YARD ALLOCATION PROBLEM WITH TIME DIMENSION AT SEAPORT TERMINAL

WANG TIANTIAN

PhD

The Hong Kong Polytechnic University

This programme is jointly offered by The Hong Kong Polytechnic University and Zhejiang University

2021

The Hong Kong Polytechnic University
Department of Logistics and Maritime Studies

Zhejiang University
School of Management

Yard Allocation Problem with Time Dimension at Seaport Terminal

WANG Tiantian

A thesis submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy

June 2021

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for award of any other degree or diploma, except where due acknowledgement has been made in the text.

WANG Tiantian

Publications Arising from the Thesis

The third chapter of this thesis is based on the following publication:

- Wang, T., Ma, H., Xu, Z., & Xia, J. (2021). A new dynamic shape adjustment and placement algorithm for 3D yard allocation problem with time dimension. *Computers & Operations Research*, 105585.

Acknowledgments

First of all, I would like to express my thankfulness to Professor Ma Hong and Professor Xu Zhou for their kind guidance and unconditional support during my research life. I am grateful for the discussions with Dr. Xia Jun. Without their endless help, I would not stand at this current point. Finally, I would thank my parents for their unselfish love and company all these years.

Yard Allocation Problem with Time Dimension at Seaport Terminal

by WANG Tiantian

Department of Logistic and Maritime Studies

The Hong Kong Polytechnic University

Abstract

Seaports constitute the most important cargo distribution terminal in modern logistics. According to statistics released by the United Nations in 2020, the annual container throughput of Port of Singapore had reached nearly 38 million TEUs in 2019, and the number keeps increasing year by year. Large Asian ports have limited land and yet, they play key roles as international maritime hubs. How to balance various facilities to store a growing number of containers within the limited yard space? This is a challenge for port managers.

This thesis considers the yard space allocation problem with a time dimension. The time dimension is mainly reflected in two aspects. First, visits to ports of different vessels have different patterns in terms of time periods, which adds complex constraints when allocating storage spaces at different periods. Second, containers transported by the same vessel always arrive at the yard during a time interval. A consignment strategy can help avoid overlapping of storage space allocations. The storage space should also be non-decreasing with time. In essence, the second yard space allocation problem with time dimension is a three-dimensional packing problem with irregular shapes. This makes it challenging to obtain a high-quality space allocation schedule with existing algorithms. Besides, the complexity caused by the time dimension and the scheduling of yard trucks

are closely related to space allocation. Unreasonable space allocation can lead to traffic congestion in yard lanes and lead to inefficient container handling due to uneven workload distribution, affecting the port operations on the berth side. Therefore, yard space allocation plays a critical role in the overall seaport schedule.

This thesis addresses the yard space allocation problem with the time dimension at container terminals. We build mathematical models and design efficient algorithms by considering traffic congestion in the yard and low space utilization in a yard block. This thesis addresses two optimization problems relevant to the yard space allocation problem: allocation of containers to blocks at the yard level and optimization of flexible storage spaces allocation to different requests at the block level. The main contributions of this thesis are summarized as follows.

(1) Yard-level space optimization. Considering vessels with multiple periods, we take the sub-block as the primary storage space allocation unit in this study. Both the sub-block location decision and the loading and unloading time decisions are considered. To avoid traffic congestion in the yard, we minimize the operating cost of yard trucks and the total time deviation between the actual operating time and the expected handling time of all vessels. We propose a three-stage heuristic algorithm based on a critical element, which can efficiently obtain an allocation schedule that minimizes the transportation cost and time deviations. As a result, containers from each vessel are allocated to different blocks. Each block is assigned with containers from different vessels.

(2) Block-level space optimization. Following the assignation of containers to a block at the yard-level optimization, we aim to obtain a space-saving allocation schedule for this block. We remove the boundaries between the fixed subblocks and consider a rectangular storage space with flexible shapes and positions. Such flexible rectangular spaces are allocated to containers at each time. In this study, a new dynamic shape adjustment and placement algorithm is proposed, which can efficiently obtain a solution with high space utilization in each block. The decision of this optimization problem refines the space allocation and saves available storage space for a block.

In summary, the algorithms proposed in this thesis can coordinate yard space allo-

cation and scheduling of yard trucks, so that the yard can avoid traffic congestion and optimize the space usage in each block. Our research provides an effective yard allocation solution for port managers and can help them make competitive decisions which accommodate more container storage requests in limited yard area.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Background

Globalization is leading to optimal allocation of resources globally and is the most significant economic phenomenon in the 21st century. In the most recent two decades, globalization has been booming despite the global financial crisis, natural disasters, trade wars, and epidemics. On the one hand, even though the world economy plunged because of the financial crisis in 2008, it returned to its previous level only six years later. On the other hand, the Sino-US trade friction that began in 2018 and the COVID-19 epidemic that began in 2020 have hit global trade activities to some extent. However, China quickly controlled the epidemic at home, getting rid of the adverse effects of the epidemic earlier than the United States. As a result, global trade has recovered rapidly. In November 2020, the cargos that US imports from China increased by 22% compared with the same period last year, reaching an ever high level. From the above facts, we can see that economic globalization will remain a mainstream economic trend. Globalization implies movement of many goods between different countries around the world. In the modern supply chain, seaports play a crucial role as distribution hubs of containers. Containers are widely used as standard carriers of goods. Thus, some large ports need to deal with a massive number of containers every day. Figure 1.1 shows the statistics of port throughput from 2015 to 2019 (United Nations 2020). As can be seen, the annual container throughput of Singapore Port has reached nearly 38 million Twenty-feet Equivalent Units (TEUs) in 2019, and this number is still increasing year by year. Therefore, it is essential to improve the operations of containers at the port to enhance the operational efficiency of the entire supply chain.

Container port has been widely concerned by practitioners because of its vital role in

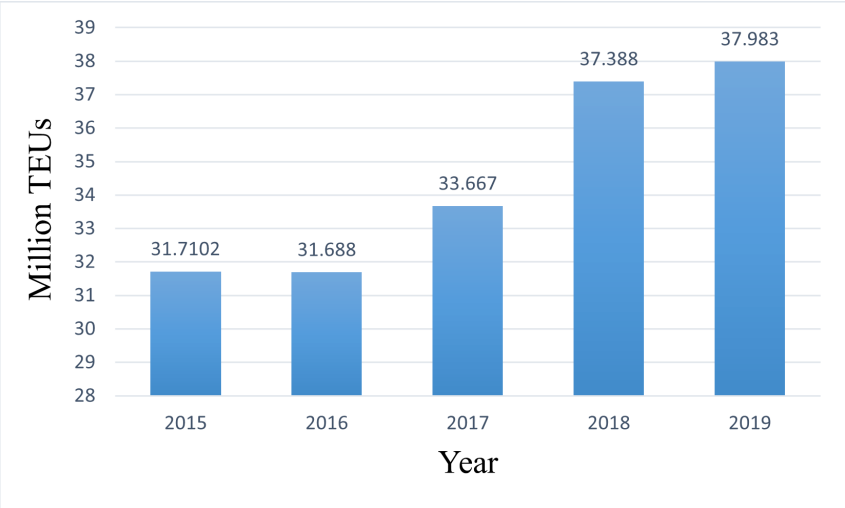


Figure 1.1: Throughput of Port of Singapore from 2015 to 2019

the supply chain. Usually, port optimization includes the berth-side and the yard-side optimization problems. Berth-side optimization comprises the berth allocation problem and the quay crane scheduling problem. Yard-side optimization includes yard allocation problem, truck scheduling problem, and yard crane scheduling problem. In particular, yard allocation is the most critical optimization problem on the yard side, which is closely related to the truck scheduling problem.

From the perspective of port operations, the berth optimization problem has been regarded as the crucial issue which restricts port development. However, with the employment of giant quay cranes, i.e., the twin 40-ft quay cranes, the berth-side operational efficiency has improved considerably. Thus, the bottleneck of port optimization has moved to the yard side. The yard allocation problem is to assign yard storage spaces to containers to improve yard storage capacity. Usually, we can improve the yard storage capacity from two aspects. The first way is to increase yard capacity by expanding the yard by acquiring more land. The second way is to improve space utilization while maintaining the current yard space. However, the cost of expansion is high for most ports. Limited by land resources, there is no extra space available for expansion. Therefore, improving space utilization is the typical way to improve yard storage capacity in practice.

From the perspective of academic research, the yard allocation problem is more complex than the berth allocation problem, and it has attracted more attention in recent years.

On the one hand, we can transform the berth allocation problem into a two-dimensional packing problem because of its favorable modeling properties. Many studies have proposed some efficient solution methods (Buhrkal et al., 2011). However, the yard allocation does not have such properties. In addition, the yard allocation decision is at the end of all port optimization decisions and is affected by the preceding operations (Zhen et al., 2011). These two reasons complicate the yard allocation. On the other hand, according to Google Scholar search, there are about 15500 related studies on berth allocation and 47600 studies on yard space allocation, carried out between 1990 and 2020, revealing that yard optimization is more critical and challenging.

Inappropriate allocation of yard space often leads to port congestion. For example, severe congestion occurred at the container terminal of Shanghai Port in April 2017. Congestion not only caused vessel delays in Shanghai Port but also generated high waiting costs. Besides, congestion at one important port can lead to congestion at other ports worldwide through the shipping supply chain, increasing the cost of the supply chain. It has been common for some major ports to have frequent congestions in the past twenty years. Two kinds of factors lead to port congestion. The first factor is human factors caused by strikes and information asymmetry among all parties in the port. The second factor is that some ports lack scientific support for container operations. During the actual operation of the port, its managers can reduce the congestion caused by the first factor by sharing information with shipping liners and other parties. For the second factor, they have to work out the schedule for the container operations at all stages of port processing through operational optimization. Since the yard allocation problem is crucial, we take congestion into account in this thesis to make decisions more in line with reality.

In addition, the yard allocation problem is closely related to yard truck scheduling—trucks transport containers between the yard and the berth for loading and unloading purposes. When containers in two adjacent subblocks are loaded or unloaded simultaneously, there can be traffic congestion in lanes within the yard (Zhen et al., 2016). Yard congestion leads to loading and unloading taking more time than the vessel's expected handling time. As a result, a longer waiting time for the vessel is inevitable. Usually,

severe yard congestion also leads to port congestion and can cause unexpected costs to the port, vessels, trucks, and other parties. Therefore, the interaction between yard space allocation and truck scheduling jointly affects the traffic congestion in the yard.

Time is another factor highly relevant to the yard allocation problem. First, vessels from the same ship liner usually visit the port within a fixed period, and the periods of different ship liners are often different (Zhen et al., 2016). Different storage spaces need to be allocated to different vessels. Even for vessels from the same ship liner, the actual number of loaded containers arriving at the port may be different in each period. Therefore, we need to assign different yard spaces for them. Because of the multi-periodic properties of vessels arising in the time dimension, yard allocation becomes more complex. Second, considering the actual arrival of the containers, we often need to resolve changes in storage spaces allocation after containers arrive. According to the consignment strategy, containers shipped to the same destination are stored in the continuous yard storage space. Containers to different destinations cannot share the same storage space. In practice, each batch of containers destined for the same destination comes from several sources, and each container from each source arrives at an independent time. Therefore, for a batch of containers shipped to the same destination, the storage spaces required at different times are different. Moreover, the storage space of these containers remains non-decreasing over time before they are moved away (Chen et al., 2002). Considering the above two reasons, we can say that the yard allocation problem is closely related to time.

To sum up, right allocation of yard space helps improve the efficiency of the port. In order to solve the yard allocation problem, we need to consider the scheduling of trucks and the time dimension. Based on the two factors, this thesis proposes a way to optimize storage space allocation in the yard.

1.2 Research Topics and Objectives

The primary research framework of this thesis is depicted in Figure 1.2. First, two research problems are extracted and articulated out of the practical yard problems and the relevant literature. Then, we investigate the two problems for planning for allocations at the yard, shifting from large to small units and from the fixed subblock to the flexible rectangle. In the first topic, we take the whole yard as the planning space. We assign subblocks to vessels with multiple periods. We try to avoid traffic congestion by decisions on subblock locations and handling time. Then we take one block as the planning space. Unlike the fixed subblock unit in the first topic, we adopt a flexible rectangular storage space as the basic space allocation unit. For each research topic, this thesis considers the influence of time on space allocation. The relationship between the two topics are displayed in Figure 1.3. The details of the two topics are as follows.

(1) In the first topic, we take the whole yard as the planning space. The primary space allocation unit is a fixed subblock. We study the subblock allocation problem for transshipment vessels. Each vessel has multiple equal periods, and different vessels visit the port in different periods. Containers are transferred between different vessels. During the transshipment process, containers are temporarily stored in several subblocks. Specifically, the decisions to assign containers are based on the location and the handling time. The aim is to reduce the total traveling distance of trucks and the total time spent by vessels for loading and unloading during a given planning horizon.

