamic characteristics of the problem and the uncertain information on containers. For example, online-rule-based heuristics are used to determine the stacking position of each container separately in real-time (e.g, Park et al., 2011; Lin et al., 2017; Petering et al., 2017; He et al., 2019). Simulation-based methods have been used to optimize and evaluate performance measures (e.g., Dekker et al., 2006; Borgman et al., 2010; Guldogan, 2011). Moreover, facing the uncertainties and disturbances in the container stacking environment (e.g., equipment breakdown, breakage of machines,

and a fault in a container placing), decentralized approaches such as case based reasoning (Rekik et al., 2018) and multi-agent approach (Rekik and Elkosantini, 2019) are developed for the reactive container stacking systems.

The offline planning approach focuses on finding an optimal plan at the beginning of the planning period for an offline environment where the input data of the defined problems are known. This paper follows this research stream.

2.2.1 Offline SLAP

One of the challenges for addressing the offline SLAP of import containers is the presence of uncertainties in the retrieval sequence. To tackle the uncertainties, different assumptions on the containers' retrieval times have been made. Assuming that containers are retrieved with a certain probability distribution, Kim and Kim (1999) develop mathematical models to determine the optimal average stacking height to minimize the expected total number of relocations over a planning horizon. Assuming that import containers are arriving with constant rates and are retrieved with different probability of departure time, Saurí and Martín (2011) propose a probabilistic distribution-based mathematical model to estimate the number of relocations for the whole block under specific stacking strategies. The stacking strategies define the rules of mixing different groups of containers and clearing containers to reduce unproductive moves. A few studies assume that the exact retrieval time/sequence is known so that the number of relocations can be measured exactly (Chang and Zhu, 2019; Wang et al., 2020) or relocations can be completely avoided (Razouk et al., 2016). Under this assumption, Chang and Zhu (2019) develop a two-stage model for the storage space assignment of inbound containers in rail–water intermodal container terminals, which selects the optimal block for containers to balance the workload in different blocks at the first stage and assigns containers into the optimal slots to reduce the amount of overlapping (number of relocations) at the second stage. Wang et al. (2020) develop a multi-objective optimization model to minimize container overlapping amounts and crane moving distance for stacking both inbound and outbound containers in a rail-truck transshipment terminal. Razouk et al. (2016) develop a MIP model for the slot assignment of inbound containers, where the traveling distance between the berth and the storage bay is minimized and relocations are avoided. In addition, by assuming group retrieval priorities given by the truck arrival time windows, Zhu et al. (2020) combine the container stacking problem with the ship unloading problem - the inbound containers unloading and stacking problem (ICUSP) - to optimize both the container unloading sequence from the vessel and the container storage locations in the yard with the objective of minimizing the expected number of relocations during the container retrieval process.

We remark that, in reality, due to the dynamic and random arrivals of external trucks, the exact retrieval sequence of import containers is difficult to know when containers are stacked. For terminals that are equipped with a truck appointment system, relative retrieval priorities may be obtained from the truck appointment information, but only after the containers have been stacked in the yard. This is because the fact that, in most cases, appointments are not bookable until the container has already been customs cleared for pick-up, which can be days after the container have been stacked in the terminal (e.g., DP World London Gateway; The Port of Long Beach and Port of Los Angeles, 2017). In this study, we do not assume the information about the container retrieval sequence.

In the offline SLAP that deals with export containers, the container loading sequence is pre-determined to some extent by certain criteria, which makes the optimization problem relatively well defined. Optimization models including (mixed) integer programming model (Preston and Kozan, 2001; Gharehgozli and Zaerpour, 2018), dynamic programming model (Kim et al., 2000; Zhang et al., 2010; Zhang et al., 2014), and simulated annealing algorithm (Kang et al., 2006) have been proposed.

2.2.2 Objective functions

Some of the SLAPs aim at improving the efficiency of the future retrieval process (e.g., Kim et al., 2000; Saurí and Martín, 2011; Zhu et al., 2020), while some others focus on the efficiency of the stacking process (e.g., Luo et al., 2016) or the efficiency of both the stacking process and the future retrieval process (e.g., Razouk et al., 2016; Rekik and Elkosantini, 2019; Wang et al., 2020). This paper focuses on the efficiency of the future retrieval process.

In the recent decade, many terminals have been driven towards automated container terminals (ACTs) due to their low labor cost, low energy consumption, high safety, etc (Zhou et al., 2018; Wang et al.,

2019). However, the container stacking problems at the ACTs are not yet adequately studied in the literature. This paper addresses the SLAP at ACTs. The components of the objective functions of the SLAP at ACTs are different from those at the convention terminals due to the yard layout difference between the two types of terminals.

In conventional terminals, the yard blocks are typically positioned parallel to the quay, and containers are transferred between the yard crane and trucks at the empty lane at the side of each block. For export containers, this type of block layout can reduce the travel distance of the yard crane during the container retrieval process by storing export containers of the same group in the same bay to avoid the yard crane traveling across different bays. This is because export containers of the same group are usually loaded onto a ship consecutively (Kim and Park, 2003). However, for import containers, there is little chance to reduce such distance by optimizing their storage locations because the retrieval sequence of import containers is unknown. Therefore, the objective functions of the SLAPs concerning the retrieval efficiency of import containers at conventional terminals mainly focus on the performance metric of the number of relocations (e.g., Kim and Kim, 1999; Saurí and Martín, 2011; Zhu et al., 2020).

In ACTs, the yard blocks are typically positioned perpendicular to the quay, and containers are transferred between the Automated Stacking Crane (ASC) and trucks at the ends of each block. Under this type of block layout, regardless of the retrieval sequence, to retrieve a container, the ASC needs to travel across a number of bays from the location of the container in the block to the transfer point at the end of the block. There is a trade-off between the ASC travel time and the number of relocations, which both contribute to container retrieval times and truck waiting times. The following works address the container stacking problems at ACTs. For import containers, Yu and Qi (2013) propose three optimization models under the non-segregation and segregation strategies to determine the number of import containers allocated to each bay. The objective function is to minimize the total retrieval time that is the sum of the expected relocation time and the ASC travel time. Yu et al. (2021) propose an integer programming model to allocate import containers to a given yard space so as to minimize the total waiting time of the automated guided vehicle in the stacking process and the external truck waiting time in the future retrieval process. Park et al. (2011) propose an online search algorithm to first select the yard block with the lowest workload and then the yard stack with the minimum weighted sum of four criteria including the stacking cost, the retrieval cost, the relocation cost, and the waste of storage space. For export containers, Zhao et al. (2015) propose a simulation-based optimization method for allocating outbound containers to yard bays aiming to minimize the quay crane waiting time. Gharehgozli and Zaerpour (2018) propose an integer programming model to determine the storage locations of outbound barge containers under a shared stacking strategy with the objective of minimizing the total travel time of the ASC. Preston and Kozan (2001) develop a mixed integer programming model to determine the storage locations of export containers in a multimodal container terminal, in which it is implicitly assumed that the yard block is perpendicular to the quay. The objective is to minimize the total transfer time of each yard machine that is the sum of the travel time and the relocation time. In addition, a couple of studies consider both import and export containers. For example, Dekker et al. (2006) determine the container storage locations by several variants of category stacking rules and use a detailed simulation program to measure the workload of the ASC, the number of containers that cannot be stacked, and the number of relocations. Xia et al. (2016) determine the yard stacks allocated to each container group by a meta-heuristic to maximize the vessel handling efficiency by a weighted sum objective function involving the work balance among blocks and the travel distance between vessels and blocks.

It can be concluded that it is appropriate to use the total retrieval time (i.e. the sum of the crane travel time and the expected relocation time) as the objective function of the SLAP for import containers at ACTs (see e.g., Yu and Qi 2013). Moreover, minimizing the total retrieval time is also a good proxy for truck waiting times (see e.g., Gharehgozli and Zaerpour, 2018). In the next sub-section, we will discuss some studies on estimating the number of relocations.

2.3 Container relocation estimation

Due to the uncertainty in the containers' retrieval sequence, the number of relocations during the retrieval process cannot be easily determined in advance (Bruns et al., 2016). Given an initial stacking configuration, the minimum number of relocations needed to retrieve all the containers depends on the retrieval sequence and the locations of the relocated containers. Such a problem is studied in the

(Stochastic) Container Relocation Problem (c.f. Ku and Arthanari, 2016a, b, Galle et al., 2018a, Feng et al., 2020). Another relevant stream of research is to estimate the number of relocations based on input parameters such as the stack dimensions, the number of containers, the container arrival and departure rates. In this stream, a few studies assume that new arrival containers are added to the stacks during the container retrieval process. For example, Sculli and Hui (1988) are among the first to explore the impacts of the store dimensions, the stacking policies, and the number of different types of containers on the relocation ratio by a simulation model, in which the arrival and departure rates of containers are assumed to be equal and random. De Castilho and Daganzo (1993) derive general formulas, which are functions of the total number stacks and the retrieval rate, to estimate the expected number of moves per container under the segregation and non-segregation stacking policies. On the other hand, some studies only consider retrievals. Kim (1997) are among the first to estimate the number of relocations by assuming that the probability for a container to be picked up next is the same among all the containers. They develop an exact evaluation procedure and a regressive equation to estimate the expected number of relocations for an arbitrary pickup and an approximated formula to estimate the expected total number of relocations. Yu and Qi (2013) improves the accuracy of the approximated formula in Kim (1997) when there are fewer than two tiers of containers. In addition, under the assumption that containers are retrieved batch by batch in a random order, Zhou et al. (2020) derive the number of relocations to retrieve all containers from a yard segment, which is based on the storage space, the number of containers, and the number of container classes, by a discrete event simulation.

All the above studies in the estimation stream estimate the number of relocations based on a prescribed relocation strategy that is used to select the stack for the relocated container. On the other hand, the expected number of blocking containers can also be a good proxy of the number of relocations, which can be derived without assuming any relocation strategy. In this respect, Galle et al. (2016) theoretically prove that the expected minimum number of relocations to retrieve all containers in a bay in a uniformly random order converges to a simple and intuitive lower bound when the number of stacks in a bay is large. The lower bound represents the expected number of blocking containers, which depends on only the number of containers in each stack. This lower bound is used in a recent study on yard crane scheduling by Galle et al. (2018b) to approximate the number of relocations to retrieval all the containers in a single stack when no information is assumed on the retrieval requests. In this paper, we do not assume any information on the retrieval sequence of non-smart containers, that is, they are retrieved in a uniformly random order, and no containers are added to the stacking area during the retrieval process. Such a problem setting is considered in Kim (1997), Yu and Qi (2013), and Galle et al. (2016). Given the advantage of the simplicity and the acceptable accuracy of the lower bound in Galle et al. (2016), we use this lower bound, i.e., the expected number of blocking containers, to approximate the expected number of relocations for retrieving the containers in a non-smart stack (see Section 3.4.2).

2.4 Research gap

There are only two most relevant studies that have investigated grouped-based stacking strategies for import containers, aiming at minimizing the expected number of relocations (Jang et al. 2013) or the total retrieval time (Yu and Qi 2013) during the retrieval process. In Jang et. al (2013), the unit loads (including inbound containers) are classified into multiple groups and the retrieval order is issued for a specific group. Each stack can store multiple groups of containers, and there is no decision on which groups should be relocation-free. In Yu and Qi (2013), import containers are categorized by the arrival times of incoming vessels. The emphasis is to analyze the long-term performance of various segregation and non-segregation strategies models by considering the dynamic of container arrivals and departures. There has been no research on allocating import containers to exact storage slots by utilizing customer information alone while not relying on the container retrieval time. Motivated by the IFF program in practice, our work is the first academic research in this area. We propose a smart stacking strategy that explicitly differentiates smart stacks and non-smart stacks. A smart stack can only store containers from a single customer to guarantee zero relocation, while a non-smart stack can accept containers from multiple customers. Different from the existing stacking strategies, the smart stacking strategy incorporates intelligence into the stacking decision-making by simultaneously optimizing the customers or containers that are allocated to smart stacks and non-smart stacks and the locations of these stacks, by making use of the customer information of the containers. Specifically, we focus on the short-term

operational decisions to allocate a batch of import containers to specific locations in a given storage area of a yard block under the smart stacking strategy, with the objective of minimizing the total retrieval time including the ASC travel time and the relocation time.

3. Problem Description

In this section, we describe the SLAP under consideration in the following aspects: the problem geometry, the problem definitions, two smart stacking policies, and the objective function.

3.1 Problem geometry

At an ACT, a yard block is oriented perpendicular to the quay, as shown in Fig. 1. The configuration of a block consists of M bays, R rows, and K tiers. Bays are indexed by b from landside to seaside, $1 \le b \le M$, rows by r from left to right, $1 \le r \le R$, and tiers by k from bottom to top, $1 \le k \le K$. R is limited by the width of the ASC and K by the height of the ASC, which represents the maximum stacking height. Typically, R ranges from 6 to 13, K from 3 to 6, and M from 40 to 60. A stack is a vertical column located in a bay and a row, which can be characterized by a two-dimensional vector (b, r) representing its location on the ground. A slot is an unit space for storing a Twenty-Foot Equivalent Unit (TEU) container located in a bay, a row and a tier, which can be characterized by a three-dimensional vector (b, r, k).

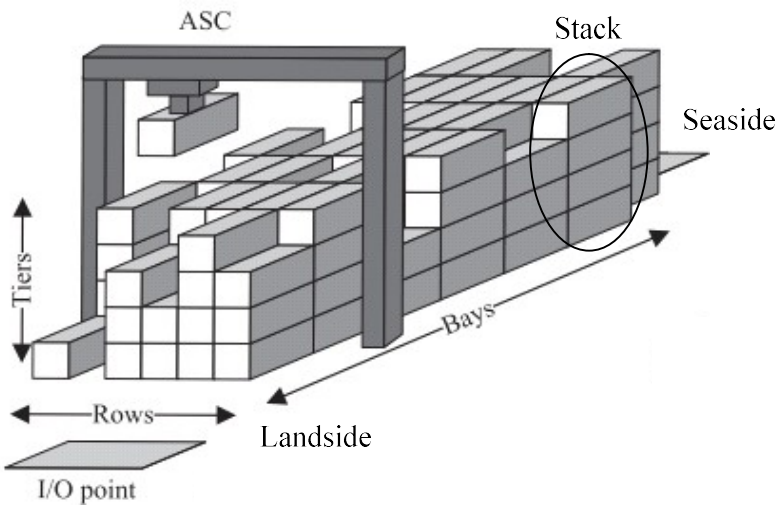


Fig. 1. A stacking area in a yard block with a single ASC. Adapted from Gharehgozli and Zaerpour (2018)

At each end of the block, i.e., landside and seaside, there are several input/output (I/O) points where vehicles park waiting for the service of the ASC. During the import container retrieval process, an external truck with a retrieval request parks at one of the I/O points at the landside. The ASC picks up the required container in the block and then drop off it onto the truck at the I/O point. The location of the I/O point where the truck parks may affect the travel distance of the ASC along the row direction when serving the request. In this paper, we assume that all trucks park at the middle I/O point, i.e., the middle point in front of the first bay, which is considered to be located at bay 0, row $(1+R)/2$ and tier 1. We name this delivery point as the depot, denoted by o. This assumption is in line with the literature (Gharehgozli and Zaerpour, 2018). It is also reasonable statistically because on average a truck can be regarded as being served in the middle I/O point. Moreover, we will show in Section 3.4 that after a certain bay, the travel time of the ASC is determined only by the gantry crane travel time along the bay direction whereas the travel time along the row direction has no impact.