(2) In the second topic, we consider the space optimization problem in a block. The planning space is a block, and the primary space allocation unit is the adjustable rectangular space. This adjustable rectangle relaxes the fixed boundary of the subblock and is more flexible than fixed subblock. This means length and width of the space are adjustable according to the actual number of containers. With the arrival of containers, storage space for the containers keeps increasing. Thus, the flexible storage space varies with time, which completes the problem. To improve the space utilization of a single subblock, we build a mathematical model and design efficient algorithms to handle this

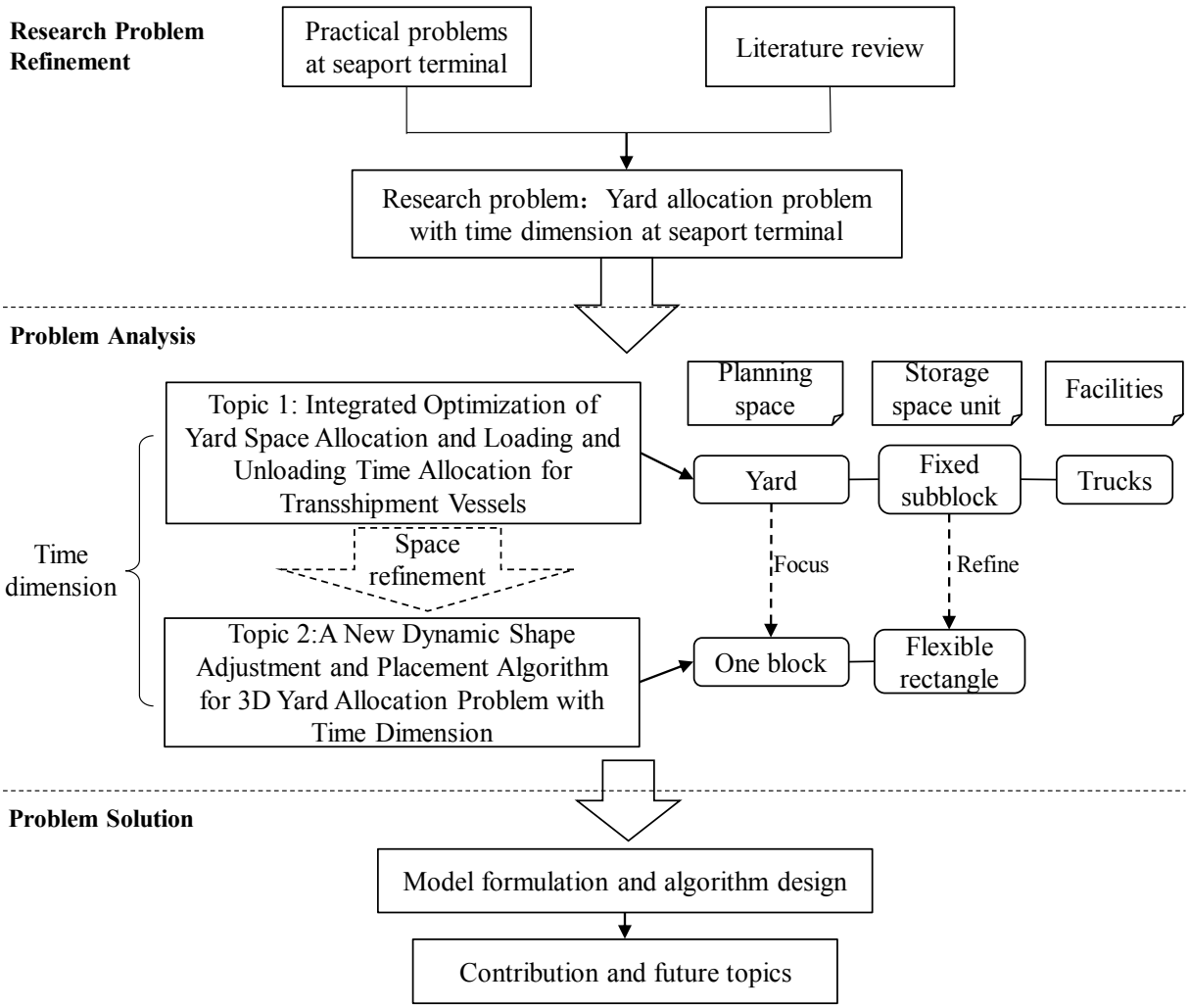


Figure 1.2: The logical outline of this thesis

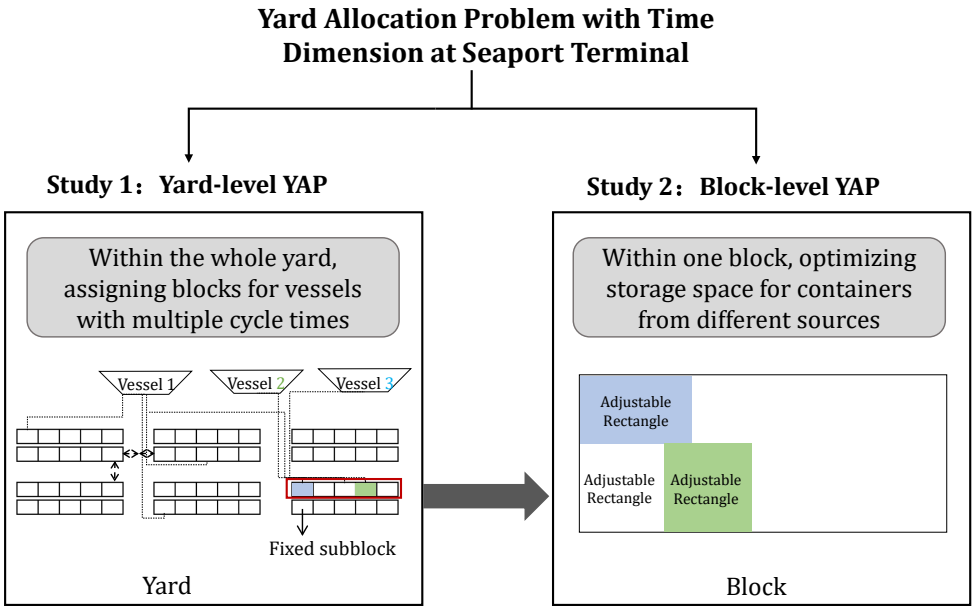


Figure 1.3: The relationship between two topics

problem.

The following are the difficulties addressed when finding solutions:

(1) Construction of the decisions and constraints of avoiding traffic congestion in the yard. In the first topic, we aim to minimize the traveling distance of trucks and the turnaround time of the vessels. These two objectives are relevant to yard congestion. A shorter route length indicates a more concentrated subblock assignment near the berth side. Such an assignment is more likely to cause congestion in the local yard area. Conversely, traffic congestion can prolong vessels' handling time. Therefore, it is challenging to construct proper decisions to avoid yard congestion while achieving the two objectives of this topic.

(2) Construction of the flexible rectangular storage space and the corresponding solution method design. We adopt the flexible rectangular storage space with variable length and width as the primary space allocation unit in the second topic. Compared with the fixed subblock used in the first topic, the rectangular storage space is more flexible. However, such flexibility enlarges the scale of the problem, making it difficult to solve. Therefore, it is challenging to define the flexible storage space and design an efficient algorithm to solve it efficiently.

1.3 Literature Review

This section summarizes the relevant literature on the container allocation problem at a seaport yard. This section includes three subsections: Subsection 1.3.1 summarizes the related research on allocating all container storage space in the whole yard. Subsection 1.3.2 summarizes the related research on the allocation of container storage space in a single block of the yard. Finally, Subsection 1.3.3 summarizes the research on the yard facility scheduling problems related to container allocation.

1.3.1 Storage space allocation in the whole yard

The storage space allocation problem usually considers two spatial ranges: the yard and a single block. This section reviews extant research covering the yard. The related research has adopted different storage spaces as the primary space allocation units, including subblock, bay cluster, and flexible subblock.

Subblock is the most popular space allocation unit (Han et al., 2008) in existing studies. By equally dividing an entire block into several fixed size storage units, we get the subblock. A subblock contains several adjacent bays. Han et al. (2008) studied space allocation for transshipment containers with the subblock as the basic unit. To balance the traffic flows in the yard, they determined the workload allocated to each subblock. Zhen et al. (2011) studied the joint allocation of yard and berth with the subblock as the basic unit. Wang et al. (2018) studied the integrated optimization model of berth allocation, quay crane scheduling, and yard storage space allocation. They proposed a heuristic algorithm based on column generation to solve the problem. Since a subblock represents a continuous and fixed storage area, it was typically combined with the consignment strategy (Carlo et al., 2014; He et al., 2020; Jiang et al., 2014, 2012b; Yu et al., 2017). According to the consignment strategy, containers meant for the same destination are stored in continuous storage space.

In recent years, researchers have paid attention to the improvement of yard efficiency. These studies include designing more efficient space allocation units and balancing the workload of multiple facilities in the yard.

(1) Flexible yard space allocation units: Park et al. (2011) designed a dynamic adjustment algorithm based on an online container stacking strategy. Computational experiments have shown that real-time monitoring and adjusting the storage space has a noticeable effect on improving the space utilization rate of the yard. Zhen et al. (2011) and Zhen (2014) studied the overall storage space allocation of the yard under uncertain conditions. They pointed out that flexible subblocks can better adapt to the uncertainty than fixed subblocks of the same size. The research showed that although the size-fixed

subblock could make it easier to solve the problem, it is also necessary to design more flexible storage units to improve space efficiency further.

Flexible subblock as primary storage unit: Jiang et al. (2012a) considered a space allocation strategy in which containers from two storage requirements share the space in the one subblock. Their numerical experiments showed that the shared subblock strategy could accommodate more containers than the fixed subblock strategy. Jiang et al. (2012b) proposed a new subblock sharing strategy and designed an algorithm to determine the shared storage space. The experimental results showed that the storage efficiency of the adjacent subblock could be improved compared with the fixed subblock. Tan and He (2016) studied the space allocation under the shared subblock strategy and the yard crane scheduling problem under the shared yard crane strategy. They found a balance between the yard space utilization and the cost of operating the yard cranes. The numerical experiments of Petering et al. (2016) showed that the yard efficiency of the subblock as the primary storage unit was not high. Tan et al. (2017) further studied the decision-making problem of flexible subblock based on the above research. In their study, a flexible subblock represented a continuous storage area composed of one or more bays. Because the size of each subblock in the same block was different, they also determined the number of yard cranes serving different subblocks to ensure operational efficiency. He and Tan (2018) studied the allocation problem of the flexible subblock of the yard when the number of containers to be stored was uncertain.

Bay cluster as primary storage unit: Ng et al. (2010) studied the location and number of bay clusters allocated to different vessels, knowing that vessels visit ports periodically. They balanced the workload of the yard at different times by deciding the variable sub-blocks. Tao and Lee (2015) studied the yard allocation problem of transshipment vessels. They took adjacent or otherwise bays in the same block as a bay cluster. Each bay cluster determined the specific allocation of different containers in different yard blocks. They built a MIP model and proposed a three-stage heuristic algorithm to solve the problem. Hu et al. (2019) studied the container storage space allocation and truck scheduling problem at the operational level with the bay cluster as the primary storage unit. They

minimized the traveling distance of the trucks and the time required for serving all containers.

In summary, size-fixed storage allocation units, such as subblock, bay cluster, or combination of fixed storage spaces, can reduce the difficulty of solving the storage space allocation problem while avoiding reshuffling. However, fixed basic storage units inevitably result in waste of storage space in the yard and costs caused by scheduling of related facilities. Therefore, by using the flexible storage space as the primary storage space unit, we can improve the yard space utilization and reduce the cost of operations in the yard.

(2) Workload balance of multiple facilities in the yard. Ng et al. (2010) studied the locations and number of bay clusters allocated to vessels while considering vessels' time periods. They determined the flexible subblocks that could help balance the workload of the yard in different periods. Won et al. (2011) took the block as the primary storage space unit and considered the balance of the block capacity, the yard crane workload, and the truck workload in the process of container space allocation. Bazzazi et al. (2009); Sharif and Huynh (2013) studied the problem of balancing the load between different blocks via storage space allocation in the yard. Li and Yip (2013) investigated the effects of different storage strategies on the yard's overall workload balance and container handling fluency based on export containers. They put forward a two-stage algorithm and the numerical experiments showed that the cross-block allocation strategy could improve yard efficiency. Yu and Qi (2013) studied two storage strategies for import containers at an automated port. Numerical experiments showed that the strategy of reserving storage space results in a higher waiting time than reshuffling. Martin Alcalde et al. (2015) forecasted the container storage capacity of the yard in a period. Besides, they put forward the best space utilization of the yard with the prediction of traffic congestion, which laid the groundwork for the subsequent space allocation of the yard. Zhen (2016) studied the joint space allocation of all blocks with subblock as the primary storage unit considering traffic congestion. Liu et al. (2016) studied a multi-objective problem of space allocation for export containers under uncertainty. Subblock was the basic storage allocation unit in this study. They minimized the operating cost of the yard while maximizing the system's

robustness. A two-stage heuristic approach was designed to solve the problem.

Optimization of berth-side problems is closely related to the decisions of the yard-side optimization problems. Besides, the yard is a critical resource relevant to other resources at a port. Therefore, the yard space allocation is usually optimized with the scheduling of other resources. Robenek et al. (2014) designed a branch and price algorithm to solve the joint optimization problem of yard spaces and berth locations. Jin et al. (2015) studied the integrated optimization problem of berth allocation and yard subblock allocation and designed a heuristic algorithm based on column generation to solve the problem. Luo et al. (2016) studied the optimization of storage space allocation and AGV scheduling at an automated port. Liu (2020) studied the multi-objective optimization problem of block allocation and berth allocation in the yard. They minimized the deviation between vessels' actual handling time and the expected handling time, and the total transportation distance of all containers. Ma et al. (2017) studied berth and yard allocation with a multi-segment irregular berth. Jiang and Jin (2017) studied space allocation for transshipment containers and the scheduling of yard crane serving different subblocks. They designed a branch and price algorithm to solve the problem. Zhou et al. (2018) studied storage space allocation in a new hybrid grid structure.

1.3.2 Storage space allocation in one block

After determining the original sources of containers placed in each block, the optimization problem in the block is to determine the specific location of the containers in the block.

It is worth mentioning that the 3DYAPT addressed in this thesis has some similarities with the well-known bin packing problem, though there are obvious differences. The three-dimensional bin packing problem (3DBPP) (Martello et al., 2000; Silva et al., 2019) can be regarded as a special case of the 3DYAPT, where the shape of the area for satisfying the space requirements is given, and the requirement of non-decrease of space over time is relaxed. While the 3DYAPT deals with container allocation based on adjustable storage

space shapes (items), in a general 3D packing, to the best of our knowledge, the shapes of the items are not adjustable. On the other hand, both the 3DBPP and 3DYAPT have the objective of finding a compact layout of items. The ideas for solving the 3DBPP can be used for solving the 3DYAPT too, such as the extreme point method (Wu et al., 2010) and the skyline method (Wei et al., 2011). However, because 3DYAPT incorporates a time dimension that makes the item shapes irregular and adjustable, this special property makes it impossible to apply methods used in 3DBPP directly to 3DYAPT. Therefore, it is necessary to design a new solution approach to cope with this special property of the 3DYAPT. The two-dimensional bin packing problem (2DBPP) is another special case with one dimension not considered. Many algorithms have been designed for the 2DBPP, but they cannot be directly applied to solve the 3DYAPT, mainly due to the changeable and non-decreasing space requirement. Moreover, a generalized case of 2DBPP is called the Packing Adjustable Rectangle (PAR) problem or Packing Problems with Soft Rectangles (PRSR), in which the length and width of each rectangle can be changed based on certain reshaping rules (Ibaraki and Nakamura, 2006; Ji and Jeng, 1990). An exact algorithm and a hybrid of local search and mathematical programming methods have been developed only for small 2D instances. We find it hard to adapt them to the 3D case, especially to a typical 3DYAPT instance containing hundreds or thousands of containers.

Chen et al. (2002) studied a yard allocation problem with time constraints (2DYAPT) that minimizes the utilized yard length. Each request in the 2DYAPT contains two dimensions, length and time. Similar to the 3DYAPT, 2DYAPT also requires the yard length not to decrease with time and there should be no overlap in yard length of different requests. Several metaheuristics, such as simulated annealing (SA), tabu search (TS), squeaky wheel optimization (SWO) and genetic algorithm (GA), are provided to obtain a near-optimal solution. Lim and Xu (2006) improved the solutions of the 2DYAPT by designing an effective critical shaking neighborhood search (CSNS) algorithm that picks and changes the order of a critical request in the sequence. Fu et al. (2007) extended the 2DYAPT to the 3DYAPT, which considers both the length and the width of the yard as well as a time dimension. They argued that considering square-shaped storage spaces for

containers is effective and a bottom-left strategy is adopted to allocate spaces to requests. However, storage spaces generated by the bottom-left strategy are not always compact as they often lead to sparse unused space that can hardly be used for other requests. The square-shaped storage space is also an intuitive and simplified assumption that has an obvious limitation on space utilization in practice.

1.3.3 Facilities scheduling related to yard storage space allocation

In the storage space allocation problem, a majority of existing literature has considered scheduling of yard facilities. The yard facilities mainly include the yard trucks and the yard crane. Next, we introduce researches related to these facilities.

(1) Scheduling of trucks: The existing research has investigated the scheduling problem of yard trucks at the operational level. At the operational level, they aimed to minimize the makespan of all storage requests. However, due to the mutual influence of scheduling of various yard facilities, the scheduling of the yard trucks was often optimized jointly with its pre-sequence or follow-up facilities or its service provider.

Joint optimization of trucks utilization and yard space: Zhang et al. (2003) studied the space allocation problem of the yard and proposed a two-stage method. In the first stage, workloads of different blocks were balanced. In the second stage, they determined the number of containers allocated to each block from different vessels. This study aimed to minimize the total transportation distance of all containers between the vessels and the yard. Cao et al. (2008) studied container space allocation in the yard and truck scheduling at the operational level. They minimized the total waiting time of vessels. Lee et al. (2009) studied integrated optimization of the yard space and yard trucks from the operational level. They minimized the total delay time of vessels and the operating time of trucks. Niu et al. (2016) studied the joint optimization of storage space and trucks. They aimed to minimize the makespan of all tasks and designed a particle swarm optimization (PSO) algorithm and a bacterial colony optimization (BCO) algorithm to

solve the problem. Wang et al. (2017) studied the joint optimization problem of subblock allocation and truck allocation and designed three algorithms based on a tree-based search to solve the problem. Hu et al. (2019) studied the container storage space allocation and truck scheduling problem at the operational level with the bay cluster as the primary storage unit. They minimized the traveling distance of the trucks and minimized the time consumed for serving all containers. Zhou et al. (2020a) studied the joint scheduling problem of yard cranes and trucks' stop-point positions in the block. They minimized the makespan of the yard cranes for serving all containers. They proposed a two-stage heuristic to solve the problem.

Joint optimization of trucks and yard cranes: Cao et al. (2010) studied the problem of joint optimization of trucks and yard cranes at the operational level. They minimized the makespan of all requirements and used the Benders' decomposition to solve the problem. Chen et al. (2013) studied the joint optimization of yard cranes and trucks. Different vessels could utilize the same truck to reduce the no-loading route distance of this truck in this problem. They determined the truck's path to serve different vessels and proposed a three-stage heuristic algorithm to solve the problem.

Joint optimization of the trucks and quay cranes: Tang et al. (2014) studied joint optimization of the truck path and quay crane from the operational perspective. Based on the European yard layout, Kaveshgar and Huynh (2015) studied the problem of joint optimization of trucks and quay cranes at the operational level to minimize the time needed for handling of all containers.

Joint optimization of trucks, yard cranes, and quay cranes: He et al. (2015) studied the problem of joint scheduling optimization of yard cranes, quay cranes, and trucks at the operational level. They aimed to minimize delays of all requests and yard facilities' energy consumption. They used a genetic algorithm (GA) and particle swarm optimization (PSO) to solve the problem. Yang et al. (2018) studied the optimization of joint scheduling of automated port quay cranes, yard cranes, and AGVs at the operational level. They minimized the completion time of all tasks by determining the routes of the AGVs.

Joint optimization of trucks and human resources: Tadumadze et al. (2019) studied the scheduling problem of trucks at the operational level by deciding the number of workers.

To sum up, although several factors affect the scheduling of yard trucks, the relationship between yard space allocation and yard truck scheduling is the closest. Accordingly, much research has been around this topic.

(2) Yard cranes. As the yard cranes are a scarce resource in the yard, different blocks have to share the yard cranes to complete the container operation within a specified time (Ng, 2005). Besides, for each yard crane, we need to plan a sequence of blocks to be served to guarantee the shortest makespan of all containers (Li et al., 2009). Next, we summarize the literature considering the following two aspects: the scheduling problem of sharing yard cranes between different blocks in the yard and the sequencing of spaces to be served by yard cranes.

Scheduling of shared yard cranes in the yard: Zhang et al. (2002) studied the scheduling problem of cranes in the yard. By deciding the working time of each yard crane and its route across different blocks, they improved the efficiency of the shared use of yard cranes. Zhang et al. (2002) studied the dynamic utilization of cranes in the yard. They minimized the delayed workload by determining the moving times and paths of the cranes across different blocks. Ng (2005) studied the sharing of single-rail yard cranes in two adjacent blocks. They optimized the scheduling of yard cranes by considering the conflicts caused by the movement of yard cranes in different blocks. Guo and Huang (2012) optimized multiple yard cranes in multiple blocks based on the European yard layout. For each block, they considered the safe distance between two cranes. They minimized the total waiting time of all cranes. Sharif et al. (2012) used an agent-based approach regarding the yard crane and the block as different agents to solve the yard crane scheduling problem at different block intervals. They minimized the number of unfinished works. Jin et al. (2016) studied the storage space allocation problem, in which the subblock was the basic unit in the yard. They put forward five different assignments of the yard cranes and scheduled all cranes to minimize their operating costs. Jin et al. (2016) considered that a yard section contains four blocks and multiple yard cranes, and the minimum

space allocation unit was a subblock. Their study minimized the total moving cost of all yard cranes between each block by scheduling storage space and yard cranes. Chu et al. (2019) studied the scheduling problem of three shared yard cranes in the same rail in two adjacent blocks.

Yard crane scheduling in one block: According to the number of yard cranes, dispatching of cranes in one block can be divided into three types. (a) Single yard crane: only one yard crane serves the block; (b) Twin yard cranes: two yard cranes serve the block; (c) Multiple yard cranes: at least three yard cranes serve the block.

Single yard crane: Lee and Kim (2010) subdivided the operation of the yard crane into four different types. Based on the operational cycle time of the yard crane, they studied the offset of the basic cycle time of the four different operations and compared the time modes of different operations. Guo et al. (2011) studied the scheduling of a yard crane in a block. Under the condition that the arrival time of all trucks visiting the block can be predicted, they minimized the waiting time of all trucks in the yard by determining trucks' locations and the sequence for servicing of each truck. They proposed an A* algorithm to solve the problem. Chang et al. (2011) proposed a dynamic rolling time-domain strategy for schedule optimization decision of the yard crane in the block. Gharehgozli et al. (2014) studied the path scheduling problem of one yard crane in a block to deal with multiple requirements of storage and withdrawal of containers. They minimized the total moving time of the yard cranes for completing all requirements. Galle et al. (2018) studied the scheduling problem of a yard crane in a block. They considered the storage, withdrawal, and reshuffling operations simultaneously based on the Asian two-side truck stop layout. Gharehgozli et al. (2019) sequenced all the pickup and storage operations of the single yard crane in a European layout yard block. In their study, the total traveling time of the yard cranes was minimized.

Twin yard cranes: Li et al. (2009) studied the scheduling of two yard cranes in a block. They considered three primary factors: the interference between movement of different yard cranes, safe operating distance between the yard cranes, and scheduling of each yard crane's storage and withdrawal operations. Vis and Carlo (2010) studied one large and

one small yard crane based on the European yard layout. The two yard cranes can cross each other on different rails. Gharehgozli et al. (2015) considered a block in the European yard layout, in which two yard cranes that cannot cross each other maintained a safe distance during their operations. They determined the operations sequence of all service requirements in the block to minimize the makespan. They designed an adaptive large neighborhood search algorithm (ALNS) to solve the problem. Hu et al. (2016) studied the scheduling problem of the two yard cranes in a block at Shanghai Yangshan Port. They considered the safe distance between the two cranes to avoid interference with each other. They minimized the completion time for handling all containers. Speer and Fischer (2017) compared four different configurations of the block and yard cranes. Nossack et al. (2018) studied the scheduling of two impassable yard cranes in a yard based on the European layout of the yard. They allocated each yard crane to various requirements and determined the order in which each yard crane processes these requirements. They aimed to minimize the completion time of the required operation. They used a branch and cut algorithm and Bender's decomposition method. Kress et al. (2019) studied the scheduling problem of two yard cranes in a block under the European yard layout. The two yard cranes were on the same rail. Their study aimed to minimize the berth-side processing time of containers.

Multiple yard cranes: Wu et al. (2015) considered several yard cranes in a block. These yard cranes could not cross each other and had to maintain a safe distance. This was a multi-objective problem at the operational level. The objective was to minimize the total deviation time and the total delay of yard cranes. They solved this problem by transforming the multi-objective problem into a single-objective problem. Zhen et al. (2018) studied the yard crane scheduling problem of a new port layout with a multi-layer frame quay crane. Chen et al. (2020) studied the joint scheduling problem of multi-yard cranes and multiple AGVs in a large-scale automatic port.

CHAPTER 2

INTEGRATED OPTIMIZATION OF YARD SPACE ALLOCATION AND LOADING AND UNLOADING TIME ALLOCATION FOR TRANSSHIPMENT VESSELS

This chapter focuses on yard space allocation and handling time assignment for transshipment vessels at a seaport yard. Specifically, we avoid the traffic congestion caused by container trucks in the yard by considering both the location decision and the unloading and loading time decision of each subblock. Also, we guarantee that each transshipment vessel is able to complete loading and unloading in time and does not get delayed for its subsequent voyage. To efficiently tackle this problem, we build several mixed-integer programming models and propose a critical element-based three-stage assignment heuristic algorithm.

2.1 Introduction

Efficient allocation of storage space for transshipment containers plays a crucial role in scheduling yard spaces and vessels. Vessels from the same liner visiting the port usually have the same cycle times, while vessels from different liners have a heterogeneous periodical pattern of arrival and departure. In this study, we consider the multi-period yard space allocation problem for vessels from different ship liners, in which cycle times of different vessels overlap. Such heterogeneous and periodical nature in the time dimension of various vessels further complicates this problem.

Much research efforts have been made for examining optimization problems regarding the yard and many have focused on storage space allocation, but few have considered the loading time decision and then unloading time decision for each storage space. Zhen et al.

(2011) studied the integrated optimization of allocating berth positions, quay cranes, and yard spaces for vessels with an identical period. Their objective was to minimize the total route length of trucks and the time deviation of all vessels. They used an estimated average distance to simplify the actual distance between each berth position and each reserved yard space. With such a simplification, they avoided considering the detailed loading route and unloading route for each container truck. Also, they considered only the congestions in horizontal yard lanes in their model. Zhen et al. (2016) optimized the total traveling distance of loaded container trucks by allocating proper storage spaces for transshipment vessels with multiple periods. Subblock is the primary space allocation unit. They minimized the total route length (loading/unloading) of all transshipped containers for all arriving vessels during the whole planning horizon. They considered the actual route length between a berth position and a subblock. Both the traffic congestions in horizontal yard lanes and vertical yard lanes were taken into consideration. Two decisions were made: locations of the reserved subblocks for each arriving vessel and the number of transshipped containers allocated to each reserved subblock of the arriving vessel. They proposed a local branch-based method to assign the locations and used the CPLEX solver to compute container numbers.