3.2 Problem definitions

We make the offline operational-level decision-making in a single planning period, in which we assign each import container to a slot in a given storage area in a single block. At the beginning of the planning period, we are given a batch of N incoming import containers to be stacked and a storage area where these containers are to be stored. The given storage area is composed of B empty bays (i.e., $B \times R$

empty stacks) in a block. Let Θ denote the set of the B bays, $|\Theta| = B$. Let b, $b \in \Theta$, denote both a bay and its bay index. Note that the B bays do not necessarily need to be located consecutively in the block. A smaller b corresponds to a bay closer to the landside and a greater b corresponds to a bay closer to the seaside. It is worth noting that the models and solution approaches proposed in this paper can be easily extended to a non-empty storage area where some stacks have been occupied by existing containers.

As the yard is a scarce resource at container terminals (de Melo da Silva et al., 2018), terminal operators tend to make high utilization of the yard storage space. Therefore, the number of available bays B should be limited depending on the yard utilization rate. Usually, terminal operators would set an average utilization rate for the yard space. Let u denote the utilization rate of a bay, which is defined as the percentage of all its slots being occupied by the containers stored in the bay. In our experiments, given N, R and u, B is given by $B = \lceil N / \lfloor R \cdot K \cdot u \rfloor \rceil$, which represents the required number of empty bays to store N containers when each bay is utilized at its pre-set utilization rate u.

The batch of N containers to be stacked belong to C customers. The containers are grouped by customers. Let $c \in \{1, \ldots, C\}$ denote the index of customers or groups, and the number of containers in a group c is called group size, denoted by v_c. When all the containers in a stack are in the same group and the stack is not allowed to be used for relocation, no relocations are needed when retrieving the containers in this stack as containers are 'peeled off' from the top of the stack. We refer to such containers as *smart containers* (i.e., free-flow containers) and such stacks as *smart stacks*. In another situation, when the containers in a stack are from more than one group or all the containers in a stack are in the same group but the stack can be used for relocation, relocations may be needed for retrieving the containers in this stack. We refer to such containers as *non-smart containers* (i.e., non-free-flow containers) and such stacks as *non-smart stacks*. Accordingly, we refer to a bay in which all the stacks are smart as a *smart bay*, in which all the stacks are non-smart as a *non-smart bay*, and in which both smart and non-smart stacks exist as a *mixed bay*.

A smart stack occupies the entire K slots of the stack regardless of how many containers are allocated to it. This is because a smart stack is dedicated to a single group and is not allowed to be used for relocation. If a bay is a smart bay, the capacity of the bay is equal to RK; otherwise, its capacity is equal to RK-(K-1) because at least (K-1) empty slots need to be reserved for relocation in order to avoid deadlock (Tang et al., 2015; Chang and Zhu, 2019). To ensure the B bays are sufficient to store all the containers, the bay utilization rate u must satisfy the condition $\lfloor R \cdot K \cdot u \rfloor \le R \cdot K - (K-1)$.

The following assumptions are made for formulating the problem.

A1. Each container is associated with a customer. The containers belonging to the same customer form a group. If the customer information of a container is unknown, this container forms a group on its own.

A2. The containers in a smart stack are retrieved from the top to the bottom without requiring any relocation.

A3. The containers in a non-smart stack are retrieved in uniformly random order and relocations are needed.

A4. The container weight is not considered by assuming it will not influence its stacking location and all import containers to be stacked are of standard 20 ft.

A4 can be explained as follows. The weight attribute of a container is less important for the import container stacking than the export container stacking, because the retrieval sequence of import containers is not constrained by their weights. Although the weight of a container may raise the safety issue in stacking (e.g., Razouk et al., 2016; Güven and Eliiyi, 2019), the majority of studies on import containers ignore the weight constraints (e.g., Dekker et al., 2006; Guldogan, 2011; Wang et al., 2020; Zhu et al., 2020). We, therefore, make this assumption by focusing on the smart stacking strategy for import containers. This will help us better demonstrate the value of customer information.

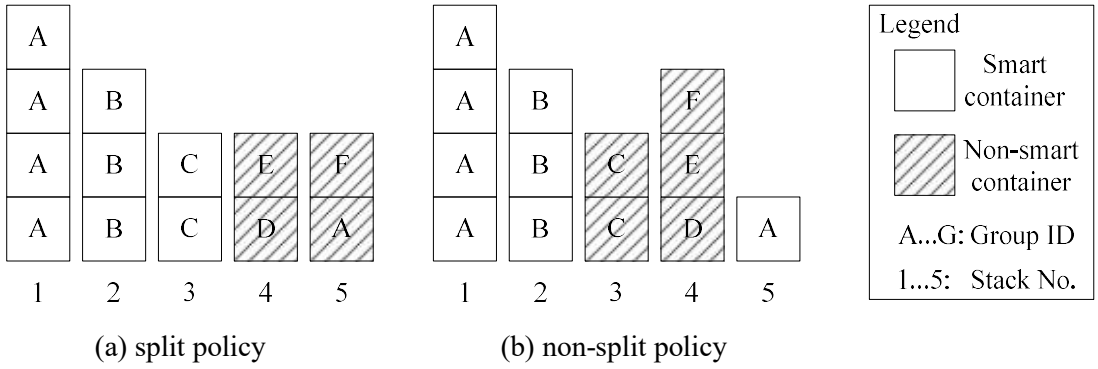
3.3 Two smart stacking policies

Since a customer may have multiple containers and the capacity of a stack is restricted by the maximum stacking height K, more than one stacks may be needed to store the containers from a single customer. Two variants of the smart stacking strategy could be designed, depending on whether all or

part of the containers of a customer are smart. Accordingly, we propose two smart policies, *non-split policy* and *split policy*.

Under the non-split policy, a group of containers is not allowed to be split between smart and non-smart stacks. In other words, they are either wholly allocated to smart stacks or wholly allocated to non-smart stacks; and we are concerned with which groups/customers should be smart. Under the split policy, a group of containers can be split between smart stacks and non-smart stacks; and we determine how many containers from a group should be smart. Under either policy, we need to determine where these smart stacks and non-smart stacks should be located in the block.

Fig. 2 provides an example solution under the two policies respectively for a single bay to illustrate the influence of splitting on the number of relocations. Under both policies, the maximum stacking height K=4 forces the groups that have more than K containers to be divided and allocated to different stacks (e.g., group A). In Fig. 2(a), under the split policy, the five containers of group A are split between smart stack 1 and non-smart stack 5 where one container of group A is mixed with one container from group F. In Fig. 2(b), under the non-split policy, to make all the containers of group A smart, which are allocated to smart stacks 1 and 5, stacks 3 and 4 are determined to be non-smart stacks. Note that although the two containers in stack 3 are from the same group C, they are non-smart containers and stack 3 is a non-smart stack because stack 3 has to be used for accommodating relocations from stack 4 in order to avoid deadlock. Besides, one more container is added to stack 4 compared to that under the split policy, which increases the possibilities of relocation for stack 4. It can be seen that the split policy can save more relocations than the non-split policy.



(a) split policy (b) non-split policy

Fig. 2. Illustration of the solutions under two smart policies for a single bay

In this paper, we optimize and evaluate the effects of smart stacking under both two policies. We focus more on analyzing the non-split policy because it is closer to the current practice of container terminals and easier to implement from the perspective of customer administration. In Section 4, we develop mathematical models under each policy, and then we compare the two models both theoretically (section 4.2) and computationally (section 6.5).

3.4 Objective function

The objective of our SLAP is to minimize the total retrieval time that is the sum of the ASC's travel time and relocation time. In the following two sub-sections, we present how the two components in the objective functions are measured.

3.4.1 ASC's travel time

We first introduce the ASC's working pattern, and then we give the mathematical expression of the travel time. In the end, we present the properties of the ASC's travel time.

The container retrieval operations are performed by a single ASC at the landside of the block. To serve a retrieval request, the ASC needs to perform both horizontal travel and vertical travel activities. During horizontal travels, the ASC moves its gantry along bays and its trolley along rows simultaneously in Chebyshev distance. During vertical travels, the ASC moves its spreader up and down. These travel activities can be divided into four phases as shown in Fig. 3. Firstly, the crane performs an empty drive from its current position to the target stack where the requested container is stored (horizontal). Secondly, the crane lowers its spreader to pick up the container and then hoists the spreader up (vertical). Thirdly,

the crane performs a loaded drive from the target stack to depot o (horizontal). Finally, the crane lowers its spreader to set down the container on the truck at the depot and then hoists the spreader up (vertical).

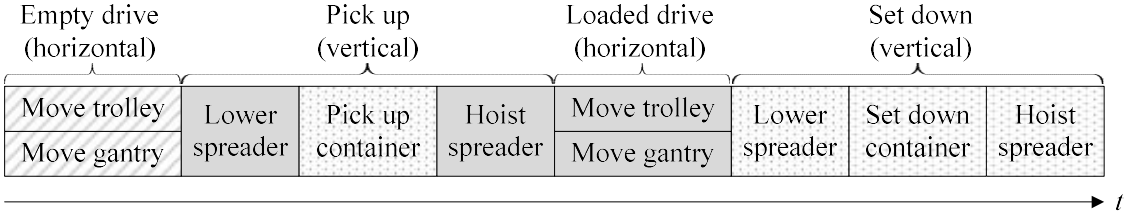


Fig. 3. Pattern of typical yard crane movements for a retrieval cycle. Solid grey box indicates variable parts, dotted grey box indicates constant parts, and striped grey box indicates irrelevant parts. (Adapted from Galle et al. (2018b))

Based on this pattern, we now analyze the travel time spent in each phase and derive the relevant part that will be included in the optimization model.

(1) Empty drive. The empty drive time depends on which stack the ASC currently stops when a retrieval request arrives, that is, where the ASC ended its previous request. This is usually optimized in the yard crane schedule problem (see e.g. Galle et al., 2018b) when multiple types (stacking and retrieval) of requests are considered and their stacking positions and service sequences are to be determined simultaneously. In this paper, since we only focus on the retrieval requests, the empty travel time is not relevant and thus is not included in the objective function.

(2) Loaded drive. The loaded drive time depends on the stack (b, r) where the requested container is stored, which is a variable part. Let T_b be the gantry moving time from bay b to bay zero (i.e., the artificial bay in front of bay 1), and T_r be the trolley moving time from row r to the middle I/O point, which is located at the middle of bay zero. As the ASC moves along bays and rows simultaneously in Chebyshev distance, the ASC's loaded drive time is calculated as $T_{br} = \max\{T_b, T_r\}$. If $T_b > T_r$, the travel time along the bay T_b is dominant ($T_{br} = T_b$); otherwise, the travel time along the row T_r is dominant ($T_{br} = T_r$). We call a bay b with $T_{br} = T_b$ for any $1 \le r \le R$ a *dominant bay*; otherwise, it is called a *non-dominant bay*. We call a stack (b, r) with $T_{br} = T_r$ a *dominant stack*. It should be noted that if we assume that external trucks will be parked in positions [1, R] randomly with a uniform distribution, then the ASC's loaded drive time (i.e., the horizontal travel time along the bays and the rows) on average will have a different expression. However, this does not affect the arguments of our study. In addition, Appendix C compares the ASC's horizontal travel time between the assumptions of middle I/O point and random I/O point, which demonstrates the reasonability of assuming the middle I/O point.

(3) Pick up. The time spent in this phase consists of three parts: the spreader's lowering time (without a container), the time to handle and stabilize the container, and the spreader's hoisting time (with a container). Among them, the time to handle and stabilize the container is constant (e.g., 20 seconds), and thus it is not included in the objective function. The times of the other two parts depend on the tier k where the requested container is stored, which is a variable part. Let T_k^E denote the time spent in lowering the spreader from the crane height (i.e., tier K) to tier k, and T_k^L the time spent in hoisting the spreader from tier k to tier K. Then, the variable part for the pick-up phase is expressed by $T_k = T_k^E + T_k^L$.

(4) Set down. Containers are dropped off onto trucks at the depot. Since the locations of the trucks are fixed, the set-down time is constant and thus it is not included in the objective function.

To sum up, the variable part of the ASC's travel time to retrieve a container stored at slot (b, r, k) is expressed by $T_{brk} = \max\{T_b, T_r\} + T_k^E + T_k^L$. This variable part T_{brk} will be included in our objective function. Namely, only the relevant travel time of the ASC when retrieving a container is considered in the optimization model.

In the following, we present the structural properties of the ASC's travel time in Property 1. The properties will be utilized in the rest of the paper, including designing the heuristic algorithm and analyzing the results of the experiments. Property 1 states that (i) for the stacks in the same row, the ASC's horizontal travel time increase with the bay index; (ii) after a certain bay $\hat{b}$, the ASC's horizontal travel time depends only on the bay index regardless of the row index, and for the stacks in any rows after bay $\hat{b}$, the ASC's horizontal travel time increases with the bay index. According to our definition, all the bays b that satisfy $b \ge \hat{b}$ belong to dominant bays, and we say bay $\hat{b}$ is the

minimum dominant bay. In our experiment in which a block has twelve rows, $\hat{b} = 3$, indicating that the ASC's horizontal travel for retrieving a container located in a bay b, $b \geq 3$, is determined only by the bay index. The point (iii) in Property 1 indicates that the ASC's vertical travel time decreases with the tier where the container is stored. The proof of Property 1 is provided in Appendix A.

Property 1. Let l^x and l^y be the length and width of a 20-ft (twenty-foot equivalent unit) standard container, respectively. Let v^x be the ASC gantry moving speed with load and v^y be the ASC trolley moving speed with load. For a block with R rows and M bays: (i) for any r ($1 \leq r \leq R$), $T_{br} \geq T_{b'r}$ for $1 \leq b' < b \leq M$; (ii) if there exists a bay $\hat{b}$ ($1 \leq \hat{b} \leq M$) that satisfies $\frac{\hat{b} \cdot l^x}{v^x} \geq \frac{(R-1)/2 \cdot l^y}{v^y}$ and $\frac{(\hat{b}-1) \cdot l^x}{v^x} < \frac{(R-1)/2 \cdot l^y}{v^y}$, then $T_{br} = T_b$ for $1 \leq \hat{b} \leq b \leq M$ and $1 \leq r \leq R$, and $T_{br} > T_{b'r'}$ for $\hat{b} \leq b' < b \leq M$, $1 \leq r \leq R$ and $1 \leq r' \leq R$; (iii) $T_k < T_{k'}$ for $1 \leq k' < k \leq K$.

3.4.2 Relocation time

Relocation moves are often inevitable when retrieving the containers from non-smart stacks since containers in these stacks are requested by external trucks in random order. As in the literature (Zhao and Goodchild, 2010; Yu and Qi, 2013), we assume that the number of relocations and the time needed to perform one relocation is independent, and thus the total relocation time is estimated by the expected number of relocations and the average time to relocate one container.

Since the exact number of relocations cannot be easily determined in advance, this part of the objective function is often replaced by a lower bound on the number of relocations (Bruns et al., 2016). When no information is available on the container retrieval sequence, the lower bound of the expected number of blocking containers is regarded as a good proxy for the expected number of relocations when the number of stacks in a bay is large (Galle et al., 2016). We use the lower bound in Galle et al. (2016) to approximate the expected number of relocations given that it depends only on the number of containers in a stack which can make the model tractable. In our preliminary study, we have also examined the alternative method from Kim (1997) and Yu and Qi (2013) to estimate the number of relocations. The numerical results show that the new model using the alternative estimation method produce quite similar solutions, especially in terms of the percentage of the smart containers and the total ASC travel time, but the new model is more computationally expensive because it involves more variables and constraints.

We define α_k to be the expected number of blocking containers in a stack of k containers in the case when no information on the containers' retrieval sequence is available. From Galle et al. (2016), we have $\alpha_0 = 0$ and

$$\alpha_k = k - \sum_{i=1}^{k} \frac{1}{i}, \ \forall k \in \{1, \ldots, K\} \tag{1}$$

The expected number of relocations for a non-smart stack of k containers is calculated by Eq. (1). Let $\overline{T}$ be the average time of relocating a container. Then, the total expected relocation time for a non-smart stack of k containers can be expressed by the product of α_k and $\overline{T}$.

4. Mathematical models

In this section, we present the mathematical models under the non-split policy and the split policy respectively. The notations used in both models as given as follows. The unique notations used in each model will be introduced when introducing the corresponding models.

Parameters:

N: the total number of containers to be stacked.

Θ : the set of empty bays, $|\Theta| = B$.

R: the number of rows (stacks) in a bay.

K: the maximum stacking height.

$\overline{T}$: the average time needed to perform a relocation (in seconds).

α_k : the expected number of relocations in a non-smart stack of k containers, which is defined by Eq. (1).

T_{brk} : the ASC's travel time to retrieve a container located at slot (b, r, k) (in seconds).

Auxiliary Variables:

z_b : equals zero if all the stacks in bay b are smart stacks, and one otherwise;

h_{br} : the number of non-smart containers in stack (b, r);

w_b : the expected number of relocations in bay b, which is a continuous variable;

f_{brk} : equals one if there are k non-smart containers stored in stack (b, r), and zero otherwise;

y_{brk} : equals one if there is a container stored at tier k of stack (b, r), and zero otherwise.

4.1 Non-split model

We now present the mathematical model under the non-split policy, that is, a group of containers is either wholly allocated to smart stacks or wholly allocated to non-smart stacks. We first develop an original formulation, and then we develop an improved formulation by enhancing the variable representation of the original one. Both formulations are mixed-integer programming (MIP) models. The key decision variables in the non-split model include which groups (customers) should be selected as *smart groups*, how the containers from a smart group should be distributed to multiple smart stacks, and where these smart stacks should be located in the block. Note that all the containers in a smart group should be allocated to some smart stacks.

4.1.1 Original formulation

The newly defined parameters and decision variables in the original formulation are as follows.

Parameters:

v_c : the size of group c.

Decision Variables:

x_{br}^{ck} : equals one if stack (b, r) is a smart stack that is allocated to k containers of group c, and zero otherwise;

s_c : equals one if group c is a smart group, and zero otherwise;

The original formulation (denoted by M1) is presented below:

[M1]:

$$\min \sum_{b\in\Theta}\sum_{r=1}^{R}\sum_{k=1}^{K} T_{brk}\cdot y_{brk} + \sum_{b\in\Theta} w_b \cdot \overline{T} \tag{2}$$

$$\text{s.t.} \quad \sum_{c=1}^{C}\sum_{k=1}^{K} x_{br}^{ck} \le 1, \ \forall b\in\Theta, \ \forall r \in \{1,\dots,R\} \tag{3}$$

$$\sum_{b\in\Theta}\sum_{r=1}^{R}\sum_{k=1}^{K} x_{br}^{ck}\cdot k = s_c \cdot v_c, \ \forall c \in \{1,\dots,C\} \tag{4}$$

$$\sum_{b\in\Theta}\sum_{r=1}^{R}\left(\sum_{c=1}^{C}\sum_{k=1}^{K} x_{br}^{ck}\cdot k + h_{br}\right) = N \tag{5}$$

$$z_b \le R - \sum_{r=1}^{R}\sum_{c=1}^{C}\sum_{k=1}^{K} x_{br}^{ck}, \ \forall b\in\Theta \tag{6}$$

$$z_b \cdot R \ge R - \sum_{r=1}^{R}\sum_{c=1}^{C}\sum_{k=1}^{K} x_{br}^{ck}, \ \forall b\in\Theta \tag{7}$$

$$\sum_{r=1}^{R}\left(\sum_{c=1}^{C}\sum_{k=1}^{K} x_{br}^{ck}\cdot K + h_{br}\right) \le RK - (K-1)\cdot z_b, \ \forall b\in\Theta \tag{8}$$

$$h_{br} \le \left(1 - \sum_{c=1}^{C}\sum_{k=1}^{K} x_{br}^{ck}\right)\cdot K, \ \forall b\in\Theta, \ \forall r \in \{1,\dots,R\} \tag{9}$$

$$\sum_{k=1}^{K} y_{brk} = \sum_{c=1}^{C}\sum_{k=1}^{K} x_{br}^{ck}\cdot k + h_{br}, \ \forall b\in\Theta, \ \forall r \in \{1,\dots,R\} \tag{10}$$

$$y_{brk} \le y_{br,k-1}, \ \forall b\in\Theta, \ \forall r \in \{1,\dots,R\}, \ \forall k\in\{2,\dots,K\} \tag{11}$$

$$w_b = \sum_{r=1}^{R}\sum_{k=1}^{K} \alpha_k \cdot f_{brk}, \ \forall b\in\Theta \tag{12}$$

$$\sum_{k=1}^{K} f_{brk} \le 1, \ \forall b\in\Theta, \ \forall r \in \{1,\dots,R\} \tag{13}$$

$$\sum_{k=1}^{K} k \cdot f_{brk} = h_{br}, \ \forall b \in \Theta, \ \forall r \in \{1,\ldots,R\} \tag{14}$$

$$y_{brk} \in \{0,1\}, \ f_{brk} \in \{0,1\}, \ \forall b \in \Theta, \forall r \in \{1,\ldots,R\}, \ \forall k \in \{1,\ldots,K\} \tag{15}$$

$$z_b \in \{0,1\}, \ w_b \geq 0, \ \forall b \in \Theta \tag{16}$$

$$h_{br} \in Z+, \ \forall b \in \Theta, \forall r \in \{1,\ldots,R\} \tag{17}$$

$$s_c \in \{0,1\}, \ \forall c \in \{1,\ldots,C\} \tag{18}$$

$$x_{br}^{ck} \in \{0,1\}, \ \forall b \in \Theta, \forall r \in \{1,\ldots,R\}, \forall c \in \{1,\ldots,C\}, \ \forall k \in \{1,\ldots,K\} \tag{19}$$

The objective function (2) is to minimize the total retrieval time, which is the sum of the total ASC travel time and the total relocation time. Constraints (3) ensure that each smart stack is allocated to a specific number of containers of at most one group. Constraints (4) guarantee the non-split policy, that is, for a smart group, all the containers in the group are allocated to smart stacks. Constraints (5) ensure that all given containers are stored in the given storage area. Constraints (6) and (7) define the auxiliary decision variables z_b that indicate if all the stacks in a bay are smart. z_b is forced to equal zero if all the stacks in bay *b* are smart and equal one if there are non-smart stacks in bay *b*. Constraints (8) guarantee the capacity feasibility of each bay. For a smart bay (i.e, $z_b = 0$), the capacity of the bay is equal to *RK*; otherwise, its capacity is equal to *RK*-(*K*-1) because at least (*K*-1) empty slots need to be reserved for relocation in order to avoid deadlock. Constraints (9) guarantee that the height of a non-smart stack is not more than *K*. Constraints (9) also ensure that there is no non-smart container in a smart stack. If stack (*b*, *r*) is a smart stack, i.e., $\sum_{c=1}^{C}\sum_{k=1}^{K} x_{br}^{ck} = 1$, h_{br} is forced to equal zero. Constraints (10) and (11) determine the height of each stack and guarantee that containers are stacked from the ground and are stacked on top of one another. Constraints (12) calculate the total relocation time for retrieving the containers in a bay. Constraints (13) and (14) define the auxiliary decision variables f_{brk} by h_{br}. If $h_{br} = 0$, $f_{brk} = 0$, $\forall k \in \{1,\ldots,K\}$; otherwise, $f_{br,h_{br}} = 1$ and $f_{brk} = 0$, $\forall k \in \{1,\ldots,K\} / h_{br}$. Finally, Constraints (15)-(19) specify the domains for the decision variables.

4.1.2 Properties of optimal solutions

We now propose two propositions regarding the properties of optimal solutions of the original formulation, based on which we will develop an improved formulation in the next sub-section.

First, we define a *pile* as the set of containers stacked in the same stack. We refer to a pile that is stacked to the maximum stacking height as a *full pile*. A smart pile is implied by the smart stack.

Proposition 1. Let h_{p_i} be the height of pile p_i. In the optimal solution to M1, for any two smart piles p_i and p_j which are located at stacks (b_i, r_i) and (b_j, r_j) respectively, if $h_{p_i} > h_{p_j}$, then $T_{b_i r_i} \leq T_{b_j r_j}$.

Sketch of the proof. The proof is proved in Appendix A. We suppose by contradiction that $h_{p_i} > h_{p_j}$ and $T_{b_i r_i} > T_{b_j r_j}$ in the optimal solution σ^*. We construct a feasible solution σ' such that $h_{p_i} > h_{p_j}$ and $T_{b_i r_i} < T_{b_j r_j}$, and we show that σ' leads to a smaller objective value than σ^* which provides a contradiction to σ^* being an optimal solution.

Proposition 2. Let P_c be the number of piles of a smart group *c* in the optimal solution, then $P_c = \lceil v_c / K \rceil$, and the P_c piles are composed of P_c -1 full piles and one pile whose height equals $v_c - (P_c - 1) \cdot K$.

Sketch of the proof. The proof is provided in Appendix A. Let us rank the P_c piles in ascending order of the ASC's horizontal travel time needed to retrieve a container from the stack where the pile is located. According to Proposition 1, the P_c piles can be ordered as $p_1, p_2, \ldots, p_{P_c}$ with $h_{p_1} \geq h_{p_2} \geq \ldots \geq h_{p_{P_c}}$ and $T_{b_1 r_1} \leq T_{b_2 r_2} \leq \ldots \leq T_{b_{P_c} r_{P_c}}$. There are two cases depending on the size of v_c, i.e., $v_c \geq K$ and $v_c < K$. In the case of $v_c \geq K$, it is sufficient to suppose by contradiction that in the optimal solution there is at least

one non-full pile in the first P_c -1 piles. Let p_i be the highest non-full pile in the first P_c -1 piles. We can construct a feasible solution such that the height of p_i is increased by one by moving one container from pile p_{P_c} to pile p_i. In the case of $v_c < K$, it is sufficient to suppose by contradiction that in the optimal solution $h_{p_1} < v_c$. We can construct a feasible solution by moving one container from pile p_{P_c} to pile p_1. In both cases, the feasible solution we construct can lead to a lower objective value than the optimal solution, which provides a contradiction to the supposing.

4.1.3 Improved formulation

Based on the propositions of the optimal solution in Section 4.1.2, we can propose an improved formulation, which is easier to solve than [M1].

To simplify the narrative, we introduce the concept of optimal partition. The idea of the new formulation is to make decisions on which customers and partitions should be smart without the need of associating the smart partitions with specific smart customers. Proposition 2 indicates that in the optimal solution of the original formulation, each group is partitioned into $P_c = \lceil v_c / K \rceil$ piles such that P_c -1 piles have K containers and one pile has $v_c - (P_c - 1) \cdot K$ containers. We define the *optimal partition* of a group as the pattern that divides the group into the partitions each corresponding to a pile of the group in the optimal solution. Let $\pi_c = \{p_i \mid i \in [1,...,P_c]\}$ denote the set of partitions of group c by using the optimal partition, where $P_c = \lceil v_c / K \rceil$ is the number of partitions of group c, p_i is the number of containers in partition i. Without the loss of generality, we let $p_i = K$, $i \in \{1,\ldots,\lceil v_c / K \rceil - 1\}$, and $p_{\lceil v_c/K \rceil} = v_c - (P_c - 1) \cdot K$.

We partition each group by using the optimal partition. With such pre-processing, the improved formulation is developed based on the unit of partition. The newly defined parameters and decision variables in the improved formulation are as follows.

Parameters:

u_{ck} : the number of partitions with k containers for group c, $c \in \{1,\ldots,C\}$, $k \in \{1,...,K\}$, which is defined by

$$u_{ck} = \begin{cases} P_c - 1, k = K \\ 1, k = v_c - (P_c - 1) \cdot K \\ 0, k = \{1,...,K\} / \{K, v_c - (P_c - 1) \cdot K\} \end{cases}$$

Decision Variables:

x_{br}^k : equals one if stack (b, r) is a smart stack that is allocated to a partition with k containers, $k \in \{1,...,K\}$, and zero otherwise.