Based on the work of Zhen et al. (2016), this study considers the multi-period storage space optimization problem of transshipment vessels. Given the berth location and the number of reserved subblocks for each vessel, we intend to determine the locations and the relevant loading time and unloading time of the reserved subblocks. We use the same bi-objective as that in Zhen et al. (2011). We aim to maintain smooth traffic of container trucks in the yard lanes over the entire planning horizon, such that the total traveling cost of trucks is minimized, and each vessel is operated in time. This study intends to avoid traffic congestion in both the horizontal and vertical yard lanes. Different from the rough consideration of both traffic flow of the vertical lane as a whole in Zhen et al. (2016), we distinguish the unloading traffic and loading traffic as two different flows. In this study, both the location and handling time are decisions correlated to traffic congestion, while only the location is taken as a decision and the unloading time and loading time are

parameters in Zhen et al. (2016). In addition, we consider more on the location decisions. That is, except for the locations of reserved subblocks for an arriving vessel same as in Zhen et al. (2016), this study also determines the locations of subblocks for the source vessels of this arriving vessel.

In this study, the planning time is assumed to be a minimum common long period for vessels with different periods. Accordingly, the complexity of the problem increases significantly compared with the case of one period, and the difficulty of solving this problem increases exponentially. Therefore, the algorithms proposed for solving one period in Zhen et al. (2011) cannot handle this multi-period problem. Besides, the local branch method that considers only the location decisions in Zhen et al. (2016) are not adaptive to handle the two mutually related decisions in this study. It is essential to design a new and efficient solution method for this problem.

2.2 Problem Description

We consider the decisions for transshipment vessels at a seaport yard, which comprises storage space allocation and handling time assignment. At the port, transshipment of containers between different vessels refers to transferring a batch of containers from a source vessel to a destination vessel. Since the arrival times of the source and destination vessels are different, and it takes a long time to load and unload numerous containers, the transshipment containers are handled according to the following process: (1) Source vessel arrives. (2) The transshipment containers are unloaded from the source vessel and transported to several subblocks in the yard. (3) Destination vessel arrives. (4) The transshipment containers stored in the above subblocks are transported from the yard to the destination vessel, which completes the transshipment process between the two vessels.

2.2.1 Yard

As the yard is on inshore land, containers transshipped between the yard and the vessels are transported by container truck (CT). Classified by different uses, the yard area can be divided into two parts. One is the storage area for containers, which includes multiple blocks. The other is the aisle area for truck movement. The two parts are described below in detail.

Minimum storage unit: subblock

In the existing literature, the concept "Yard Template" is widely used to split storage space in the yard. Generally, the yard is divided into many area-fixed spaces named "Subblocks". Each subblock is the basic storage unit when allocating yard space to vessels. All containers in a subblock are to be transferred to the same destination vessel, which is the requirement of the "Consignment strategy". It is common that the yard divided into subblocks follows the consignment strategy. Consignment strategy is a container storage method with the following advantages: First, ship liners usually have their preferred storage space in the yard, which can be labeled in advance by a set of subblocks. Second, each subblock only includes containers to the same destination vessel, which can reduce the number of container reshuffles in a subblock. Meanwhile, the workload of yard cranes can be reduced, which means saving of the cost of waiting time and energy consumption.

Usually, there is a large number of transshipment containers, which require more than one subblock for storage. For subblocks allocated to the same destination vessel, locations in the yard do not need to be adjacent. That is, the allocated subblocks can be in any preferred space of this vessel within the yard. When there are high-frequency operations of loading and unloading of a vessel, discrete allocation of its reserved subblocks can disperse the workload of yard cranes within local areas. Accordingly, the traffic flows of yard trucks are dispersed, which reduces the potential for congestion in the yard.

Yard traffic

In the yard, traffic lanes can be divided into horizontal lanes and vertical lanes.

Horizon lanes are those parallel to the direction of the berth. The most common horizontal lanes are located between two subblocks. Due to the limited space between subblocks, the number of trucks simultaneously passing a horizontal lane is also limited. Typically, only one container truck is allowed the go through side by side at the same time. Besides, to avoid conflicts of the movement of trucks, it is stipulated that trucks can only move in the same direction in each horizontal lane. As shown in Figure 2.1, the green arrows show the direction of traffic flow between two subblocks. In addition, there is a kind of particular horizontal lane between the berth and the yard. Because the space along the berth side is more abundant than that between subblocks and the traffic is more complex near the berth, this horizontal lane has traffic flows in both directions. The yellow arrow in Figure 2.1 shows this kind of particular horizontal lane.

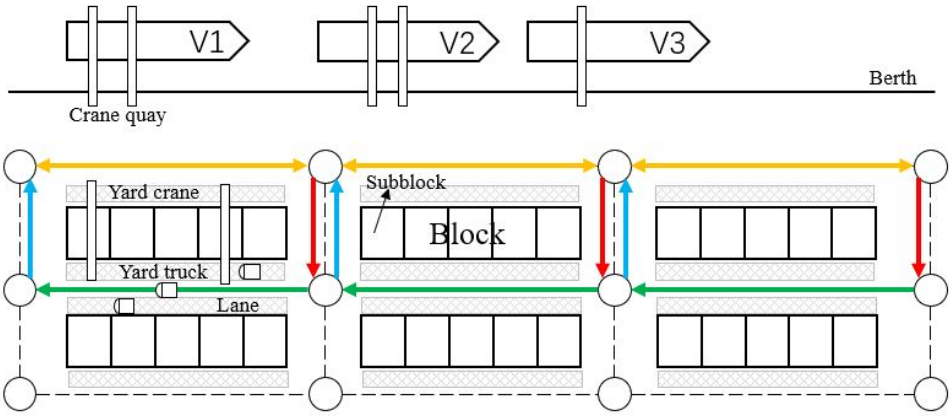


Figure 2.1: Directions of traffic lanes

Vertical lanes are those perpendicular to the berth's direction. The typical vertical lanes are located between the short edges of two adjacent subblocks. Because yard cranes in each subblock do not occupy the space of the vertical lanes and vertical lanes need to load the bi-directional traffic flows between the berth and the yard, vertical lanes are more expansive than horizontal lanes. Thus, each vertical lane comprises a loading direction lane and an unloading direction lane. Also, the lane in each direction can accommodate multiple traffic flows simultaneously. As shown in Figure 2.1, in each vertical channel, the red arrow shows the direction of traffic flow from the berth to the yard, and the blue arrow shows the direction of traffic flow from the yard to the berth.

In this chapter, "traffic flow" is defined as the circuit of a truck traveling from a starting point through a distance and then back to the starting point. In detail, each traffic flow can contain several trucks which share the same starting point and route. Meanwhile, different trucks in the same traffic flow can travel only in the front and back sequence in the lane. In this problem, we deem that different starting points lead to different traffic flows. Besides, each traffic flow occupies a traveling position in the lane. Therefore, the "maximum number of traffic flows of a lane" is defined as the maximum number of trucks allowed to travel side by side simultaneously in the lane. For a horizontal lane, its maximum number of traffic flows is 1. For a vertical lane, we define u_{bk}^{max} as its maximum number of traffic flows from the berth to the yard, and u_{kb}^{max} as the maximum number of traffic flows from the yard to the berth.

2.2.2 Vessel

Vessel's period

Vessels from the same ship liner usually visit the port within a fixed period, while vessels of different ship liners visit the port during different periods. For the convenience of description in the rest of this chapter, we treat a vessel as a ship liner, since vessels from the same liners have the same period. Usually, vessels visit the port every week, ten days, or two weeks, of which one week is the most common period. In this chapter, we separate the entire planning horizon into a series of equal time units. Then, the period of each vessel is composed of a set of time units. Accordingly, different periods of the same vessel contain the same number of time units. Different vessels usually have different periods. Thus, the number of time units of the periods of different vessels are also different.

For simple illustration in this section, we use V to denote a vessel, P to denote a period. Besides, their combination, i.e., pair $V-P$ represents vessel V at its period p. As shown in Figure 2.2, there are three vessels, $V1$, $V2$, and $V3$, during the entire planning horizon. Their periods are 4 days, 5 days, and 10 days respectively. In this example, each time unit equals one day. Thus, the periods of $V1$, $V2$, and $V3$ contain 4, 5, and 10 time

units respectively.

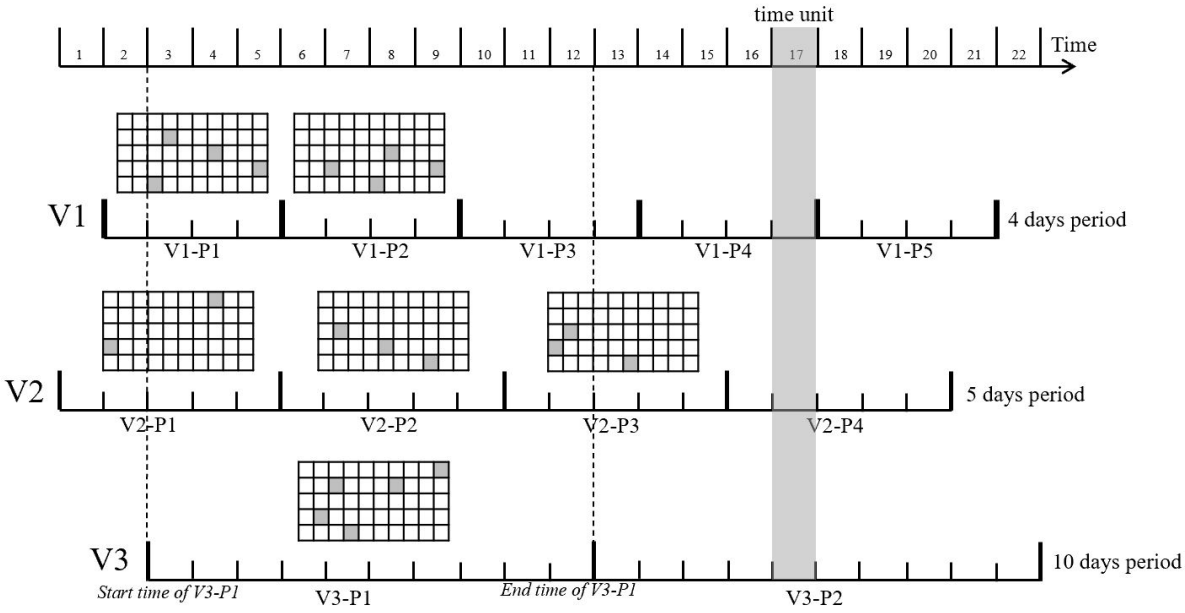


Figure 2.2: Illustrations of the allocation of subblocks in a period

There are three conditions for assigning subblocks to vessels in each period. (1) When a subblock is assigned to vessel V at period p, this subblock remains unchanged at every time unit in period p. (2) For a vessel, the assignment of subblocks varies according to different periods. The reason is that the numbers of transshipment containers at different periods are different. Besides, plans of other vessels can affect the current vessel. (3) For a given time unit, the assigned subblocks for different vessels at the same time should not conflict. As shown in Figure 2.2, period $P1$ of vessel $V1$ contains time units 2 to 6. Four subblocks are reserved for $V1 - P1$. These four subblocks are assigned to $V1 - P1$ from time 2 until time 6. Vessel $V2$ contains two periods $P1$ and $P2$. The number of subblocks reserved for $P1$ and $P2$ is different, i.e., 2 and 3, respectively. Also, the locations of the subblocks for P1 and P2 are different. Time unit 8 corresponds to three vessels' periods, namely, $V1 - P2$, $V2 - P2$, and $V3 - P1$. As can be seen from Figure 2.2, they are not assigned the same subblock.

Decisions of vessels

Due to the limitation of storage space in a vessel, when vessel i arrives, unloading of containers should precede loading of containers. The transshipment containers in vessel i

are first unloaded to the yard before containers in the yard are loaded onto vessel i. Then, vessel i leaves the port after all containers transshipped to it are loaded. To describe the decisions of vessel i, we first define three parameter sets as follows. (1) V_{ip}^D represents the set of destination vessels of vessel i. For vessel i at period p, the transshipment containers are transported to vessels in set V_{ip}^D. (2) V_{ip}^S represents the set of source vessels of vessel i. For vessel i at period p, all its transshipment containers come from vessels in set V_{ip}^S. (3) K_{ip} represents the subblocks reserved for containers unloaded to vessel i. In other words, subblocks of set K_{ip} store the containers for vessels in set V_{ip}^S. The number of subblocks in set K_{ip} is a known parameter, denoted by n_{ip}. Before the arrival of vessel i at period p, the ship liner informs the port manager of the number n_{ip}. For example, in Figure 2.3, vessel $V3$ and period $p1$ comprise pair $V3-P1$. Here, $V3$ is vessel i as mentioned above, and P1 is period p. As shown in Figure 2.3, vessel $V3-P1$ arrives at $t=7$. Then, its destination vessels are in set $V_{V3-P1}^D = \{V1-P2, V2-P2\}$. Similarly, the source vessels of pair $V3-P1$ are in set $V_{V3-P1}^S = \{V1-P1, V2-P1\}$. Given that $n_{3,1}=5$, then set $K_{3,1}$ is composed of the five assigned subblocks as shown in the figure.

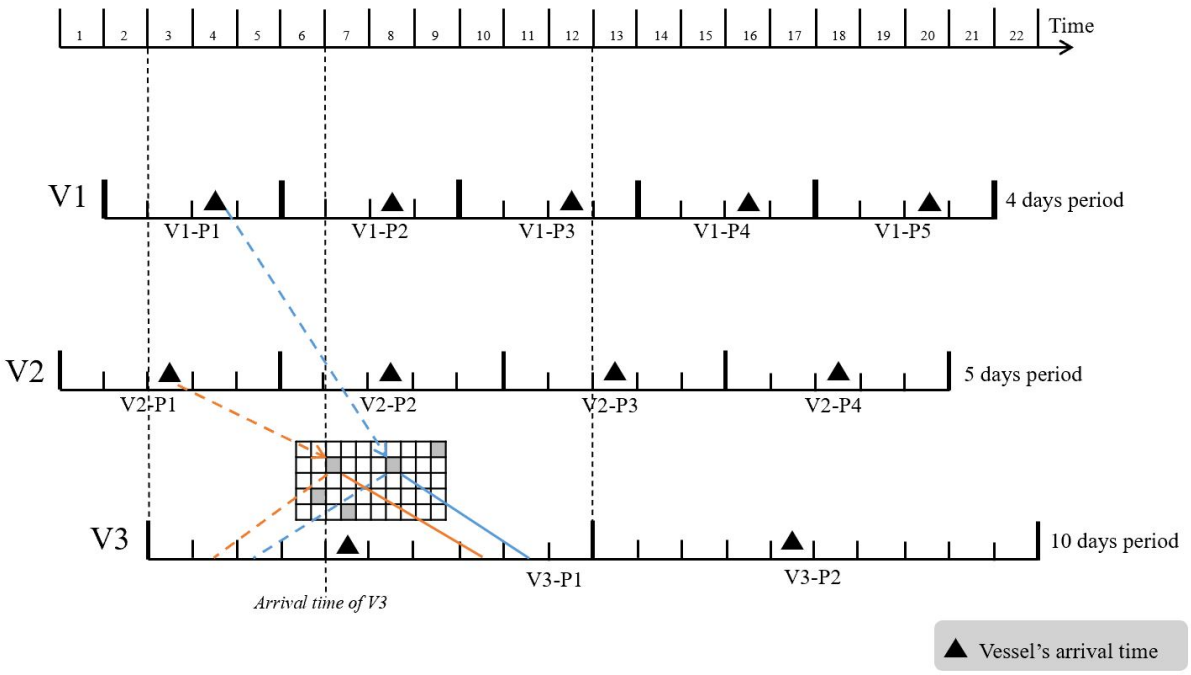


Figure 2.3: Illustrations of the decisions of a vessel

For vessel i at period p, the port manager has to make the following four types of

decisions.

Type 1 decision: the location of each subblock k ($k \in K_{ip}$).

Type 2 decision: the location of each subblock k ($k \in K_{ip}$) corresponding to each source vessel j ($j \in V_{ip}^S$).

Type 3 decision: the unloading time for source vessel j ($j \in V_{ip}^S$) to unload containers to subblock k ($k \in K_{ip}$).

Type 4 decision: the loading time for vessel i to load containers from each subblock k ($k \in K_{ip}$).

To illustrate these decisions, let us take the vessel-period pair $V3-P1$ in Figure 2.3 as an example. In Figure 2.3, the dotted lines represent the unloading decisions, while the solid lines represent the loading decisions. It is known in advance that $n_{3,1} = 5$. Therefore, we are required to determine the locations of five subblocks (marked in dark in the figure) from the preferred storage area of $V3-P1$, which is the type 1 decision. The five subblocks store containers from source vessels $V1-P1$ and $V2-P1$. Thus, these five subblocks also need to be assigned to $V1-P1$ and $V2-P1$, which is the type 2 decision. As shown in Figure 2.3, $V1-P1$ unloads to the two subblocks marked by the blue dotted lines, and $V2-P1$ unloads to the three subblocks marked by the yellow dotted lines. With the above two types of decisions, we can make decisions corresponding to locations. Next, we consider the handling time (loading time and unloading time) decisions of the five subblocks. As shown in Figure 2.3, the unloading time of $V1-P1$ is $t = 5$ as marked by the blue dotted line, and the unloading time of $V2-P1$ is $t = 4$ as marked by the yellow dotted line, which is the type 3 decision. Finally, the loading time decisions need to determine the time required for loading containers from the five subblocks to $V3-P1$. In Figure 2.3, the loading times of two illustrative subblocks marked by the blue and yellow solid lines are $t = 11$ and $t = 10$.

2.2.3 Objective

Location of each subblock needs to be determined for its loading and unloading process related to each vessel. Meanwhile, the loading time and unloading time of each subblock are also to be determined. In this chapter, the number of required subblocks for both the loading and unloading process are parameters. Also, the berth position and the locations of the quay cranes handling each vessel are known. With the decisions of storage locations and handling time, we aim to minimize the traveling distance of all trucks while avoiding traffic congestion in the yard. Besides, the total time deviation of handling time of all vessels over the entire planning horizon is also to be minimized.

First, we consider the first part of the objective, that is, to minimize the total traveling distance of trucks. In the transshipment process, one batch of containers involves two operations: unloading operation from the vessel to its storage subblock and the loading operation from the subblock to its destination vessel. Therefore, the traveling distance of yard trucks includes both the unloading distance and the loading distance. In Figure 2.4, the green-colored circle represents a complete unloading route, where the solid line represents that the trucks are going to be loaded and the dotted line represents that the trucks are being unloaded. Similarly, the red-colored circle represents a complete loading route. During the transshipment process, both the loading and the unloading process correspond to transferring a large number of containers. Primarily, we assume several trucks serve the unloading and loading demand between a vessel and a subblock, ensuring that the unloading or loading process can be complete in a short time. Accordingly, the waiting time of the vessels is reduced. Therefore, we assume that each truck can only serve one vessel within one circuit route in this problem. Besides, we consider the truck that can carry only one container at a time. Thus, the dotted distance that the truck travels without carrying a container is essential to serve a complete unloading/loading route. Based on the above reasons, we use the length of a circuit route as the distance of unloading/loading between a vessel and a subblock.

Then, we consider the second part of the objective, that is, to minimize the total time

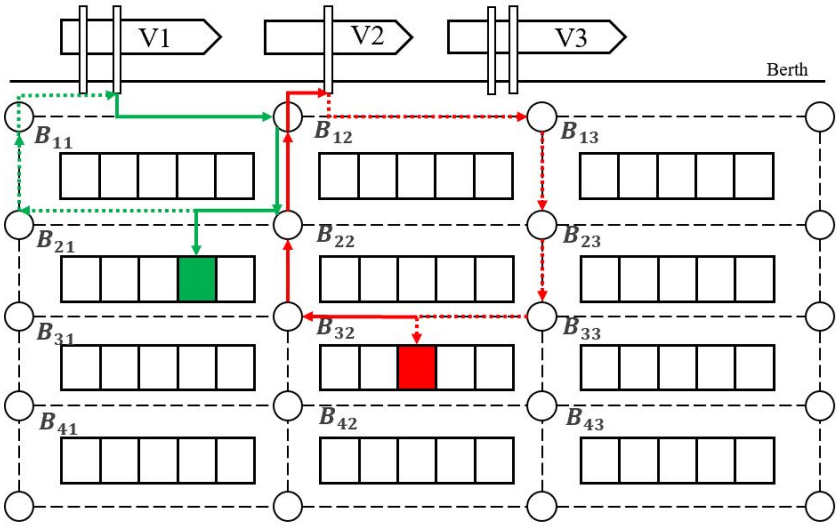


Figure 2.4: Illustrations of the unloading route and loading route

spent by all vessels over the entire planning horizon. For vessel i in period p, we use notation $t_{ip}^{F,start}$ to represent its arrival time, which is a known parameter. As shown in Figure 2.5, vessel $V3$ arrives at $t = 7$ in the first period, that is, $t_{3,1}^{F,start} = 7$. Besides, each vessel is expected to be handled within a given time interval, namely, the feasible time interval. If a vessel can complete the unloading and loading process within its feasible time interval, it will travel according to its plan in the following voyage. Otherwise, this vessel needs to speed up, slow down, or wait not to influence its schedules at different ports. For vessel i in period p, we define its expected time interval as $[\alpha_{ip}^{start}, \beta_{ip}^{end}]$, and its feasible time interval as $[t_{ip}^{F,start}, t_{ip}^{F,end}]$. As shown in Figure 2.5, the expected time interval is $[8, 11]$ and the feasible time interval is $[7, 12]$. The unloading and loading operations of the transshipment containers do not need to be completed within the expected time interval but need to be within the feasible time interval. It is acceptable to handle containers at time units beyond the expected time interval. The handling time deviation is the period of time by which the actual time exceeds the expected time interval. However, operations during the time deviation lead to extra costs. Therefore, we intend to minimize the total time deviation of all vessels during the planning horizon.

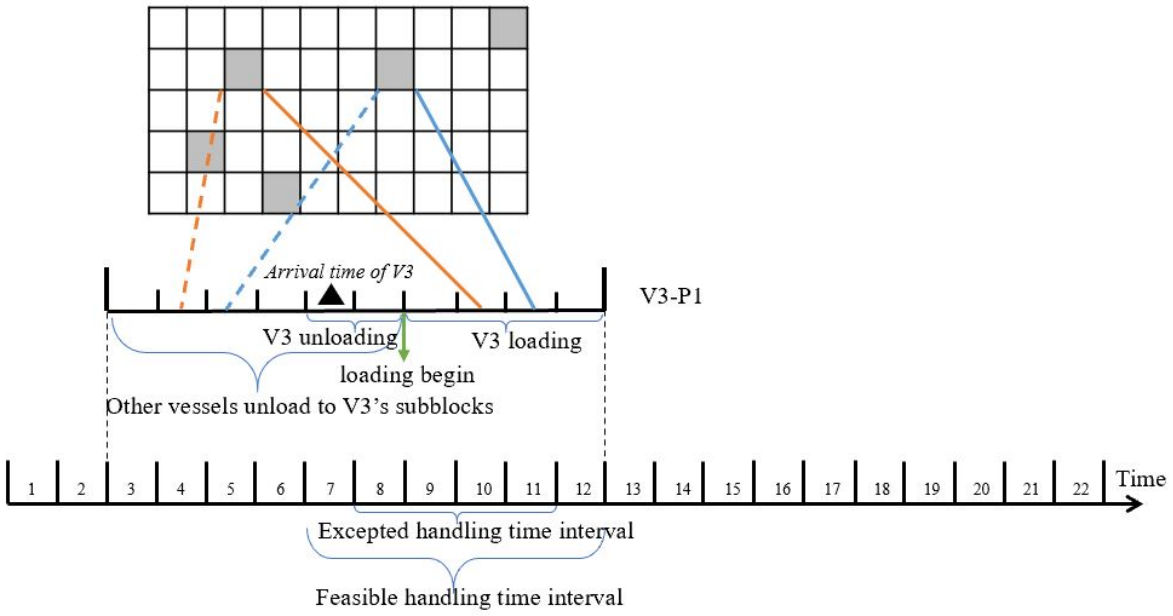


Figure 2.5: Illustrations of the handling time of a vessel

2.2.4 Constraints on subblock assignment

In this problem, there are three types of constraints on the assignment of subblocks. That is, the vessel-related constraints, the same-block constraints, and the adjacent-lane constraints. Next, we introduce these constraints one by one.

Vessel-related subblock allocation constraints

First, it is necessary to ensure that there is no conflict. That is, each subblock can only be assigned to one vessel during a time unit.

Second, as described in Subsection 2.2.2, the assigned subblock to vessel i should remain unchanged at every time unit t of period p.

Third, for vessel i at period p, the assigned subblocks should be selected from the preferred yard area. Let notation K_i denote the set of preferred subblocks of vessel i. That is, each assigned subblock k of vessel i satisfies $k \in K_i$.

Fourth, vessel i at period p should be assigned a given number of n_{ip} subblocks. Here, n_{ip} is a known parameter.

Constraints on subblocks in the same block

The loading of a vessel is usually carried out in a short time. Therefore, there are frequent container operations near the assigned subblocks. These operations include the retrieval of containers by yard cranes and transportation of containers by yard trucks.

If two subblocks in the same block are assigned to the same vessel i, the two traffic flows in the horizontal lane of this block may lead to traffic congestion. Conflict 1 in Figure 2.6 shows this kind of congestion. Besides, each block is usually equipped with two yard cranes. Each subblock is served by only one yard crane. If two subblocks in the same block are assigned to vessel i simultaneously, the yard cranes will be very busy loading containers to vessel i. If a subblock in this block is assigned to another vessel j, and vessel j has loading or unloading requirements when vessel i is loading, no crane is available for vessel j. Thus, the operation time of vessel j can be affected by the assignment of vessel i. Based on the above two reasons, each vessel can be assigned with at the most one subblock from the same block.