The improved formulation (denoted by M2) is presented below:

[M2]:

$$\min \sum_{b \in \Theta} \sum_{r=1}^{R} \sum_{k=1}^{K} T_{brk} \cdot y_{brk} + \sum_{b \in \Theta} w_b \cdot \bar{T} \tag{2}$$

$$\text{s.t.} \quad \sum_{k=1}^{K} x_{br}^k \le 1, \ \forall b \in \Theta, \ \forall r \in \{1,\ldots,R\} \tag{20}$$

$$\sum_{b \in \Theta} \sum_{r=1}^{R} x_{br}^k = \sum_{c=1}^{C} s_c u_{ck}, \ \forall k \in \{1,...,K\} \tag{21}$$

$$\sum_{b \in \Theta} \sum_{r=1}^{R} \left(\sum_{k=1}^{K} x_{br}^k \cdot k + h_{br} \right) = N \tag{22}$$

$$z_b \le R - \sum_{r=1}^{R} \sum_{k=1}^{K} x_{br}^k, \ \forall b \in \Theta \tag{23}$$

$$z_b \cdot R \ge R - \sum_{r=1}^{R} \sum_{k=1}^{K} x_{br}^k, \ \forall b \in \Theta \tag{24}$$

$$\sum_{r=1}^{R} \left(\sum_{k=1}^{K} x_{br}^k \cdot K + h_{br} \right) \le RK - (K-1) \cdot z_b, \ \forall b \in \Theta \tag{25}$$

$$h_{br} \le \left(1 - \sum_{k=1}^{K} x_{br}^k \right) \cdot K, \ \forall b \in \Theta, \ \forall r \in \{1,\ldots,R\} \tag{26}$$

$$\sum_{k=1}^{K} y_{brk} = \sum_{k=1}^{K} x_{br}^{k} \cdot k + h_{br}\ ,\ \forall b \in \Theta\ ,\ \forall r \in \{1,\ldots,R\} \tag{27}$$

$$x_{br}^{k} \in \{0,1\}\ ,\ \forall b \in \Theta\ ,\ \forall r \in \{1,\ldots,R\}\ ,\ \forall k \in \{1,\ldots,K\} \tag{28}$$

Constraints (11) – (18)

The objective function (2) is the same as in [M1]. Constraints (20) ensure that each smart stack is allocated to a partition with a specific number of containers. Constraints (21) guarantee the non-split policy by forcing the total number of partitions with *k* containers for all the smart groups equals the total number of smart stacks each storing *k* containers. Constraints (22) ensure that all given containers are stored in the given storage area. Constraints (23) and (24) define the auxiliary decision variables z_b that indicate if all the stacks in a bay are smart. Constraints (25) guarantee the capacity feasibility of each bay. Constraints (26) guarantee that the height of a non-smart stack is not more than *K* and there is no non-smart container in a smart stack. Constraints (27) determine the height of each stack. Constraints (11) – (18) inherit from [M1].

Proposition 3 states that the optimization problem defined by [M1] and the optimization problem defined by [M2] are equivalent problems.

Proposition 3. [M1] and [M2] are equivalent problems.

Sketch of the proof. The proof is provided in Appendix A and is in three parts. In order to prove the equivalence between [M1] and [M2], we first show there is an implied constraint for [M1] that can be derived by the definition of u_{ck}. By using this implied constraint, we can reformulate [M1] into an equivalent counterpart [M1-1]. Secondly, by using the transformation between x_{br}^{ck} and x_{br}^{k}, we can reformulate [M1-1] into an equivalent counterpart [M1-2]. Lastly, we show that [M1-2] and [M2] are equivalent. □

4.2 Split model

In this section, we develop the mathematical model under the split policy, that is, a group of containers can be split between smart stacks and non-smart stacks. Intuitively, selecting a full pile to be a smart stack is more beneficial than selecting a non-full pile to be a smart stack because of more reduction on relocations and better utilization of the stack space. The split policy offers the opportunities to increase the number of full smart piles and decrease the number of non-full smart piles. We restrict the split policy to the cases that all smart piles must be selected from the optimal partitions defined in Section 4.1.3. This treatment makes the split model easy to compare with the non-split model.

We pre-process each group by the optimal partition as that in Section 4.1.3, and as a result, we get a total of $\sum_{c=1}^{C}\sum_{k=1}^{K} u_{ck}$ partitions for all the containers. We define a new parameter $n_k = \sum_{c=1}^{C} u_{ck}$ that denotes the total number of partitions with *k* containers, $k \in \{1,\ldots,K\}$, in which u_{ck} is defined in Section 4.1.3. The decisions of the split model focus on which partitions should be smart. The newly defined parameters and decision variables in the split model are as follows. Note that x_{br}^{k} has been defined for [M2], we introduce it here again for the purpose of differentiating it from the decision variable x_{br}^{ck} in [M1].

Parameters:

n_k : the total number of partitions with *k* containers, $k \in \{1,\ldots,K\}$, which is defined by $n_k = \sum_{c=1}^{C} u_{ck}$, $k \in \{1,\ldots,K\}$.

Decision variables:

x_{br}^{k} : equals one if stack (*b*, *r*) is a smart stack that is allocated to a partition with *k* containers, $k \in \{1,\ldots,K\}$, and zero otherwise.

The split model (denoted by M3) is presented below:

[M3]:

$$\min \sum_{b\in\Theta}\sum_{r=1}^{R}\sum_{k=1}^{K} T_{brk} \cdot y_{brk} + \sum_{b\in\Theta} w_b \cdot \overline{T} \tag{2}$$

$$\text{s.t.}\ \sum_{b\in\Theta}\sum_{r=1}^{R} x_{br}^{k} \le n_k\ ,\ \forall k \in \{1,\ldots,K\} \tag{29}$$

Constraints (11) – (18), (20), and (22) – (28)

Constraints (29) ensure that the number of smart stacks allocated to the partitions with k containers is not more than the total number of partitions with k containers. The other constraints inherit from [M2].

The following two lemmas compare the split model and the non-split model theoretically. Lemma 1 states that the objective value of [M3] is not greater than that of [M2]. Lemma 2 states that [M2] and [M3] have the same objective value when the maximum group size is no more than the maximum stacking height K. The proofs of Lemma 1 and Lemma 2 are provided in Appendix A.

Lemma 1. Let $f(\mathrm{M2})$ and $f(\mathrm{M3})$ be the objective functions of the optimization problems defined by [M2] and [M3], respectively, then $f(\mathrm{M3}) \le f(\mathrm{M2})$.

Lemma 2. Let $V^m = \max_{c\in\{1,\dots,C\}}\{v_c\}$, then $f(\mathrm{M2}) = f(\mathrm{M3})$ when $V^m \le K$.

4.3 Complexity of the SLAP

The SLAP is NP-hard in general. This can be proved by reducing the Set Partitioning Problem to a special instance of the SLAP. We define a 'bay profile' as a feasible assignment of containers to a bay, and each bay profile is associated with a total time for retrieving all the containers in this bay profile. For all available bays, a finite set of all feasible bay profiles can be defined in advance, denoted by P. Consider a special case of the SLAP, in which we are given only a subset of P, denoted by P'. Then, the remaining task of the SLAP is to find a subset of bay profiles from P' with the minimum total retrieval time subject to that it covers all of the containers and no two of the bay profiles share the same container. This instance is equivalent to the Set Partitioning Problem. Therefore, we have reduced the Set Partitioning Problem to a special case of the SLAP. Because the Set Partitioning Problem is known to be NP-hard (Rasmussen and Larsen, 2011), the SLAP is NP-hard.

Due to the NP-hardness of the SLAP, the proposed MIPs are computationally expensive to solve for large-scale problems, which is not realistic for real-world decision-making (c.f. the results in Section 6.3). Therefore, in the next section, we will develop an efficient heuristic algorithm that is able to find near-optimal solutions within several seconds for the SLAPs of practical scales.

5. Divide-and-conquer heuristic

In this section, we develop a heuristic algorithm for the non-split variant by employing the divide-and-conquer strategy. The framework of the heuristic is introduced in Section 5.1 and the details are presented in Sections 5.2-5.5.

5.1 Framework

Divide-and-conquer is an important paradigm to design computationally efficient algorithms in computer science (Li et al., 2009). The principle underlying divide and conquer algorithms is that: the original problem is decomposed into two or more subproblems until they become sufficiently simple to be solved directly; the subproblems are solved independently and their solutions are composed to give a solution to the original problem (Smith et al., 1985). The divide-and-conquer paradigm has become a commonly used strategy to design efficient algorithms for complicated combinatorial optimization problems (e.g., Reimann et al., 2004; Jin et al., 2016; Wei et al., 2019).

The structure of the non-split model [M2] motivates us to employ the divide-and-conquer strategy to decompose the original problem into several subproblems and solve them sequentially and iteratively. It is observed that in [M2], the decision vector $\boldsymbol{x}$ about the smart stacks and the decision vector $\boldsymbol{h}$ about the non-smart stacks are coupled only by Constraints (22) and (25) - (27). By relaxing these constraints, the original problem can be decomposed into two smaller subproblems: one involving only the smart stack decisions and the other only concerning the non-smart decisions, which are much easier to solve. The solution to the original problem can be obtained by solving a third subproblem in which the solutions to the first two subproblems are combined.

Fig. 4 provides the framework of the divide-and-conquer (D&C) heuristic. At each iteration, three subproblems are solved sequentially based on the updated number of available bays. Initially, all the given B empty bays are taken into account; then at each later iteration, the number of available bays is reduced by one. Subproblem 1 is first solved to determine the smart piles. Here, we use "smart piles" rather than "smart stacks" because subproblem 1 only concerns the heights of smart stacks but does not determine their locations; instead, the smart piles are considered to be temporarily stacked in a certain

area in the block to guarantee the feasibility of the capacity constraint. After that, the number of non-smart containers and the remaining storage space for non-smart containers can be obtained and passed to subproblem 2 where the non-smart piles are determined subsequently. Finally, the smart piles and the non-smart piles resulted from subproblems 1 and 2 are passed to subproblem 3, in which the locations of each pile are determined and the objective function is updated. The algorithm terminates at the iteration with $B^{\min}$ bays, which is the minimum number of required bays supposing each bay can be fully utilized, or the iteration where the objective function stops improving.

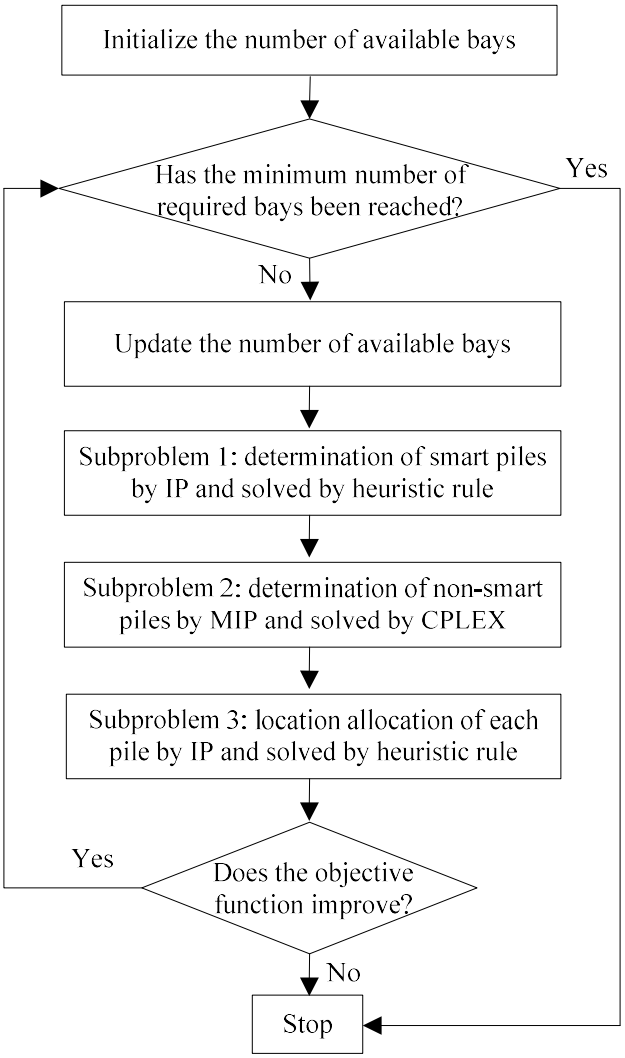


Fig. 4 The framework of the divide-and-conquer algorithm

5.2 Updating scheme and stopping criteria

Let Θ_i denote the set of available bays at the ith iteration, and B_i the size of Θ_i. Θ_0 is initialized as Θ and B_0 is initialized as B. After solving the three subproblems at each iteration, Θ_{i+1} is updated by removing the bay closest to the seaside in Θ_i, and accordingly, the number of available bays is updated by $B_{i+1}=B_i-1$. The idea behind this updating scheme is to minimize the objective function through the trade-off mechanism between the ASC travel time and the relocation time. When the available storage space gets larger, the number of relocations tends to decrease but the retrieval time tends to increase. The objective function is minimized at a certain point where the number of occupied stacks is optimal. Therefore, by scanning the number of available stacks, we are able to minimize the objective function. However, instead of scanning the number of stacks, we can reduce the number of iterations and thus save the solution time by scanning the number of bays. Although this does not guarantee optimality, it is expected that the optimality gap is small because of the property of the ASC travel time stated in Property 1. Let B_1 denote the set of bays in Θ_i located before bay $\hat{b}$ (recall that $\hat{b}$ is the minimum

dominant bay that satisfies $T_{\hat{b}r} = T_{\hat{b}}$ for any $1 < r < R$), and B_2 the set of bays in Θ_i located after bay $\hat{b}$ (including bay $\hat{b}$ if $\hat{b}$ is in Θ_i). According to Property 1, for the bays in B_1, the ASC's horizontal travel time for retrieving a container depends on both the row index and the bay index of the stack where the container is stored; and for the bays in B_2, it only depends on the bay index. In the realistic situation, in most cases, $|\mathrm{B}_1|$ is much smaller than B, which means that the bays in B_1 must be occupied. Since for the bays in B_2, the ASC's horizontal travel time depends only on the bay index, there would be not much difference among the solutions with one more stack available or one less stack available in such bays.

On solving the three subproblems at each iteration, the total retrieval time can be obtained. The search process terminates either if the iteration reaches the minimum number of required bays $B^{\min}$ or if the objective function does not improve. $B^{\min}$ is calculated by $B^{\min} = \lceil N / (R \cdot K) \rceil$, supposing each bay can be fully utilized.

5.3 Subproblem 1: smart piles

Subproblem 1 deals with the selection of smart piles. The objective of subproblem 1 is to maximize the number of smart containers. By considering the constraints associated with only smart containers, subproblem 1 for the ith iteration can be formulated as an integer programming model denoted by [Sub1] in Appendix B.1.