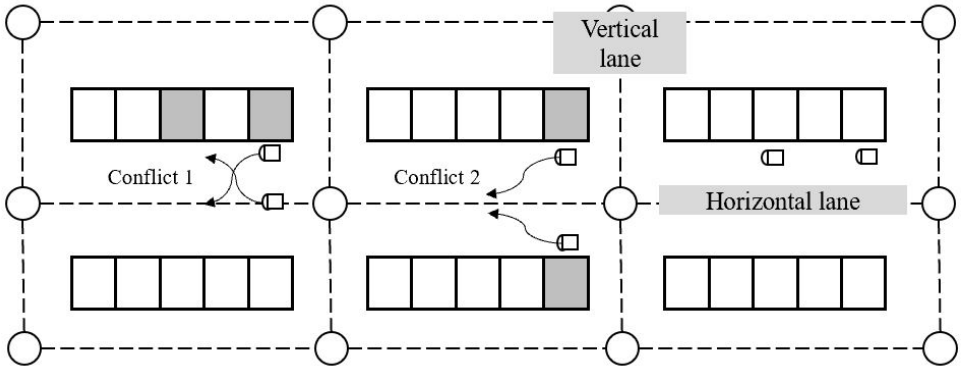


Figure 2.6: Two types of conflicts in the horizontal lane

Constraints on subblocks near same horizontal lane

This problem's primary storage allocation unit is the subblock, in which 240 containers can be stored. Therefore, to make sure each vessel leaves the port on time, we need to complete the loading operation in a short time. Meanwhile, the unloading operation, which is ahead of the loading operation, also needs to be completed quickly. The heavy workload during the unloading or loading process causes an increase in number of trucks near the related subblocks. As mentioned in Subsection 2.2.1, the horizontal lane between

two blocks allows no more than one traffic flow at a time. Thus, if two subblocks along the same horizontal lane are loading or unloading simultaneously, there will be traffic congestion in this lane. Conflict 2 in Figure 2.6 shows such congestion. Trucks have to wait to avoid conflict in the horizontal lane. Therefore, for vessels' operational efficiency, subblocks from two blocks sharing the same horizontal lane cannot be assigned to a vessel simultaneously.

2.2.5 Constraints on handling time assignment

For vessel i at period p, its handling time includes two parts. (1) the time that vessel i takes to unload containers to each assigned subblock; (2) the loading time that vessel i takes to load containers from each assigned subblock. The assignment of unloading time and loading time should satisfy the following constraints.

First, the unloading time and loading time of vessel i should not be earlier than the arrival of vessel i. As shown in Figure 2.5, vessel $V3$'s unloading time interval is $[7, 8]$, and its loading time interval is $[9, 12]$, both of which are within its feasible time interval $[7, 12]$. Besides, both of them are not earlier than the arrival time $t = 7$.

Second, due to the stowage scheme of vessel i, its unloading time should be ahead of its loading time. As shown in Figure 2.5, vessel $V3$'s unloading time interval is $[7, 8]$, and its loading time interval is $[9, 12]$. Obviously, its latest unloading time $t = 8$ is ahead of its earliest loading time $t = 9$.

Third, when containers are transshipped from vessel j to vessel i via subblock k, the unloading time from vessel j to subblock k should be ahead of the loading time from subblock k to vessel i. In this problem, the loading start time of vessel i is a decision variable, namely $\tau_{ip}^{L,start}$, which equals to the minimum loading time of all containers assigned to the subblock. For example, as shown in Figure 2.5, the loading start time of vessel $V3$ at period $P1$ is the time unit marked by the green-colored arrow, i.e., $\tau_{3,1}^{L,start} = 9$. Accordingly, the unloading time interval for vessel j to transship to vessel i is $[t_{3,1}^{F,start}, \tau_{3,1}^{L,start} - 1]$, that is $[4, 8]$.

2.2.6 Constraints on the maximum traffic flow

In Subsection 2.2.4, traffic congestion in the horizontal lane is constrained by the subblock assignment constraints. However, we need additional constraints to control traffic congestion in the vertical lane. Since the maximum traffic flow exists in both (unloading and loading) directions of a vertical lane, the number of traffic flows passing a lane from either direction cannot exceed its maximum traffic flow.

Even if the subblock assignment constraints in Subsection 2.2.4 are satisfied, traffic flows in the vertical lanes may still exceed the maximum number, which results in traffic congestion. This congestion corresponds to three types of decisions in the yard. First, if the subblocks using the same vertical lane are allocated simultaneously, congestion occurs in this vertical lane. Second, given that the locations of the assigned subblocks are balanced, the simultaneous loading and unloading of some subblocks will still lead to traffic congestion. Third, if both the assignation of the locations and the handling times of subblocks are not balanced, congestion will happen. Therefore, we need to decide both the location and the handling time of each assigned subblock to limit the traffic flow in the vertical lane. In this way, traffic congestion can be avoided. We add such traffic flow constraints on both the loading and unloading direction of each vertical in this problem.

2.2.7 Constraints on containers distribution

Before introducing the decisions and constraints, we first define the following notations.

Notations of parameters: (1) K_i is the set of preferred subblocks of vessel i. (2) n_{ip} is the number of subblocks required to be reserved for vessel i in period p. (3) n_{jip} is number of subblocks reserved for vessel j which is the source vessel of vessel i in period p. (4) q_{jip} is the number of transshipment containers from vessel j to vessel i in period p. Notations of decisions: (1) K_{ip} is the set of assigned subblocks for vessel i in period p. Obviously, we have $K_{ip} \subset K_i$ and $|K_{ip}| = n_{ip}$. (2) K_{jip} is the set of assigned subblocks for the source vessel j of vessel i in period p. Obviously, we have $K_{jip} \subset K_{ip}$ and $|K_{jip}| = n_{jip}$. (3) z_{jikp} is the number of containers transshipped from vessel j to vessel i via subblock k.

z_{jikp} needs to meet four constraints: (1) for vessel i in period p, containers transshipped from source vessel j to vessel i can be stored only at the subblocks assigned to vessel i. In other words, if a subblock is not assigned to vessel i, it cannot be used to store containers from source vessel j. That is, if $k \notin K_{ip}$, $z_{jikp} = 0$. (2) When q_{jip} containers are transshipped from source vessel j to vessel i in period p, the number of containers allocated to each assigned subblock k ($k \in K_{jip}$) cannot exceed the total transshipment volume, that is, $z_{jikp} \leq q_{jip}$. If subblock k is assigned to vessel i but not assigned to source vessel j, no containers from vessel j will be stored in subblock k. That is, if $k \in K_{ip}$ and $k \notin K_{jip}$, $z_{jikp} = 0$. (3) Containers transshipped from vessel j to vessel i in period p should be all stored in its subblocks. That is, $\sum_{k \in K_{jip}} z_{jikp} = n_{jip}$. (4) Subblocks assigned to source vessel j cannot store containers from other source vessels of vessel i, i.e., $\cap_{j \in V_{ip}^S} K_{jip} = \emptyset$.

2.3 Mathematical Formulation

In this section, we build a mixed-integer programming (MIP) model for this problem. As mentioned in the problem description, three decisions need to be taken. (1) Locations of the assigned subblocks. (2) Number of containers allocated to each assigned subblock. (3) Unloading and loading time of containers in each subblock assigned to each vessel. This problem aims to minimize the traveling distance of trucks transporting all transshipment containers within the planning horizon through the above decisions. Meanwhile, this problem aims to minimize the total time deviation between the actual handling time and the expected handling time of all vessels.

2.3.1 Notations

This section shows all mathematical notations used in the problem description and the mathematical model. First, we list the descriptions of some notations.

Indices:

i, j: vessel;

p: period;

t: time unit;

k: subblock;

u: vertical lane.

Sets:

V: set of all vessels;

K: set of all subblocks;

T: set of all time units in planning horizon;

U: set of all vertical lanes.

Parameters:

K_i: set of the preferred subblocks of vessel i;

T_i: set of all periods of vessel i;

$|T_i|$: the number of periods of vessel i;

T_{ip}: set of time units of vessel i in period p;

$|T_{ip}|$: the number of time units of vessel i in period p;

V_{ip}^S: set of source vessels of vessel i in period p;

V_{ip}^D: set of destination vessels of vessel i in period p;

n_{ip}: the number of subblocks reserved for vessel i in period p;

n_{jip}: for vessel i in period p, the number of subblocks reserved for its source vessel j;

q_{jip}: the number of containers transferred from source vessel j to vessel i in period p;

N_b: set of subblocks in block b;

$\mathbb{N}_b$: set of N_b in the yard;

N_h: set of neighboring subblocks along horizontal lane h;

$\mathbb{N}_h$: set of N_h in the yard;

π_{ipku}: equals 1 when the route between vessel i in period p and subblock k passes lane u;

w_u^U: the maximum number of unloading traffic flows of vertical lane u;

w_u^L: the maximum number of loading traffic flows of vertical lane u;

C_k: the capacity of a subblock;

M: a very large positive integer;

d^U_{ipk}: the unloading route length from vessel i to subblock k in period p;

d^L_{ipk}: the loading route length from subblock k to vessel i in period p;

η_1: the coefficient of traveling route in the objective;

η_2: the coefficient of time deviation in the objective.

Decision variables:

x_{tki}: set to one if subblock k is assigned to vessel i at time unit t, and zero otherwise;

$\bar{x}_{pki}$: set to one if subblock k is assigned to vessel i in period p, and zero otherwise;

$\bar{y}_{pkij}$: set to one if subblock k is assigned to source vessel j of vessel i in period p, and zero otherwise;

z_{ijkp}: the number of container transferred from vessel i to vessel j via subblock k in period p;

λ^U_{ijkt}: set to one if containers transferred from vessel i to vessel j via subblock k are loading at time unit t;

δ^L_{it}: set to one if vessel i are loading at time unit t;

$\tau^{L,start}_{ip}$: the loading start time of vessel i in period p;

τ^{start}_{ip}: the unloading start time of vessel i in period p;

τ^{end}_{ip}: the loading end time of vessel i in period p.

2.3.2 MIP models

In this section, we establish a mixed-integer programming (MIP) model for the problem, based on the order of the objective function, subblock allocation constraints, container allocation constraints, and loading and unloading time constraints. At the end of this section, we get the complete MIP model and decompose it into two sub-models.

We consider an integrated decision model of subblock location assignation loading and unloading time allocation in the yard in this problem. The objective includes two parts: (1) minimizing the total loading and unloading distance of containers; (2) minimizing the time deviation between the actual loading and unloading time and the expected turnaround time of each vessel. In order to obtain the first part of the objective function,

that is, the total distance obj_{route} covered by trucks, we make the following analysis.

Objective: traveling distance

The total traveling distance is related to two factors. One factor is the actual length of each loading and unloading route between subblock k and vessel i. The other factor is the number of times that each truck passes through each route. For the first factor, when the location of vessel i and the location of subblock k is fixed, the length of loading or unloading route between the two positions is a known parameter. Let notation d^L_{ipk} denote the length of loading route from subblock k to vessel i, and notation d^U_{ipk} denote the length of unloading route from vessel i to subblock k. For the second factor, as mentioned in Subsection 2.2.3, each truck can carry only one container at a time. So the number of times a truck transports containers through a route equals to the number of containers transported through this route. In fact, this number is the number of containers allocated to subblock k, which is the decision variable z_{jikp}. Variable z_{jikp} needs to satisfy the condition that $\sum_{k\in K_{ip}} z_{jikp} = q_{jip}$. This condition means that the sum of containers allocated to each subblock equals to the total number of transferred containers. According to the above definition, the total traveling distance of all transshipment containers during the entire planning horizon can be calculated by the following formula:

$$\sum_{i\in V}\sum_{p\in T_i}\sum_{k\in K_i}\left(\sum_{j\in V^S_{ip}} d^U_{jpk} z_{jipk} + d^L_{ipk}\sum_{j\in V^S_{ip}} z_{jipk}\right), \tag{2.1}$$

where V^S_{ip} is the set of source vessels of vessel i in period p. K_i is the set of preferred subblocks of vessel i. T_i is the set of all periods of vessel i. V is the set of all vessels in the planning horizon.

Objective: time deviation

The second part of the objective function, that is, the total time deviation obj_{time} between the actual handling time and the expected handling time of all vessels during the planning horizon, can be calculated by the following formula:

$$obj_{time} = \sum_{i \in V} \left[\left(\alpha_{ip}^{start} - \tau_{ip}^{start} \right)^{+} + \left(\tau_{ip}^{end} - \beta_{ip}^{end} \right)^{+} \right], \tag{2.2}$$

where $\left(\alpha_{ip}^{start} - \tau_{ip}^{start} \right)^{+} = max\{\alpha_{ip}^{start} - \tau_{ip}^{start}, 0\}$ indicates the time deviation between the start time of the unloading operation and the expected start time of vessel i in period p. $\left(\tau_{ip}^{end} - \beta_{ip}^{end} \right)^{+} = max\{\tau_{ip}^{end} - \beta_{ip}^{end}, 0\}$ indicates the time deviation between the end time of the loading operation and the expected end time of vessel i in period p.

Constraints on subblock allocation

The constraints related to subblock allocation are described below:

$$\sum_{i \in V} x_{tki} \leq 1, \quad \forall k \in K, t \in T, \tag{2.3}$$

$$\sum_{k \in K_i} \widetilde{x}_{pki} = n_{ip}, \quad \forall i \in V, p \in T_i, \tag{2.4}$$

$$\sum_{t \in T_{ip}} x_{tki} = |T_{ip}| \, \widetilde{x}_{pki}, \quad i \in V, p \in T_i, k \in K_i, \tag{2.5}$$

$$\sum_{k \in K_i} \widetilde{y}_{pkij} = n_{jip}, \quad \forall i \in V, j \in V_{ip}^{S}, p \in T_i, \tag{2.6}$$

$$\widetilde{y}_{pkij} \leq \widetilde{x}_{pki}, \quad \forall i \in V, j \in V_{ip}^{S}, k \in K, p \in T_i, \tag{2.7}$$

Constraints (2.3) ensure that each subblock can be allocated to only one vessel at a time. Constraints (2.4) ensure that the n_{ip} subblocks assigned to vessel i at period p are selected from the subblocks preferred by vessel i. Constraints (2.5) ensure that in period p, if subblock k is assigned to vessel i, then at each time unit of period p, subblock k is assigned to vessel i. Constraints (2.6) ensure that in period p of vessel i, n_{jip} subblocks reserved for vessel i are allocated to store containers from source vessel j. Constraints (2.6) ensure that in period p, unloading of containers in subblock k from the source vessel j of vessel i is meaningful only if subblock k is assigned to vessel i in that period.