In [Sub1], the decisions on the heights of smart piles and the locations of smart piles are bound together by the decision variable $\boldsymbol{x}$. As the locations of smart piles will be determined in subproblem 3, here, we only need to determine smart groups $\boldsymbol{s}$. With $\boldsymbol{s}$, we can obtain the heights of each smart pile. In order to guarantee the feasibility of the original problem, we need to make sure that a feasible solution can be found in subproblem 2, and thus the feasibility of Constraints (22), (25) and (26) in [M2] should be maintained when solving [Sub1]. Therefore, smart groups should be determined such that there are as many smart containers as possible and the remaining storage space is still enough to accommodate the non-smart containers. For this purpose, we design a heuristic rule to select smart groups.

The basic idea of the heuristic rule is to give priority to the groups who can make more contribution to reducing the number of relocations and meanwhile can utilize the storage space more efficiently. For this purpose, we introduce a group score d_c to represent the average height of each pile of group c, which is defined by $d_c = v_c / P_c$. Groups with higher d_c are given priority over those with lower d_c when selecting smart groups. For the groups with the same value of d_c, the groups with a greater number of partitions (P_c) are given priority. The rationality behind this is that a group with a higher d_c has a higher utilization rate of the stacks and thus more space can be saved for storing other smart containers, and a group with a greater number of partitions can bring more smart containers and thus can lead to fewer relocations. Therefore, these groups are more promising to become smart groups in the optimal solution.

The details of the heuristic rule used for determining the smart groups are provided in Appendix B.2.

5.4 Subproblem 2: non-smart piles

After solving subproblem 1, suppose there are N_n non-smart containers, S_s stacks for storing smart containers and S_n stacks for storing non-smart containers. Subproblem 2 determines the heights of non-smart piles. The objective is to minimize the total retrieval time of the N_n non-smart containers. Subproblem 2 can be formulated as a MIP model denoted by [Sub2] in Appendix B.3. As [Sub2] only involves the decisions of non-smart containers, it can be solved by CPLEX efficiently.

[Sub2] is developed under the assumption that a specific storage area in the block has been pre-allocated to the non-smart containers. We temporarily divide the given storage area into smart bays, non-smart bays and mixed bays. The purpose of the division is to construct a set of bay profiles that will not violate the capacity Constraints (25) in Subproblem 3. A bay profile is defined as a feasible assignment of containers to a bay. The division method is described below, and an example illustrating the pre-allocated storage area is provided in Appendix B.4.

The bays in Θ_i are divided into three subsets, which are Θ_i^1, Θ_i^2 and Θ_i^3. Let Φ_K denote the set of full smart piles output from subproblem 2, that is, the smart piles with K containers. First, the first $\lfloor |\Phi_K| / R \rfloor$ bays near the landside in Θ_i, denoted by Θ_i^1, are allocated to Φ_K, which are smart bays. Second,

the next $\lceil S_n / R \rceil$ bays in Θ_i, denoted by Θ_i^2, are allocated to all the non-smart containers, which are non-smart bays. Note that the last bay in Θ_i^2, denoted by $\overline{B}$, can be a mixed bay depending on the value of $S_n \% R$. Let $\hat{R} = S_n \% R$. Recall that we refer to a bay shared by both smart stacks and non-smart stacks as a mixed bay. If $\hat{R} = 0$, bay $\overline{B}$ is a non-smart bay. If $\hat{R} > 0$, bay $\overline{B}$ will be a mixed bay and $\hat{R}$ represent the number of non-smart stacks in this bay. In this case, a part of the stacks of bay $\overline{B}$ are reserved for smart containers and thus are not allowed to store non-smart containers. We suppose that the (R- $\hat{R}$) number of stacks at the right side of bay $\overline{B}$ are reserved for smart piles and the $\hat{R}$ number of stacks at the left side are allocated to non-smart piles. Last, the last $\lfloor S_s / R \rfloor - |\Theta_i^1|$ bays in Θ_i, denoted by Θ_i^3, are reserved for the remaining smart piles that are neither in Θ_i^1 nor in the mixed bay, which are smart bays. Note that $S_n + S_s = B_i$ and $|\Theta_i^1| + |\Theta_i^2| + |\Theta_i^3| = B_i$. If $\hat{R} = 0$, $|\Theta_i^2| \cdot R = S_n$ and $|\Theta_i^1| \cdot R + |\Theta_i^3| \cdot R = S_s$; otherwise, if $\hat{R} > 0$, $\left(|\Theta_i^2| - 1\right) \cdot R + \hat{R} = S_n$ and $|\Theta_i^1| \cdot R + (R - \hat{R}) + |\Theta_i^3| \cdot R = S_s$.

The rationale behind this pre-allocation is that in the optimal solution, according to Proposition 1, a higher stack would be allocated to a bay with a smaller index. Since the height of a non-smart pile will be no greater than K, it is reasonable to pre-allocate the first $|\Theta_i^1|$ bays to full smart piles. In addition, the purpose of solving subproblem 2 is to obtain the heights of non-smart piles rather than their locations which will be determined in subproblem 3. Pre-allocating Θ_i^2 to non-smart containers makes the non-smart piles have competitive heights to compete with the smart piles for the bays near the landside in subproblem 3.

5.5 Subproblem 3: location allocation

After solving subproblems 1 and 2, we have obtained smart piles and non-smart piles, and the relocation time has been determined. Subproblem 3 is to determine the locations of the smart piles and non-smart piles to minimize the ASC travel time, which can be formulated as an integer programming model denoted by [Sub3] in Appendix B.5.

After conducting a preliminary experiment by CPLEX, we found that [Sub3] is still computationally expensive for larger instances. Therefore, we design a heuristic rule to solve [Sub3]. The basic idea behind the heuristic rule is to allocate higher piles to the stacks near the landside. The rationale is that generally, the ASC's horizontal travel time tends to increase with the bay index as stated in Property 1. In the heuristic, first, we construct a set of bay profiles given the set of smart and non-smart piles obtained from Subproblems 1 and 2. The smart and non-smart piles are allocated to the smart and non-smart storage areas respectively which are pre-allocated in subproblem 2. Second, we allocate these bay profiles to the block based on the unit of bays by taking advantage of Property 1. These bay profiles are first sorted in descending order according to the total number of containers in each bay profile and then the sorted bays are assigned to the bays in Θ_i from the landside to the seaside in increasing order of the bay index. If the travel time of the ASC depends only on the bay index, this assignment rule is optimal given a set of bay profiles (see the warehouse layout model in Tompkins et al. (2010)). However, in our problem setting, it cannot promise optimality because the ASC's travel time is measured by Chebyshev distance. Last, we re-allocate the piles in B_1 based on the unit of piles to seek possible savings on the travel time since for $b \in \mathrm{B}_1$, T_{br} also depends on r. A higher pile is re-allocated to a stack with greater T_{br}. The advantage of the heuristic is that we do not need to consider the capacity feasibility since all the bays meet the capacity constraints. The details of the heuristic rule used for allocating the locations of the smart piles and non-smart piles are provided in Appendix B.6.

Based on the locations of each pile, the total ASC travel time can be obtained. Then, the total retrieval time is returned by the sum of the relocation time obtained by [Sub2] and the ASC travel time obtained in subproblem 3.

6. Computational experiments

In this section, we present extensive computational experiments to validate the effectiveness and efficiency of the proposed solution approach. CPLEX 12.9 is used as the MIP solver to solve [M1], [M2], [M3] and [Sub2]. All the experiments are programmed in C++ (VS2015) and are performed on a desktop with Intel® Core ™ i5-7500 3.40 GHz CPU, 8 GB of RAM, and 64-bit Windows 10 Enterprise.

6.1 Experiment design and instance generation

We present four sets of experiments, which are illustrated in Fig. 5. Firstly, we compare the computational efficiency of the two formulations of the non-split model, by which we show the superiority of the improved formulation over the original one. Secondly, we verify the effectiveness and efficiency of the proposed heuristic algorithm by comparing it with the improved formulation solved by CPLEX. Thirdly, we evaluate the effectiveness of the smart stacking strategy by comparing the proposed heuristic with a commonly used rule in practice. We also examine the impacts of customer information and bay utilization rates on the performance and the effectiveness of the smart stacking strategy. Lastly, we compare the performances of the two variants of models to evaluate the benefit of the split policy.

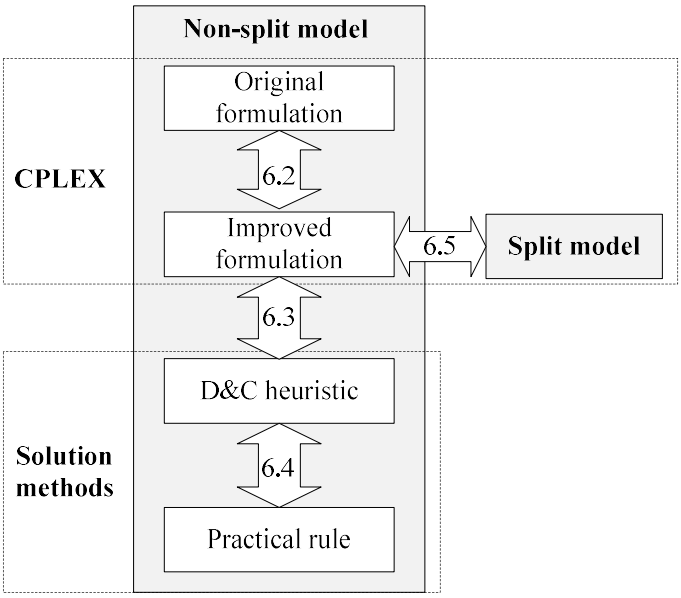


Fig. 5 The diagram of the experiment design

Table 1 lists the sets of parameters used in the experiments. To evaluate the performance of our proposed models and heuristic algorithm under different cases, each parameter is varied within a range of scenarios. Such variations can cover the dimension of a yard block in most of the modern ACTs (Galle et al., 2018b) and the practical scales of containers to be stacked into a block in a planning period (Yu and Qi, 2013). We also set a base instance that represents the dimension and the utilization rate of a bay at a typical ACT. Given N, R, K, and u, the number of available bays (B) in the storage area is obtained by $B=\lceil N/\lfloor R\cdot K\cdot u\rfloor\rceil$. In the experiments, for the convenience of analyzing and interpreting the results, the B bays are located consecutively in the block, that is, from bay 1 to bay B. We use three-tuple, "the number of containers to be stacked (N) | the dimension of the bay ($R\times K$) | the number of bays (B)", to represent the problem class. We do not distinguish the scales of the problem classes strictly because all the relevant factors – N, $R\times K$, and B – have an influence on the computational times of the exact solution. Instead, we regard a problem as a larger problem if it has a larger N while other factors are the same. The parameters associated with the operating speed of the ASC are based on Galle et al. (2018b). The relocation time per container is set to be 120 seconds according to the literature (Zeng et al., 2019) and the terminal practice.

Table 1. Parameters setting for the experiments.

Parameter	Base	Range of scenarios	Fixed parameters
Number of containers (N)		[144, 1296]	
Dimension of bay ($R\times K$)	10×6	[5, 12] × [3, 8]	
Utilization rate (u)	0.8	{0.67, 0.8, 0.85}	
Distribution of group sizes	U[1,10]	Introduced in corresponding sub-sections	
Relocation time per container			120 seconds

ASC speed:	
Trolley speed with load	1.17 meter/second
Gantry speed with load	1.17 meter/second
Hoisting speed without load	0.93 meter/second
Hoisting speed with load	0.47 meter/second
Container size:	
Container width	2.35 meter
Container length	5.90 meter
Container height	2.39 meter

Regarding the distribution of the group sizes, we consider three scenarios of complete information in which the customer information of all containers is known, and three scenarios of incomplete information in which only the customer information of a part of the containers is known. Details will be given in the corresponding sub-sections. Let V denote the size of a group. In the base instance, the group sizes are uniformly distributed in [1,10], which is represented by $V \sim U[1,10]$. For a problem class with the same number of containers and the same distribution of the group sizes, random instances are generated to vary the size of each group. The experiments in Section 6.2 are based on thirty instances for each problem class and the experiments in the other sections are based on ten instances for each problem class, which is adequate to produce the findings.

6.2 Comparison of two formulations of the non-split model

In this section, we verify the superiority of the computational efficiency of the improved formulation over the original formulation. The number of containers N takes the values in the set of {144, 336, 528, 720, 912}, which is sufficient to demonstrate the significant differences between the solution capacity of the two models. Other parameters are the same as the base instance, that is, $R = 10$, $K = 6$, $u = 0.8$, and the group sizes follow the uniform distribution $U[1,10]$. Each problem class includes thirty random instances. Both models are solved using CPLEX given a time budget of one hour. The results of the two models are reported in Table 2. "%Opt" reports the percentage of instances solved to optimum within one hour. "Time (s)" reports the average computation time for the instances solved optimally by both models. "#Var" and "#Con" report the average number of variables and constraints for the instances solved optimally by both models, respectively.

As shown in Table 2, both models obtain optimal solutions for all instances in the problem class with 144 containers, but the original model requires a longer computational time. With the increase in the number of containers, the problem becomes more difficult to solve and some instances cannot be solved to optimum within the one-hour time limit. For the problem classes with 336, 528 and 720 containers, the improved model only takes 0.7% to 2.8% of the time taken by the original model for the instances that can be solved to the optimum. In Table 2, the problem class "912|10×6|19" is the most difficult to solve as shown by that the original model fails to verify the optimum for all the instances within the time limit. For this problem class, the improved model can obtain the optimal solution for 93.3% of the instances in 419 seconds on average per instance. As can be seen from the "#Var" and "#Con" columns, the improved model substantially reduces the number of variables and reduces the number of constraints to some extent, and thus the computational efficiency is improved. Given the superiority of the computational efficiency of the improved model, it is used as a benchmark to evaluate the performance of the proposed heuristic algorithm in Section 6.3.

Table 2. Results of two formulations of the non-split model.

Problem class	Original formulation [M1]				Improved formulation [M2]			
$N\|R\times K\|B$	%Opt	Time(s)	#Var	#Con	%Opt	Time(s)	#Var	#Con
144|10×6|3	100	12.5	5262	340	100	3.8	600	319
336|10×6|7	66.7	843	26345	789	100	23.7	1397	735
528|10×6|11	30.0	1331.9	65117	1241	100	14.9	2197	1151
720|10×6|15	6.7	1707.9	113239	1685	93.3	12.4	2989	1567
912|10×6|19*	0	-	193302	2144	93.3	419.2	3796	1983

Note. For the problem class "912|10×6|19", the results of [M1] are the average of all the instances, and the results of [M2] are the average of the instances that are solved to optimality.

6.3 Comparison of the improved model and the heuristic

To validate the efficiency and effectiveness of the proposed D&C heuristic for solving the non-split model, we compare the performance of the heuristic with that of the improved formulation [M2] solved by CPLEX. CPLEX is given a time limit of one hour and returns the best solutions found so far, i.e., upper bounds when reaching the time limit. We use the default relative MIP gap of CPLEX that is 0.01%, which means CPLEX will stop as soon as it has found a feasible solution proved to be within 0.01% of optimal. Table 3 reports the results of the base instances with a range of batch sizes (N). "LB" and "UB" report the lower bound and upper bound obtained by CPLEX within the time limit, respectively. "Gap_c" reports the average gap (in percentage) between the upper bound (UB) and the lower bound (LB) obtained by CPLEX. The solutions obtained by the heuristic are reported in "Obj". "LB", "UB" and "Obj" are all reported in seconds. "Gap_h" reports the average gap (in percentage) of the solutions obtained by the heuristic against the lower bound obtained by CPLEX. "CPU(s)" reports the average computational time for the ten instances in each problem class.

From Table 3, it can be seen that CPLEX can obtain optimal solutions when the number of containers is small. As the number of containers increases, the solution time of CPLEX increases dramatically. However, the gaps between the lower bound and upper bound are negligible, which indicates that the improved model can provide near-optimal solutions for practically sized problems. As for the heuristic algorithm, it is much more efficient in that it can produce solutions that are very close to that of CPLEX within just one second.

Table 3. Comparison between CPLEX and the D&C heuristic for the base instances.

Problem class	CPLEX				Heuristic		
u=0.8	LB	UB	CPU(s)	Gap_c	Obj	CPU(s)	Gap_h
144|10×6|3	8127	8127	3.7	0.004	8179	0.1	0.644
336|10×6|7	21833	21835	15.8	0.010	21912	0.2	0.361
528|10×6|11	37448	37452	46.5	0.010	37479	0.5	0.084
720|10×6|15	56421	56427	638.7	0.010	56525	0.5	0.184
912|10×6|19	81336	81345	971.3	0.010	81523	0.6	0.229
1104|10×6|23	107811	107850	1069.6	0.036	108050	0.7	0.222
1296|10×6|27	137376	137407	1611.3	0.023	137515	0.7	0.101
Average			622.4	0.015		0.4	0.261

Note. Gap_c = (UB - LB)/LB×100%, Gap_h = (Obj - LB)/LB×100%.

In order to show the effectiveness of our heuristic under varying scenarios, we conduct more experiments with varying bay structures and ranges of group sizes. In these experiments, we focus on the problem classes with 1296 containers, which takes the longest solution time by CPLEX as shown in Table 3. The results are reported in Tables 4-6. The effectiveness of our heuristic is confirmed by the small gaps that are less than 0.6% across all the instances in Tables 4-6, as shown in the column "Gap_h". Besides, the running times of the heuristic for all these instances are less than six seconds, which confirms its efficiency. The efficiency of the heuristic owes to the decoupling of the decisions of smart stacks and non-smart stacks, and the high solution efficiency of the sub-problems.

Table 4. Comparison between CPLEX and the D&C heuristic for instances with large-scale containers, u=0.8 and V~U[1,10].

Problem class	CPLEX				Heuristic		
u=0.8	LB	UB	CPU(s)	Gap_c	Obj	CPU(s)	Gap_h
1296|10×3|54	169060	169071	1.8	0.007	169077	0.4	0.010
1296|10×4|41	141608	141622	10.8	0.010	141632	0.2	0.017
1296|10×5|33	128170	128182	64.9	0.009	128229	0.2	0.046
1296|10×6|27	137376	137407	1611.3	0.023	137515	0.7	0.101
1296|10×7|24	135920	136124	2749.6	0.150	136194	1.3	0.202
1296|10×8|21	136445	136530	1821.4	0.062	136881	1.8	0.320

1296|6×6|47	181529	181575	1215.2	0.025	182215	0.5	0.378
1296|8×6|35	150179	150233	1204.1	0.036	150470	0.6	0.194
1296|12×6|23	120467	120483	1615.7	0.013	120608	0.6	0.117
Average			1143.8	0.037		0.7	0.154

Note. Gap_c = (UB - LB)/LB×100%, Gap_h = (Obj - LB)/LB×100%.

Table 5. Comparison between CPLEX and the D&C heuristic for instances with large-scale containers, u=0.85 and V~U[1,10].

Problem class	CPLEX				Heuristic		
u=0.85	LB	UB	CPU(s)	Gap_c	Obj	CPU(s)	Gap_h
1296|10×3|52	169060	169071	1.6	0.007	169072	0.3	0.007
1296|10×4|39	144856	144870	74.6	0.010	144918	0.2	0.043
1296|10×5|31	134490	134504	162.3	0.010	134608	0.4	0.088
1296|10×6|26	145167	145280	2761.0	0.078	145418	0.6	0.173
1296|10×7|22	154631	154830	3600.0	0.129	155187	2.3	0.360
1296|10×8|20	145723	145942	3270.3	0.150	146079	2.2	0.244
1296|6×6|44	197361	197528	1460.7	0.085	197995	0.7	0.321
1296|8×6|33	161959	162056	1498.3	0.060	162154	0.6	0.120
1296|12×6|22	128507	128520	494.2	0.010	128548	1.1	0.032
Average			1480.3	0.060		0.9	0.154

Note. Gap_c = (UB - LB)/LB×100%, Gap_h = (Obj - LB)/LB×100%.

Table 6. Comparison between CPLEX and the D&C heuristic for instances with large-scale containers, u=0.85 and V~U[1,5].

Problem class	CPLEX				Heuristic		
u=0.85	LB	UB	CPU(s)	Gap_c	Obj	CPU(s)	Gap_h
1296|10×3|52	183838	183854	7.7	0.009	183854	0.4	0.009
1296|10×4|39	166948	166964	444.0	0.010	167068	0.4	0.072
1296|10×5|31	147366	147380	181.2	0.010	147445	0.5	0.054
1296|10×6|26	164986	165004	2421.2	0.011	165055	1.0	0.042
1296|10×7|22	180942	180998	3600.0	0.031	181886	4.3	0.522
1296|10×8|20	185912	185970	3600.0	0.031	186913	5.7	0.538
1296|6×6|44	226370	226392	103.0	0.010	226525	1.2	0.068
1296|8×6|33	185231	185286	3088.8	0.030	185420	0.9	0.102
1296|12×6|22	147294	147309	143.0	0.010	147398	0.8	0.071
Average			1509.9	0.017		1.7	0.164

Note. Gap_c = (UB - LB)/LB×100%, Gap_h = (Obj - LB)/LB×100%.

There is an interesting observation from the comparison of Tables 4 and 5. The problems with higher utilization (0.85) are generally more computationally expensive than those with lower utilization (0.8), although the former problems have fewer bays and thus a smaller number of variables and constraints. One possible reason is that the smart groups are determined in such a way that the groups whose average partition height is larger than $K*u$ are more likely to be smart groups, which we call "advantageous groups". Therefore, the greater the percentage of advantageous groups, the less the number of nodes that will be explored during the CPLEX branch-and-bound process, as those nodes that do not include the advantageous groups will be cut with a high probability. When the utilization rate increases, the percentage of advantageous groups decreases, and thus more nodes need to be explored to reach optimality. The heuristic performs robustly, which can be seen by the very similar optimal gaps in columns Gap_h between Table 4 and Table 5.

Overall, the improved model solved by the standard commercial solver CPLEX performs well when the computational efficiency is not a concern. However, in realistic situations, the dynamic changing of the yard status and the uncertainties of the incoming containers may require the terminal operators to make more frequent decisions quickly according to the dynamically updated information. Our proposed heuristic algorithm can generate near-optimal solutions with fairly comparable accuracy in substantially reduced time. From the application point of view, the heuristic algorithm is more readily available for

real-time decision-making because it is able to provide high-quality and robust solutions within very short running times (in seconds).

Given the high solution quality and efficiency of the proposed heuristic algorithm, the algorithm will be used to conduct the experiments of sensitivity analysis in Section 6.4.

6.4 The effectiveness of smart stacking

In this section, we evaluate the effectiveness of smart stacking by comparing the proposed heuristic algorithm with the random stacking strategy. The random stacking strategy is commonly used in practice when no information can be used to support stacking decision-making (Dekker et al., 2006). In random stacking, the containers are evenly spread over the stacks in the storage area to reduce the number of relocations. Various instances with different scenarios of group information (Section 6.4.1) and yard utilization rates (Section 6.4.2) are tested to investigate their impacts on the performance and the effectiveness of the smart stacking strategy.

6.4.1 Impact of group information

We first analyze the impact of the amount of group information and then the impact of the range of group size.

6.4.1.1 Impact of the amount of group information

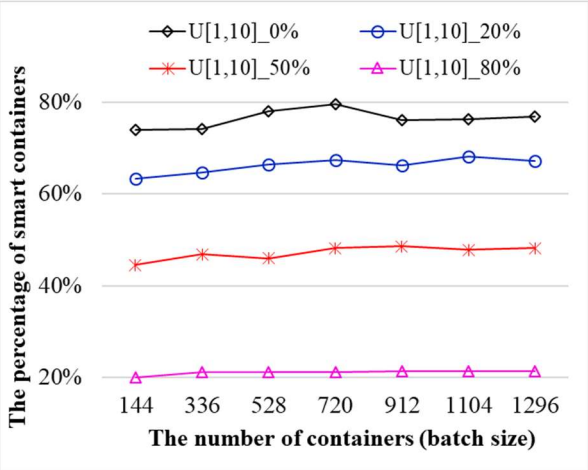
In reality, terminal operators might have incomplete information when only a part of the containers are provided with the information of customer identities or only a part of the customers are willing to participate in the IFF program. To describe the amount of group information, we assume a certain percentage of containers (20%, 50%, and 80%) without group information. For these containers, each one forms a group on its own. For example, "*U*[1, 10]_20%" represents that for 20% of the containers, each one forms a group, while for the remaining 80% containers, their group sizes obey uniform distribution *U*[1,10].

Table 7 presents the results of the scenarios with varying amounts of group information. "*U*[1,10]" represents the scenario of complete information where all the group sizes are generated from the uniform distribution [1,10]. The columns "Obj_1" and column "Obj_2" report the total retrieval time for the smart stacking and random stacking respectively. The columns "Gap(%)" report the relative difference (in percentage) between the two stacking strategies. The results show that the performance of smart stacking measured in total retrieval time depends on the amount of group information. The total retrieval time increases as the amount of information decreases, which is consistent with intuition. Moreover, the less the amount of information, the smaller the gap is between the smart and random stacking strategies. Obviously, with limited storage space, containers without group information are very unlikely to be allocated to smart stacks (Fig. 6), leading to an increasing percentage of relocation time (Fig. 7), and thus the benefit of smart stacking is decreasing. Nevertheless, smart stacking can still lead to an improvement of around 12% compared with random stacking even when only 20% of containers are provided with group information, as shown in the column of *U*[1,10]_80% in Table 7.

Table 7. The results of smart stacking under different amounts of group information.

Problem class	Smart stacking								Random stacking
	U[1,10]		*U*[1,10]_20%		*U*[1,10]_50%		*U*[1,10]_80%		
u=0.8	Obj_1	Gap(%)	Obj_1	Gap(%)	Obj_1	Gap(%)	Obj_1	Gap(%)	Obj_2
144|10×6|3	8179	46.1	9211	39.4	11045	27.3	13367	12.0	15187
336|10×6|7	21912	43.3	24028	37.8	28123	27.2	33961	12.1	38624
528|10×6|11	37479	43.0	41665	36.7	48980	25.6	57969	11.9	65803
720|10×6|15	56525	41.6	62325	35.6	71920	25.6	85304	11.8	96724
912|10×6|19	81523	38.0	87437	33.5	98739	24.9	115914	11.8	131396
1104|10×6|23	108050	36.4	113645	33.1	129424	23.8	149845	11.8	169811
1296|10×6|27	137515	35.1	145424	31.4	162873	23.2	187219	11.7	211967

Note. Gap(%) = (Obj_2- Obj_1)/ Obj_2×100%.



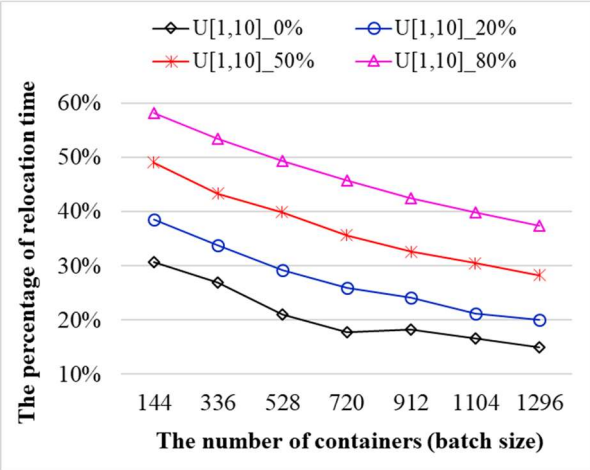


Fig. 6. The percentage of smart containers in smart stacking Fig. 7. The percentage of relocation time in the total retrieval time in smart stacking

6.4.1.2 Impact of the range of group size

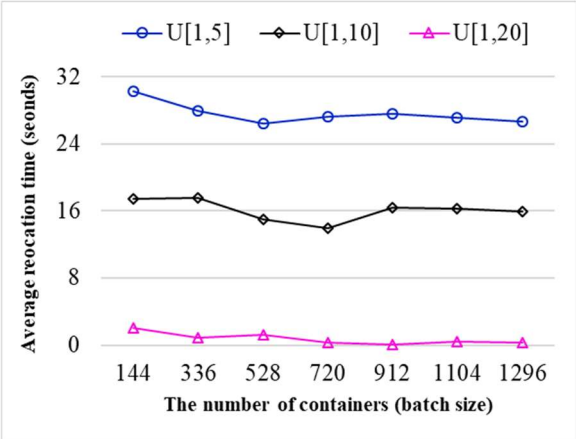
We now analyze the impact of the range of group size on the performance and the effectiveness of smart stacking. Table 8 compares the smart stacking with the random stacking for three scenarios of the range of group size, where the maximum group size is 5, 10, and 20 respectively. We also report the average relocation time per container and the ASC's average travel time per container (in seconds) in smart stacking, which are depicted in Fig. 8 and Fig. 9, respectively. The performances of smart stacking measured in terms of the total retrieval time (columns Obj_1) and average relocation time (Fig. 8) improve as the range of group size enlarges. This is intuitive because a wider range of group size can bring more full partitions (the partitions whose heights equal K) and thus more smart containers. The more interesting observation is that the ASC's average travel time shows a slightly improved trend (Fig. 9) with the enlarging of the group size. This indicates that smart stacking can also help in saving travel time. This observation can be understood from the properties of the ASC travel time (Property 1). When the range of group size gets larger, the heights of smart stacks increase, and thus a greater number of higher stacks will occupy the stacks closer to the landside. Since a higher slot needs less hoisting time and the bays closer to the landside need less gantry moving time, the ASC's total travel time will decrease.