Constraints on container number allocation

The following are the constraints related to the number of containers allocated to each subblock:

$$\sum_{k \in K_i} z_{ijkp} = n_{jip}, \quad \forall i \in V, j \in V_{ip}^S, p \in T_i, \tag{2.8}$$

$$z_{jikp} \leq \widetilde{y}_{pkij} M, \quad \forall i \in V, j \in V_{ip}^S, p \in T_i, k \in K_i, \tag{2.9}$$

$$\sum_{j \in V_i^L} z_{jikp} \leq C_k, \quad \forall k \in K, i \in V, p \in T_i, \tag{2.10}$$

Constraints (2.8) represent that in the period p of vessel i, the total number of containers transferred to vessel i from source vessel j ($j \in V_{ip}^S$) is the sum of the number of containers unloaded to each subblock from vessel j. Constraints (2.8) ensure that all transshipment containers from vessel j to vessel i are stored in subblocks. Constraints (2.9) ensure that allocation of containers to a subblock is meaningful only when the subblock is assigned. Constraints (2.10) ensure that the total number of containers stored in each subblock in period p be within the capacity of the subblock.

Constraints on handling time assignment

The following constraints are related to the expected loading time and unloading time at each subblock:

$$\lambda_{jikt}^U \leq \widetilde{y}_{pkij}, \quad \forall t \in T_{ip}, k \in K_i, i \in V, j \in V_{ip}^S, \tag{2.11}$$

$$\lambda_{jikt}^U t \geq t_{ip}^{F,start}, \quad \forall i \in V, j \in V_{ip}^S, t \in T_{ip}, k \in K_i, \tag{2.12}$$

$$\lambda_{jikt}^U t \leq \tau_{jp}^{L,start}, \quad \forall j \in V_{ip}^S, i \in V, t \in T_{jp}, k \in K_i, \tag{2.13}$$

$$\lambda_{jikt}^U t \leq \tau_{ip}^{L,start}, \quad \forall i \in V, j \in V_{ip}^S, t \in T_{ip}, k \in K_i, \tag{2.14}$$

$$\delta_{it}^L t \leq t_{ip}^{F,end}, \quad \forall i \in V, t \in T_{ip}, p \in T_i, \tag{2.15}$$

$$\tau_{ip}^{L,start} = \min_{t \in T_{ip}} \{\delta_{it}^L t\}, \quad \forall i \in V, p \in T_i, \tag{2.16}$$

$$\tau_{ip}^{start} \leq \lambda_{ijkt}^U t, \quad \forall i \in V, j \in V_{ip}^S, t \in T_{ip}, k \in K_i, \tag{2.17}$$

$$\tau_{ip}^{end} \geq \delta_{it}^U t, \quad \forall i \in V, t \in T_{ip}, \tag{2.18}$$

Constraints (2.11) guarantee that the unloading time of subblock k exists only when subblock k is assigned to source vessel j of vessel i in period p. Constraints (2.12) en-

sure that vessel j unloads containers to each subblock k at a time later than its earliest feasible unloading time. Constraints (2.13) ensure that the unloading time of vessel j be later than its arrival time. Constraints (2.14) ensure that source vessel j unloads to subblock k before vessel i loads containers from subblock k. Constraints (2.15) ensure that vessel i loads containers from each subblock earlier than its latest feasible loading time. Constraints (2.16) ensure that the loading start time of vessel i is equal to the minimum loading time of each subblock. Constraints (2.17) ensure the actual handling start time of vessel i in period p. Constraints (2.18) ensure the actual handling end time of vessel i in period p.

Constraints on traffic congestion

Following are constraints to avoid traffic congestion in the yard.

$$\sum_{i\in V}\sum_{k\in N_b}\left(\delta^L_{it}x_{tki}+\sum_{j\in V^L_i}\lambda^U_{jikt}x_{tki}\right)\le 1,\quad \forall t\in T, N_b\in\mathbb{N}_b, \tag{2.19}$$

$$\sum_{i\in V}\sum_{k\in N_h}\left(\delta^L_{it}x_{tki}+\sum_{j\in V^L_i}\lambda^U_{jikt}x_{tki}\right)\le 1,\quad \forall t\in T, N_h\in\mathbb{N}_h, \tag{2.20}$$

$$\sum_{i\in V}\sum_{k\in K_i}\delta^L_{it}x_{tki}\pi_{iku}\le w^L_u,\quad \forall t\in T, u\in U, \tag{2.21}$$

$$\sum_{i\in V}\sum_{k\in K_i}\sum_{j\in V^L_i}\lambda^U_{jikt}x_{tki}\pi_{iku}\le w^U_u,\quad \forall t\in T, u\in U, \tag{2.22}$$

Constraints (2.19) ensure that at the most one subblock can load or unload in the same block during a time unit. Constraints (2.20) ensure that subblocks in adjacent blocks sharing the same horizontal lane cannot load or unload at the same time. Constraints (2.21) limit the loading traffic flows passing each lane simultaneously. Constraints (2.22) limit the unloading traffic flows passing each lane simultaneously.

To sum up, the complete MIP model (M1) is written as follows.

$$\text{[M1]} \quad \min \quad \eta_1 \sum_{i\in V}\sum_{p\in T_i}\sum_{k\in K_i}\left(\sum_{j\in V_{ip}^S} d_{jpk}^U z_{jipk} + d_{ipk}^L \sum_{j\in V_{ip}^S} z_{jipk}\right)$$
$$+ \eta_2 \sum_{i\in V}\left[\left(\alpha_{ip}^{start} - \tau_{ip}^{start}\right)^+ + \left(\tau_{ip}^{end} - \beta_{ip}^{end}\right)^+\right], \tag{2.23}$$

$$\text{s.t.} \quad \text{Constraints} \quad (2.3) - (2.22), \tag{2.24}$$

$$x_{tki} \in \{0,1\}, \quad \forall i \in V, t \in T, k \in K_i, \tag{2.25}$$

$$\widetilde{x}_{pki} \in \{0,1\}, \quad \forall i \in V, p \in T_i, k \in K_i, \tag{2.26}$$

$$\widetilde{y}_{pkij} \in \{0,1\}, \quad \forall i \in V, j \in V_{ip}^S, p \in T_i, k \in K_i, \tag{2.27}$$

$$z_{jikp} \geq 0, \quad \forall i \in V, j \in V_{ip}^S, p \in T_i, k \in K_i, \tag{2.28}$$

$$\lambda_{jikt}^U \in \{0,1\}, \quad \forall i \in V, j \in V_{ip}^S, t \in T_{ip}, k \in K_i, \tag{2.29}$$

$$\delta_{it}^L \in \{0,1\}, \quad \forall i \in V, t \in T_{ip}, \tag{2.30}$$

$$\tau_{ip}^{L,start} \geq 0, \quad \forall i \in V, p \in T_i, t \in T_{ip}, \tag{2.31}$$

$$\tau_{ip}^{start} \in \{1,\ldots,T\}, \quad \forall i \in V, p \in T_i, \tag{2.32}$$

$$\tau_{ip}^{end} \in \{1,\ldots,T\}, \quad \forall i \in V, p \in T_i, \tag{2.33}$$

In model M1, constraints (2.3)-(2.10) are only related to the traveling distance obj_{route} in the objective. They correspond to the subblock allocation constraints and the container allocation constraints. In particular, we get the following model with obj_{route} as its objective, namely, model M2.

$$\text{[M2]} \quad obj_{route}$$
$$\text{s.t.} \quad \text{Constraints} \quad (2.3) - (2.10), (2.19) - (2.22), (2.25) - (2.28) \tag{2.34}$$

Besides, in model M1, constraints (2.11)-(2.18) are related only to loading and unloading time allocation. These decisions affect only the obj_{time} term in the objective function. Constraints (2.19)-(2.22) of model M1 guarantee that traffic congestions are avoided; they are affected by both the loading and unloading time decision and the subblock allocation

decision. It is noted from the constraint (2.11) that the decision on the loading and unloading time of vessels is made based on the decision of the subblock allocation. Thus, when the decisions of subblock locations are given, traffic congestion can be avoided by constraints (2.19)-(2.22), i.e., the loading and unloading time allocation decisions. Then, another sub-model M3 is defined as follows.

$$\begin{aligned} &[\text{M3}] \quad obj_{time} \\ &\text{s.t.} \quad \text{Constraints} \quad (2.11) - (2.22), (2.29) - (2.33) \end{aligned} \tag{2.35}$$

In model M3, values related to the subblock decisions shown in Constraints (2.25)-(2.28) are parameters.

2.4 Solution Method

We define four notations to show the four sets of decision variables. That is, $\mathcal{A}^K$ indicates the set of decisions of locations. $\mathcal{A}^N$ indicates the set of decisions of container numbers. $\mathcal{A}^U$ indicates the set of decisions for unloading time. $\mathcal{A}^L$ indicates the set of decisions of loading time. In the rest of this section, we use these four notations to represent the relevant decisions. In model M1, decisions $\mathcal{A}^K$, $\mathcal{A}^U$ and $\mathcal{A}^L$ are integer variables. In addition, the four decisions are related to each other. Therefore, the scale of the problem increases rapidly when the planning horizon becomes longer. Accordingly, it is hard for the existing commercial solvers to calculate a feasible solution. Besides, periods of different vessels share some common time units without a general pattern. Therefore, allocations for different vessels cannot be divided into several independent sub-problems regarding each vessel. Thus, this problem cannot be tackled by the column generation method for large-scale problems. Based on the above reasons, we design a critical element-based three-stage assignment (CETSA) algorithm to solve this problem.

First, we present two propositions before introducing the CETSA algorithm.

Proposition 1 (feasibility condition) By solving model M2, we obtain the com-

bination of feasible solutions of subblock location and the number of containers, i.e., $(\mathcal{A}^K, \mathcal{A}^N)$. Given the above values of $(\mathcal{A}^K, \mathcal{A}^N)$ as parameters, we can solve model M3 and get its feasible solutions $(\mathcal{A}^U, \mathcal{A}^L)$. Then, the combination of the solutions of model M2 and model M3 $(\mathcal{A}^K, \mathcal{A}^N, \mathcal{A}^U, \mathcal{A}^L)$ constitutes a feasible solution of model M1.

Proposition 2 (optimality condition) Given one feasible solution $(\mathcal{A}^K, \mathcal{A}^N)$ of model M2, we can obtain the optimal solution of M3 under M2's solution by searching all feasible values of $(\mathcal{A}^U, \mathcal{A}^L)$. Then, by traversing all feasible values of M2, we can obtain the solution $(\mathcal{A}^K, \mathcal{A}^N, \mathcal{A}^U, \mathcal{A}^L)$ with the minimum objective value, which is the optimal solution of model M1.

2.4.1 Framework of a three-stage heuristic algorithm

It is known from Proposition 1 that the feasible solution of model M1 can be constructed by solving sub-model M2 and sub-model M3. According to Proposition 2, as long as sub-model M2 and sub-model M3 can thoroughly search all feasible solutions, the optimal solution of the original model M1 can be obtained. However, because M2 and M3 have to traverse many times and need to be solved efficiently in each traversal, it is impossible to traverse all feasible solutions thoroughly. For this reason, we design an iterative algorithm based on constructive solutions. This algorithm solves M2 and M3 efficiently at different stages. The framework of this algorithm is shown in Algorithm 1.

In Algorithm 1, Lines 1-4 show the generation of the initial feasible solution. As can be seen from the algorithm, the key to the initial solution is to generate a feasible $\mathcal{A}^K$. In order to obtain the initial $\mathcal{A}^K$, we only consider the constraints related to the allocation of subblock locations in model M2, that is, constraints (2.3)-(2.7) and (2.19)-(2.20). At this stage, traffic congestion constraints are not considered since the loading and unloading time decisions are unknown at the initial stage.

The initial $\mathcal{A}^K$ is generated while limiting traffic congestion in the yard. We use the following four empirical rules to generate the initial $\mathcal{A}^K$. (1) All subblocks in the yard are sorted according to the frequencies preferred by the vessels. Then, we assign each subblock

Algorithm 1 General framework of the CETSA algorithm

Input: all parameters and data sets

Output: a near-optimal feasible solution

1: Generate an initial subblock locations assignment $\mathcal{A}^K$ for sub-model M2
2: Based on $\mathcal{A}^K$, generate the container number assignment $\mathcal{A}^N$, and obtain the initial traveling distance obj_{route}
3: For sub-model M3, fix the values of variables corresponding to $\mathcal{A}^K$
4: Solve sub-model M3 and obtain a feasible solution of $(\mathcal{A}^U, \mathcal{A}^L)$ and the objective value of obj_{time}
5: Fix the value of the relevant variables in subproblem M2 to the value of the solution $(\mathcal{A}^U, \mathcal{A}^L)$ of M3
6: Modify the current subblock allocation assignment $\mathcal{A}^K$
7: Based on the modified $\mathcal{A}^K$, compute a new container number assignment $\mathcal{A}^N$ and obtain the objective value obj_{route}
8: Fix the variable value of $\mathcal{A}^K$ in sub-model M3 to the value of $\mathcal{A}^K$ computed from M2
9: Solve M3, obtain solution $(\mathcal{A}^U, \mathcal{A}^L)$ and objective obj_{time}
10: Update the optimal solution
11: Repeat steps 5-10 until the stopping criteria is satisfied
12: Return solution $\mathcal{A}^K, \mathcal{A}^N, \mathcal{A}^U, \mathcal{A}^L$ and objective $obj_{route} + obj_{time}$

to a vessel in frequency-increasing order. (2) If two vessels prefer the same subblock, the subblock is assigned to the vessel requiring the most subblocks. (3) When two vessels require an equal number of subblocks, the subblock is assigned to the vessel that prefers the least subblocks. (4) When the above three rules cannot solve the allocation of a subblock, we assign this subblock to a random vessel.