Furthermore, when comparing the results of the smart stacking with the random stacking (columns Gap(%)), the wider the range of group size, the larger the gap is between the two stacking strategies. This observation can also be explained by the increased percentage of smart containers.

Table 8. The results of smart stacking under different ranges of group size.

Problem class	Smart stacking						Random stacking
	$U[1,5]$		$U[1,10]$		$U[1,20]$		
u=0.8	Obj_1	Gap(%)	Obj_1	Gap(%)	Obj_1	Gap(%)	Obj_2
144|10×6|3	10171	33.0	8179	46.1	5828	61.6	15187
336|10×6|7	26115	32.4	21912	43.3	15913	58.8	38624
528|10×6|11	45071	31.5	37479	43.0	29460	55.2	65803
720|10×6|15	68832	28.8	56525	41.6	45419	53.0	96724
912|10×6|19	95927	27.0	81523	38.0	64672	50.8	131396
1104|10×6|23	125965	25.8	108050	36.4	87874	48.3	169811
1296|10×6|27	159364	24.8	137515	35.1	113606	46.4	211967

Note. Gap(%) = $(Obj_2 - Obj_1)/ Obj_2 \times 100\%$.



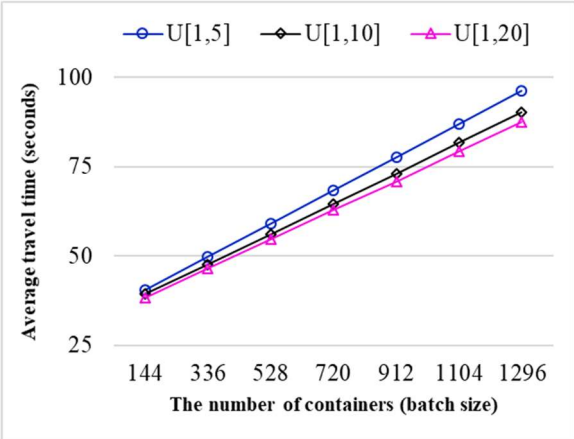


Fig. 8. The average relocation time in smart stacking Fig. 9. The average travel time in smart stacking

6.4.2 Impact of utilization rate

The utilization rate of the yard storage space varies in terminals and over time. Some terminals have adequate space and may prefer a lower utilization rate because this reduces the risk of relocation. In contrast, others may prefer to make the best use of the stack height to accommodate more containers (Bruns et al., 2016). Table 9 compares the smart stacking and the random stacking under three utilization rates.

As the utilization rate increases, the total retrieval time of smart stacking (column "Obj_1") increases, and the gap between the smart and the random stacking strategies (column "Gap(%)") decreases. This is mainly due to the decreasing percentage of smart containers (column "Sm(%)"). With the increase of the utilization rate, groups with lower average partition heights are less possible to become smart groups because that will result in wasted space. Thus, the percentage of smart containers decreases as the utilization rate increases. As a result, the average relocation time (column "Avg_r") increases. However, the average travel time (column "Avg_t") decreases. This is because the higher the utilization rate, the higher the average stacking height is, and the fewer the bays near the seaside are occupied. According to the properties of the ASC travel time, a higher slot needs less hoisting time, and the bays closer to the landside need less gantry moving time. Therefore, the ASC travel time decreases.

In addition, with the utilization rate increasing, the two stacking strategies show similar changing trends in the average relocation time and the average travel time. However, the changing trends of the total retrieval time under the two strategies are different. This is because of the trade-off effect of the two components in the objective function. When the utilization rate increases, under the random stacking, the reduction in the average travel time is sufficient (in most of the cases) to cancel out the increase in the average relocation time, whereas, under the smart stacking, the increase in the average relocation time is much more than the reduction in the average travel time.

Table 9. The results of smart stacking under different bay utilization rates for instances with $V \sim U[1,10]$.

Utilization rate	Problem class	Smart stacking				Random stacking			Gap(%)
		Obj_1	Avg_r	Avg_t	Sm(%)	Obj_2	Avg_r	Avg_t	
0.67	912|10×6|**23**	70430	0.5	76.8	95.4	138480	57.2	94.6	49.1
	1104|10×6|**28**	95082	0.3	85.8	96.8	180887	57.0	106.8	47.4
	1296|10×6|**33**	123346	0.6	94.6	94.8	227963	56.9	119.0	45.9
0.80	912|10×6|**19**	81523	16.3	73.1	76.2	131396	63.9	80.2	38.0
	1104|10×6|**23**	108050	16.3	81.6	76.3	169811	63.9	89.9	36.4
	1296|10×6|**27**	137515	15.9	90.2	76.9	211967	63.9	99.6	35.1
0.85	912|10×6|**18**	89850	26.6	71.9	61.3	130817	65.7	77.8	31.3
	1104|10×6|**22**	116051	24.7	80.5	64.1	169709	65.3	88.4	31.6
	1296|10×6|**26**	145418	23.2	89.0	66.2	212156	65.1	98.6	31.5

Note. Gap(%) = $(Obj_2 - Obj_1)/ Obj_2 \times 100\%$.

6.5 Computational comparison of the non-split model and the split model

In this section, we compare the performances of the two variants of smart stacking models to evaluate the benefit of allowing splitting a group between smart stacks and non-smart stacks. Table 10 presents the results of the two models under two scenarios of the range of group size. Both models are solved by CPLEX given a time limit of one hour, and their best objective functions found within the time limit are reported in columns "Obj_1" and "Obj_2" respectively. Columns "Sm(%)" report the average of the percentage of the smart containers for the instances that are solved to optimality by both models so that the two models are comparable by this performance. Columns "Gap(%)" report the relative difference between the objective functions of the two models.

Table 10. Comparison of the two variants of smart stacking models under two scenarios of group size.

Problem class	*U*[1,10]					*U*[1,20]				
	Non-split		Split		Gap	Non-split		Split		Gap
u=0.8	Obj_1	Sm(%)	Obj_2	Sm(%)	(%)	Obj_1	Sm(%)	Obj_2	Sm(%)	(%)
144|10×6|3	8127	74.4	6624	89.7	18.5	5821	96.3	5608	98.0	3.7
336|10×6|7	21835	73.7	18185	90.3	16.7	15900	98.2	15710	98.5	1.2
528|10×6|11	37452	77.6	32765	91.2	12.5	29451	97.9	29122	98.3	1.1
720|10×6|15	56427	83.4	50880	93.1	9.8	45383	99.3	45311	98.8	0.2
912|10×6|19	81345	75.0	72717	90.4	10.6	64620	99.5	64620	98.8	0.0
1104|10×6|23	107850	76.7	97350	90.6	9.7	87834	99.1	87711	98.5	0.1
1296|10×6|27	137407	78.5	125073	91.0	9.0	113552	99.4	113486	98.3	0.1

Note. Gap(%)= (Obj_1 - Obj_2)/ Obj_1×100%.

Based on Table 10, we have two observations. First, the superiority of the spit model over the non-split model is verified. The benefit of allowing splitting is highly related to the range of group size and is less sensitive to the problem class. For the scenarios of *U*[1,10], the relative difference between the two models is in the range of 9% and 18.5%, which is quite significant. For the scenarios of *U*[1,20], the difference is much smaller (less than 4%). This is because the non-split scenarios of *U*[1,20] have a very high percentage of smart containers, which leaves not much space to improve by allowing splitting.

Second, the reduction in the total retrieval time when allowing splitting (column "Gap(%)") is the result of an increase in the percentage of smart containers (comparing the columns "Sm(%)" of the two models). This is because, in the split model, the smart container decision is not bound to groups but partitions. Such flexibility enables more containers to merit smart stacks. When comparing the columns "Sm(%)" of the two models under the scenarios of *U*[1,20], it is observed that in the last four problem classes, the percentage of smart containers for the split model is even smaller than that of the non-split model. This counterintuitive phenomenon can be explained as follows. The experiments' results show that there exist some stacks that are assigned with only one container. These stacks can be regarded either as non-smart stacks by h_{br}=1 or as smart stacks by x_{br}^1=1, which essentially leads to multiple optimal solutions with different "Sm(%)" performances.

7. Conclusions

This paper addresses the Storage Location Assignment Problem (SLAP) for import containers at an automated container terminal. We proposed a new container stacking strategy, the Smart Stacking (SS) strategy. The SS strategy is motivated by a practical container delivery program, import free flow (IFF), aiming at eliminating the need for relocations and realizing rapid retrieval flow. Under the SS strategy, the smart (free-flow) containers of a customer are stored at dedicated stacks to guarantee zero relocation in the future retrieval process, while the non-smart (non-free-flow) containers share stacks. We focus on the offline operational-level decision making, in which a batch of import containers are to be assigned to exact slots in a given storage area in a single block to minimize the total retrieval time that is the sum of the relocation time and the crane travel time.

Two policies, non-split policy and split policy, are proposed under the SS strategy, depending on whether a single customer's containers are allowed to be split between smart stacks and non-smart stacks. For the non-split policy, we develop two MIP models: original formulation and improved formulation. The improved formulation utilizes the structural properties of the optimal solution to enhance the representation of certain variables. As a result, the number of variables can be significantly reduced and the solution efficiency is improved. To further improve the solution efficiency, a divide-and-conquer heuristic algorithm is designed that can solve the non-split model in several seconds with an accuracy

within 0.6% for practical scale problems. For the split policy, we develop a MIP model assuming that all smart piles must be selected from the optimal partitions of the non-split model. The objective function value of the split model is proved to be not greater than that of the non-split variant both theoretically and computationally. Extensive computational experiments demonstrate the effectiveness of the proposed SS strategy, models and algorithm.

This paper is the first to propose the concept of smart stacking that enables the terminal operator to incorporate the customer information to optimize import container stacking so that the total retrieval time can be minimized. On the theoretical side, the proposed SS strategy and the SLAPs advance the methods and solutions of import container stacking, and more importantly, bring about novelty in developing research opportunities in this field. On the practical side, this paper produces useful managerial implications. Firstly, the SS strategy can significantly reduce the total retrieval time compared with the traditional practice. The effectiveness of the SS strategy is more significant in the following situations: a larger amount of customer information, a wider range of group sizes, and a lower utilization rate. It is thus recommended that terminal operators should seek collaboration with the customers to access the customer information of the import containers so that port congestion can be mitigated in the retrieval processes, especially with those high-volume customers. Secondly, the proposed heuristic algorithm is fast enough to produce high-quality solutions, which can be applied in practice to support the operational-level decision-making of smart stacking. Thirdly, the retrieval time could be further reduced if the same customers' containers are allowed to be split into smart and non-smart under certain conditions. The proposed models can help terminal operators evaluate the trade-off between the additional benefit and the extra administration cost when adopting the split policy.

Future research may be conducted in the following directions. Firstly, the split policy deserves more research, e.g. investigating the optimality of the split variant and examining the relationship between the characteristics of a specific problem environment (e.g., the distribution of customer volumes/group sizes, the maximum stacking height and the utilization rate) and the reduction in retrieval time compared with the split variant. Secondly, more information about containers could be incorporated into our models such as container retrieval time and customer priority. By incorporating predictive retrieval times or advanced appointments (e.g. via vehicle booking system/truck appointment system), the smart stacking strategy can be further enhanced as there are more opportunities to save storage space because containers of different customers can share stacks according to their retrieval times. It is therefore interesting to incorporate the temporal information of containers into the models and evaluate its value. Thirdly, the SLAP studied in this paper is oriented toward a short-term operational problem. It is interesting to study the long-term storage space allocation problems that concern the storage and allocation of import containers from multiple vessels to different blocks in the yard over a planning horizon. The models and solution method developed in this paper will function as a building block for this higher-level decision. Fourthly, the model can be extended to consider more realistic constraints, such as the weight constraint, which may require light-weighted containers to be stacked above heavy ones for safety reasons. The last meaningful research direction is to apply the smart stacking strategy at rail-water intermodal container terminals. Note that a single rail carrier could have hundreds of containers from each vessel and these containers may go for the same rail head; this makes such intermodal terminals suitable for smart stacking.

Acknowledgments

This study is partially supported by the China Scholarship Council, the Royal Society (Grant No. IEC\NSFC\170100), and the EU H2020 (Grant No. 777742, EC H2020-MSCA-RISE-2017).

Reference

Azab, A., & Morita, H. (2021). The Block Relocation Problem with Appointment Scheduling. European Journal of Operational Research. In press. https://doi.org/10.1016/j.ejor.2021.06.007.

Bacci, T., Mattia, S., & Ventura, P. (2020). A branch-and-cut algorithm for the restricted Block Relocation Problem. European Journal of Operational Research, 287(2), 452-459.

Boge, S., Goerigk, M., & Knust, S. (2020). Robust optimization for premarshalling with uncertain priority classes. European Journal of Operational Research, 287(1), 191-210.

Boge, S., & Knust, S. (2020). The parallel stack loading problem minimizing the number of reshuffles in the retrieval stage. European Journal of Operational Research, 280(3), 940-952.

Borgman, B., van Asperen, E., & Dekker, R. (2010). Online rules for container stacking. OR spectrum, 32(3), 687-716.
Bruns, F., Knust, S., & Shakhlevich, N. V. (2016). Complexity results for storage loading problems with stacking constraints. European Journal of Operational Research, 249(3), 1074-1081.
Carlo, H. J., Vis, I. F., & Roodbergen, K. J. (2014a). Storage yard operations in container terminals: Literature overview, trends, and research directions. European journal of operational research, 235(2), 412-430.
Carlo, H. J., Vis, I. F., & Roodbergen, K. J. (2014b). Transport operations in container terminals: Literature overview, trends, research directions and classification scheme. European journal of operational research, 236(1), 1-13.
Caserta, M., Schwarze, S., & Voß, S. (2020). Container rehandling at maritime container terminals: A literature update. In J. W. Böse (Ed.), *Handbook of Terminal Planning* (2nd ed.) (pp. 343-382). Springer International Publishing.
Chang, Y., & Zhu, X. (2019). A Novel Two-Stage Heuristic for Solving Storage Space Allocation Problems in Rail–Water Intermodal Container Terminals. Symmetry, 11(10), 1229.
De Castillo, B., & Daganzo, C. F. (1993). Handling strategies for import containers at marine terminals. Transportation Research Part B: Methodological, 27(2), 151-166.
de Melo da Silva, M., Erdoğan, G., Battarra, M., & Strusevich, V. (2018). The block retrieval problem. European Journal of Operational Research, 265(3), 931-950.
Dekker, R., Voogd, P., & van Asperen, E. (2006). Advanced methods for container stacking. OR spectrum, 28(4), 563-586.
Dupin, C. (2015). Improving on 'Free-Flow'. Freightwaves.com. March 6, 2015. Retrieved from https://www.freightwaves.com/news/improving-on-free-flow. Accessed January 13, 2021.
DP World London Gateway. (2018). DP World's UK logistics facilities already have the customs clearance, inspection facilities and infrastructure in place to keep trade flowing. Retrieved from https://www.londongateway.com/news-media/blogs/unpacking-brexit. Accessed January 14, 2021.
Feng, Y., Song, D. P., Li, D., & Zeng, Q. (2020). The stochastic container relocation problem with flexible service policies. Transportation Research Part B: Methodological, 141, 116-163.
Gaete, M., González-Araya, M. C., González-Ramírez, R. G., & Astudillo, C. (2017). A Novel Storage Space Allocation Policy for Import Containers. International Conference on Operations Research and Enterprise Systems, vol. 884, Springer, 2017, pp. 293–316.
Galle, V., Boroujeni, S. B., Manshadi, V. H., Barnhart, C., & Jaillet, P. (2016). An average-case asymptotic analysis of the Container Relocation Problem. Operations Research Letters, 44(6), 723-728.
Galle, V., Manshadi, V. H., Boroujeni, S. B., Barnhart, C., & Jaillet, P. (2018a). The stochastic container relocation problem. Transportation Science, 52(5), 1035-1058.
Galle, V., Barnhart, C., & Jaillet, P. (2018b). Yard Crane Scheduling for container storage, retrieval, and relocation. European Journal of Operational Research, 271(1), 288-316.
Gharehgozli, A. H., Yu, Y., de Koster, R., & Udding, J. T. (2014). A decision-tree stacking heuristic minimising the expected number of reshuffles at a container terminal. International Journal of Production Research, 52(9), 2592-2611.
Gharehgozli, A., & Zaerpour, N. (2018). Stacking outbound barge containers in an automated deep-sea terminal. European Journal of Operational Research, 267(3), 977-995.
Guldogan, E. U. (2011). Simulation-based analysis for hierarchical storage assignment policies in a container terminal. Simulation, 87(6), 523-537.
Güven, C., & Türsel Eliiyi, D. (2019). Modelling and optimisation of online container stacking with operational constraints. Maritime Policy & Management, 46(2), 201-216.
He, Y., Wang, A., & Su, H. (2019). The impact of incomplete vessel arrival information on container stacking. International Journal of Production Research, 1-15.
Iris, Ç., Christensen, J., Pacino, D., & Ropke, S. (2018). Flexible ship loading problem with transfer vehicle assignment and scheduling. Transportation Research Part B: Methodological, 111, 113-134.
Jang, D. W., Kim, S. W., & Kim, K. H. (2013). The optimization of mixed block stacking requiring relocations. International Journal of Production Economics, 143(2), 256-262.
Jiang, X., Lee, L. H., Chew, E. P., Han, Y., & Tan, K. C. (2012). A container yard storage strategy for improving land utilization and operation efficiency in a transshipment hub port. European journal of

operational research, 221(1), 64-73.
Jiang, X., Chew, E. P., Lee, L. H., & Tan, K. C. (2013). Flexible space-sharing strategy for storage yard management in a transshipment hub port. OR spectrum, 35(2), 417-439.
Jin, J. G., Lee, D. H., & Cao, J. X. (2016). Storage yard management in maritime container terminals. Transportation Science, 50(4), 1300-1313.
Kang, J., Ryu, K. R., & Kim, K. H. (2006). Deriving stacking strategies for export containers with uncertain weight information. Journal of Intelligent Manufacturing, 17(4), 399-410.
Kim, K. H. (1997). Evaluation of the number of rehandles in container yards. Computers & Industrial Engineering, 32(4), 701-711.
Kim, K. H., & Kim, H. B. (1999). Segregating space allocation models for container inventories in port container terminals. International Journal of Production Economics, 59(1-3), 415-423.
Kim, K. H., Park, Y. M., & Ryu, K. R. (2000). Deriving decision rules to locate export containers in container yards. European Journal of Operational Research, 124(1), 89-101.
Kim, K. H., & Park, K. T. (2003). A note on a dynamic space-allocation method for outbound containers. European Journal of Operational Research, 148(1), 92-101.
Kizilay, D., & Eliiyi, D. T. (2020). A comprehensive review of quay crane scheduling, yard operations and integrations thereof in container terminals. Flexible Services and Manufacturing Journal, 1-42.
Ku, D., & Arthanari, T. S. (2016a). Container relocation problem with time windows for container departure. European Journal of Operational Research, 252(3), 1031-1039.
Ku, D., & Arthanari, T. S. (2016b). On the abstraction method for the container relocation problem. Computers & Operations Research, 68, 110-122.
Lee, L. H., Chew, E. P., Tan, K. C., & Han, Y. (2007). An optimization model for storage yard management in transshipment hubs. In Container Terminals and Cargo Systems (pp. 107-129). Springer, Berlin, Heidelberg.
Lee, W. S., Ottjes, J. A., Veeke, H. P., & Rijsenbrij, J. C. (2008). Using container call time information for restacking reduction. In Proc. 6th International Industrial Simulation Conference, pp. 293- 298.
Lehnfeld, J., & Knust, S. (2014). Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. European Journal of Operational Research, 239(2), 297-312.
Li, Z., Zhu, S., & Zhou, M. (2009). A divide-and-conquer strategy to deadlock prevention in flexible manufacturing systems. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 39(2), 156-169.
Lin, D. Y., & Chiang, C. W. (2017). The storage space allocation problem at a container terminal. Maritime Policy & Management, 44(6), 685-704.
Luo, J., Wu, Y., & Mendes, A. B. (2016). Modelling of integrated vehicle scheduling and container storage problems in unloading process at an automated container terminal. Computers & Industrial Engineering, 94, 32-44.
Maldonado, S., González-Ramírez, R. G., Quijada, F., & Ramírez-Nafarrate, A. (2019). Analytics meets port logistics: A decision support system for container stacking operations. Decision Support Systems, 121, 84-93.
Monaco, M. F., Sammarra, M., & Sorrentino, G. (2014). The terminal-oriented ship stowage planning problem. European Journal of Operational Research, 239(1), 256-265.
Mongelluzzo, B., 2015a. Free-flow program finds success in Los Angeles port. JOC.com. May 22, 2015. Retrieved from https://www.joc.com/port-news/us-ports/port-los-angeles/free-flow-program-finds-success-los-angeles-port_20150522.html. Accessed January 13, 2021.
Mongelluzzo, B., 2015b. Free-flow container operation launched in Port of Los Angeles. JOC.com. Feb 27, 2015. https://www.joc.com/port-news/us-ports/port-los-angeles/initiative-aimed-reducing-los-angeles-port-congestion-launched_20150227.html. Accessed January 13, 2021.
Park, T., Choe, R., Kim, Y. H., & Ryu, K. R. (2011). Dynamic adjustment of container stacking policy in an automated container terminal. International Journal of Production Economics, 133(1), 385-392.
Petering, M. E., Wu, Y., Li, W., Goh, M., de Souza, R., & Murty, K. G. (2017). Real-time container storage location assignment at a seaport container transshipment terminal: dispersion levels, yard templates, and sensitivity analyzes. Flexible Services and Manufacturing Journal, 29(3-4), 369-402.
Preston, P., & Kozan, E. (2001). An approach to determine storage locations of containers at seaport terminals. Computers & Operations Research, 28(10), 983-995.
Parreño-Torres, C., Alvarez-Valdes, R., & Ruiz, R. (2019). Integer programming models for the pre-

marshalling problem. European Journal of Operational Research, 274(1), 142-154.
Parker, B. (2015). Reduce Port Congestion By Expanding Container 'Free-Flow' Systems. SupplyChainBrain.com, December 14, 2015. Retrieved from https://www.supplychainbrain.com/articles/22934-reduce-port-congestion-by-expanding-container-free-flow-systems. Accessed January 13, 2021.
Razouk, C., Benadada, Y., & Boukachour, J. (2016). New approaches for solving the container stacking problem. In 2016 3rd International Conference on Logistics Operations Management (GOL) (pp. 1-7). IEEE.
Rasmussen, M. S., & Larsen, J. (2011). Optimisation-based solution methods for set partitioning models (Doctoral dissertation, Ph. D. thesis, Technical University of Denmark).
Rekik, I., & Elkosantini, S. (2019). A multi agent system for the online container stacking in seaport terminals. Journal of Computational Science, 35, 12-24.
Rekik, I., Elkosantini, S., & Chabchoub, H. (2018). A case based heuristic for container stacking in seaport terminals. Advanced Engineering Informatics, 38, 658-669.
Reimann, M., Doerner, K., & Hartl, R. F. (2004). D-ants: Savings based ants divide and conquer the vehicle routing problem. Computers & Operations Research, 31(4), 563-591.
Sauri, S., & Martin, E. (2011). Space allocating strategies for improving import yard performance at marine terminals. Transportation Research Part E: Logistics and Transportation Review, 47(6), 1038-1057.
Sculli, D., & Hui, C. F. (1988). Three dimensional stacking of containers. Omega, 16(6), 585-594.
Smith, D. R. (1985). The design of divide and conquer algorithms. Science of Computer Programming, 5, 37-58.
Tang, L., Jiang, W., Liu, J., & Dong, Y. (2015). Research into container reshuffling and stacking problems in container terminal yards. IIE Transactions, 47(7), 751-766.
Tanaka, S., Tierney, K., Parreño-Torres, C., Alvarez-Valdes, R., & Ruiz, R. (2019). A branch and bound approach for large pre-marshalling problems. European Journal of Operational Research, 278(1), 211-225.
Tanaka, S., & Voß, S. (2021). An exact approach to the restricted block relocation problem based on a new integer programming formulation. European Journal of Operational Research. 296(2), 485-503.
Tompkins, J. A., White, J. A., Bozer, Y. A., & Tanchoco, J. M. A. (2010). *Facilities planning* (4th ed.). John Wiley & Sons.
The Port of Long Beach and Port of Los Angeles, 2017. San Pedro Bay Ports Clean Air Action Plan 2017 Draft Final. https://kentico.portoflosangeles.org/getmedia/9d371f7b-9812-4c75-bcfd-23e83a191435/CAAP_2017_Draft_Document-Final. Accessed January 22, 2021.
Wang, N., Chang, D., Shi, X., Yuan, J., & Gao, Y. (2019). Analysis and design of typical automated container terminals layout considering carbon emissions. Sustainability, 11(10), 2957.
Wang, L., Zhu, X., & Xie, Z. (2020). Efficient container stacking approach to improve handling: efficiency in Chinese rail–truck transshipment terminals. Simulation, 96(1), 3-15.
Wei, C., Gao, W. W., Hu, Z. H., Yin, Y. Q., & Pan, S. D. (2019). Assigning customer-dependent travel time limits to routes in a cold-chain inventory routing problem. Computers & Industrial Engineering, 133, 275-291.
Xia, M., Zhao, N., & Mi, W. (2016). Storage allocation in automated container terminals: the upper level. Polish Maritime Research, 23(s1), 160-174.
Yang, J. H., & Kim, K. H. (2006). A grouped storage method for minimizing relocations in block stacking systems. Journal of Intelligent Manufacturing, 17(4), 453-463.
Yu, M., & Qi, X. (2013). Storage space allocation models for inbound containers in an automatic container terminal. European Journal of Operational Research, 226(1), 32-45.
Yu, M., Liang, Z., Teng, Y., Zhang, Z., & Cong, X. (2021). The inbound container space allocation in the automated container terminals. Expert Systems with Applications, 179, 115014.
Zaerpour, N., Yu, Y., & de Koster, R. B. (2015). Storing Fresh Produce for Fast Retrieval in an Automated Compact Cross‐Dock System. Production and Operations Management, 24(8), 1266-1284.
Zeng, Q., Feng, Y., & Yang, Z. (2019). Integrated optimization of pickup sequence and container rehandling based on partial truck arrival information. Computers & Industrial Engineering, 127, 366-382.

Zhao, W., & Goodchild, A. V. (2010). Impact of truck arrival information on system efficiency at container terminals. Transportation Research Record, 2162(1), 17-24.
Zhao, N., Xia, M., Mi, C., Bian, Z., & Jin, J. (2015). Simulation-based optimization for storage allocation problem of outbound containers in automated container terminals. Mathematical Problems in Engineering, 2015.
Zhang, C., Liu, J., Wan, Y. W., Murty, K. G., & Linn, R. J. (2003). Storage space allocation in container terminals. Transportation Research Part B: Methodological, 37(10), 883-903.
Zhang, C., Chen, W., Shi, L., & Zheng, L. (2010). A note on deriving decision rules to locate export containers in container yards. European Journal of Operational Research, 205(2), 483-485.
Zhang, C., Wu, T., Kim, K. H., & Miao, L. (2014). Conservative allocation models for outbound containers in container terminals. European Journal of Operational Research, 238(1), 155-165.
Zhang, C., Guan, H., Yuan, Y., Chen, W., & Wu, T. (2020). Machine learning-driven algorithms for the container relocation problem. Transportation Research Part B: Methodological, 139, 102-131.
Zhen, L., Jiang, X., Lee, L. H., & Chew, E. P. (2013). A review on yard management in container terminals. Industrial Engineering and Management Systems, 12(4), 289-304.
Zhen, L. (2016). Modeling of yard congestion and optimization of yard template in container ports. Transportation Research Part B: Methodological, 90, 83-104.
Zhou, C., Chew, E. P., & Lee, L. H. (2018). Information-based allocation strategy for GRID-based transshipment automated container terminal. Transportation Science, 52(3), 707-721.
Zhou, C., Wang, W., & Li, H. (2020). Container reshuffling considered space allocation problem in container terminals. Transportation Research Part E: Logistics and Transportation Review, 136, 101869.
Zhu, H., Ji, M., & Guo, W. (2020). Two-stage search algorithm for the inbound container unloading and stacking problem. Applied Mathematical Modelling, 77, 1000-1024.
Zweers, B. G., Bhulai, S., & van der Mei, R. D. (2020). Optimizing pre-processing and relocation moves in the stochastic container relocation problem. European Journal of Operational Research, 283(3), 954-971.