In Algorithm 1, Lines 5-7 solve model M2. Line 5 sets the values of parameters in model M2. Line 6 changes the values of $\mathcal{A}^K$, which is the key step of this algorithm and is implemented through a variable neighborhood search procedure. We introduce this procedure in Subsection 2.4.2. Line 7 computes the value of $\mathcal{A}^N$, which is directly solved by the solver CPLEX.

In Algorithm 1, Lines 8-9 solve model M3. Line 8 is the setting of parameter values before solving model M3. Line 9 calculates the unloading time $\mathcal{A}^U$ and the loading time $\mathcal{A}^L$, explained in detail in Subsection 2.4.7.

In Algorithm 1, Line 10 updates the optimal solution when a new feasible solution is obtained after each iteration. Line 11 shows the stopping criteria. Three conditions are adopted to stop the algorithm. (1) The total number of iterations of model M2 reaches 1000. (2) The total running time of the algorithm exceeds 1800 seconds. (3) The number of iterations without improvement reaches 600.

As shown in the framework, the key to solving this problem is finding feasible solutions to M2 and M3. Next, we design algorithms for M2 and M3, respectively.

2.4.2 Solution procedure for the location assignment $\mathcal{A}^K$

In this section, we use a variable neighborhood search (VNS) algorithm to generate a subblock allocation scheme $\mathcal{A}^K$. Each neighborhood is generated based on the critical element, described in Subsection 2.4.3. The basic idea of VNS is strategically changing the neighborhoods for the local search procedure. Two types of operators, i.e., the neighborhood operators and the shaking operators, are defined. These operators are used for changing the neighborhoods. Starting from an initial solution, the VNS generates the first neighborhood using the shaking operator and randomly selects a solution from this neighborhood. Next, a variable neighborhood local search is carried out with pre-defined neighborhood operators based on this selected solution. During this procedure, an iterative improvement algorithm is repeated between different neighborhoods until a new incumbent solution is obtained. If a new incumbent solution is found in this neighborhood, the procedure reruns with this new solution from the first neighborhood operator. Otherwise, the procedure moves to the next neighborhood operator. If no better solution is found with all neighborhood operators, we shake the current solution with the next shaking operator and randomly select a solution from this neighborhood as the incumbent solution. Then the VNS procedure continues following the above steps.

Algorithm 2 shows the basic steps of the VNS algorithm. In this algorithm, N_s ($N_s = \{s_1, \ldots, s_{max}\}$) denotes the set of shaking operators. N_o ($N_o = \{o_1, \ldots, o_{max}\}$) denotes the set of neighborhood operators. In each iteration of VNS, a feasible allocation $(\mathcal{A}^K, \mathcal{A}^N)$

Algorithm 2 The VNS algorithm for generating $\mathcal{A}^K$

Input: the initial subblock location assignment $(\mathcal{A}^K)^0$

Output: a new subblock location assignment $(\mathcal{A}^K)^*$

1: Incumbent assignment $(\mathcal{A}^K)^0$
2: Optimal assignment $(\mathcal{A}^K)^*$
3: **for** each shaking operator N_s **do**
4: $\quad\mathcal{A}^K = shake(\mathcal{A}^K, N_s)$
5: $\quad$**for** each neighborhood operator N_{o_i} $(N_{o_1} \rightarrow N_{o_{max}})$ **do**
6: $\quad\quad$local optimal solution $(\mathcal{A}^K)^N \leftarrow LocalSearch(\mathcal{A}^K, N_{o_i})$
7: $\quad\quad$**if** $(\mathcal{A}^K)^N \leq \mathcal{A}^K$ **then**
8: $\quad\quad\quad\mathcal{A}^K = (\mathcal{A}^K)^N$
9: $\quad\quad\quad N_{o_i} = N_{o_1}$
10: $\quad\quad\quad$**if** $\mathcal{A}^K \leq (\mathcal{A}^K)^*$ **then**
11: $\quad\quad\quad\quad(\mathcal{A}^K)^* = \mathcal{A}^K$
12: $\quad\quad\quad$**end if**
13: $\quad\quad$**end if**
14: $\quad$**end for**
15: **end for**
16: **return** $(\mathcal{A}^K)^*$

is determined, where $\mathcal{A}^N$ is directly computed by CPLEX after the $\mathcal{A}^K$ is determined. Therefore, in the VNS procedure, we only need to iterate over the subblock solution $\mathcal{A}^K$. In the remainder of this section, unless otherwise specified, the allocation scheme mentioned refers to solution $\mathcal{A}^K$.

Line 6 in Algorithm 2 indicates that a local search is performed on the basis of a given neighborhood operator, and the locally optimal solution is returned. During the local search, for each neighborhood solution $\mathcal{A}^K$, the algorithm calculates its integrated solution $\mathcal{A}^K, \mathcal{A}^N, \mathcal{A}^U, \mathcal{A}^L$ and its objective value. The objective value is taken as the current solution value and is used to determine the next neighborhood.

Both the neighborhood operators and the shaking operators are designed based on the critical element. Therefore, in the following subsection, we introduce the critical element before introducing the two operators.

2.4.3 The critical element

The element in the algorithm represents the assignation of subblock locations for vessel i in period p, denoted by $\mathcal{A}_{ip}^K$. Obviously, each element corresponds to a vessel-period pair. Therefore, this problem contains a set of vessel-period pairs. Then, this problem can also be viewed as being composed of a set of elements. Based on the above definition of the element, the critical element is defined in the following paragraphs.

According to the constraints on $\mathcal{A}^K$ in Subsection 2.2.4, the assignation $\mathcal{A}_{ip}^K$ for one vessel-period can affect the assignations of other vessels and other periods. Therefore, the change of one element $\mathcal{A}_{ip}^K$ impacts other elements of $\mathcal{A}^K$. However, due to the influence of various preferred subblocks, berthing locations, and handling times, the impact of modifying each element is different.

In a local search procedure of the VNS, we can generate the neighborhood of the assigned by imposing a neighborhood operator on an element. To make the local search procedure efficient, we hope to search for the most promising neighborhood. Since distinct elements result in different neighborhoods, we only need to find out the element leading to the most promising neighborhood. For assignment $\mathcal{A}^K$, we define the "critical element" as the element which improves the objective most when a disturbance is imposed on it.

In order to determine the critical element in $\mathcal{A}^K$, we develop a two-stage element scoring method in the VNS procedure. In the first stage, we grade all vessels and select the vessel with the highest score as the critical vessel. In the second stage, we grade all periods for the critical vessel and select the period with the highest score. Then, we obtain the critical element which is combined with the selected vessel and its selected period. The formulas for evaluating the scores in the two stages are introduced as follows. Let

$$D_{ip} = \sum_{k \in K_i} \left(\sum_{j \in V_{ip}^S} d_{jpk}^U z_{jipk} + d_{ipk}^L \sum_{j \in V_{ip}^S} z_{jipk} \right) \tag{2.36}$$

denote the traveling distance of vessel i in period p for assignment $\mathcal{A}^K$. In other words, D_{ip} is the length corresponding to element $\mathcal{A}_{ip}^K$.

In the first stage, we aim to select the vessel corresponding to the critical element, that is, the vessel that makes the greatest contribution to the traveling distance. Because the number of containers transferred from different vessels is different, we cannot identify whether the traveling distance of each element is mainly caused by the assignation plan or the number of containers. To find out the critical vessel of an assignment $\mathcal{A}^K$, we calculate the average distance $\bar{d}_i$ of each transshipped container for each vessel i. The formula for computing $\bar{d}_i$ is shown in Equation (2.37).

$$\bar{d}_i = \frac{\sum_{p \in T_i} D_{ip}}{\sum_{p \in T_i} \sum_{j \in V_{ip}^S} q_{jip}} \tag{2.37}$$

Then, we can get the vessel i^* corresponding to the critical element, which is the vessel with the highest $\bar{d}_i$ score, that is,

$$i^* = \arg\max_{i \in V} \bar{d}_i \tag{2.38}$$

In the second stage, for the assignment $\mathcal{A}_{ip}^K$ of vessel i in period p, we first define its conflict degree c_{ip}. Since each vessel has its preferred subblocks, and the preferred subblocks of different vessels may overlap, a certain subblock can be preferred by multiple vessels. If such a commonly preferred subblock is assigned to one vessel, available subblocks for other vessels will be reduced. Therefore, such kind of subblocks can cause more conflicts in an assignment compared with other subblocks. For these reasons, for each subblock k in assignment $\mathcal{A}_{ip}^K$, we compute its conflict degree c_{ipk} with the following formula:

$$c_{ipk} = \begin{cases} 0, & k \notin \mathcal{A}_{ip}^K, \\ |\mathbb{M}_k|, & k \in \mathcal{A}_{ip}^K,\ k \in \cap_{K_i \in \mathbb{M}_k} K_i \end{cases} \tag{2.39}$$

where the set $\mathbb{M}_k$ is a set comprising a set of preferred subblocks of vessels that prefer subblock k. It can be seen from Equation (2.39) that when subblock k is not in $\mathcal{A}_{ip}^K$, the

conflict degree of subblock k is 0. When subblock k is preferred by vessel i, the conflict degree of subblock k is equal to the number of vessels preferring subblock k, i.e., $|\mathbb{M}_k|$.

Then, we can compute the conflict degree of assignment $\mathcal{A}_{ip}^K$, namely c_{ip}, which is the sum of c_{ipk} of all subblocks in $\mathcal{A}_{ip}^K$ as shown in Equation (2.40).

$$c_{ip} = \sum_{k \in \mathcal{A}^K} c_{ipk} \tag{2.40}$$

Now, for the critical vessel i^*, we can evaluate the score of its period p ($p \in T_{i^*}$) with Equation (2.41). The period with the highest score is chosen as the period corresponding to the critical element, i.e., $p^* = \arg\max_{p \in T_i} c\,(i^*, p)$.

$$c\,(i^*, p) = \frac{D_{ip}}{\sum_{j \in V_{ip}^S} q_{jip}} \times \frac{c_{ip}}{\sum_{j \in V_{ip}^S} q_{jip}} \tag{2.41}$$

So far, after scoring at the two stages, we obtain a vessel-period pair (i^*, p^*) which corresponds to the critical element according to the critical element of assignment $\mathcal{A}^K$ is $\mathcal{A}_{i^*,p^*}^K$.

2.4.4 Operators for $\mathcal{A}^K$

The neighborhood operators

In the algorithm, we define three neighborhood operators $\mathcal{N}_{o_1}$, $\mathcal{N}_{o_2}$ and $\mathcal{N}_{o_3}$, that is, $o_{max} = 3$. In the process of VNS, these neighborhood operators are called in the order of $\mathcal{N}_{o1}$, $\mathcal{N}_{o2}$ and $\mathcal{N}_{o3}$. Based on the critical element, we define $\mathcal{N}_{o1}$, $\mathcal{N}_{o2}$ and $\mathcal{N}_{o3}$ with the following rules.

(1) Rules of neighborhood operator $\mathcal{N}_{o_1}$: For vessel i^* in period p^* regarding the critical element $\mathcal{A}_{i^*p^*}^K$, we only change the decisions of the subblock locations of its source vessels while keeping the assignment of the critical element $\mathcal{A}_{i^*,p^*}^K$ unchanged. For two source vessels j_1 and j_2 of vessel i^* in period p^*, operator $\mathcal{N}_{o_1}$ randomly chooses a subblock from

each source vessel and exchanges them. That is, in model M2, we maintain the values of the decision variables $\widetilde{x}_{p^*ki^*}$, and change the values of the decision variables $\widetilde{y}_{p^*ki^*j}$.

(2) Rules of neighborhood operator $\mathcal{N}_{o_2}$: First, we randomly select an assigned subblock k_1 from the critical element $\mathcal{A}^K_{i^*,p^*}$. Then, for vessel i^* in period p^*, we select another subblock k_2 from its preferred set K_{i^*}. Subblock k_2 cannot be assigned to any vessel in $\mathcal{A}^K$. Finally, we replace all assignments of subblock k_1 with subblock k_2 in the $\mathcal{A}^K$. That is, in model M2, operator $\mathcal{N}_{o_2}$ changes the values of the decision variables $\widetilde{x}_{p^*k_1i^*}$ from 1 to 0 and the values of variables $\widetilde{x}_{p^*k_2i^*}$ from 0 to 1. Meanwhile, it changes the values of variables $\widetilde{y}_{p^*k_1i^*j}$ from 1 to 0 and the value of variables $\widetilde{y}_{p^*k_2i^*j}$ from 0 to 1.

(3) Rules of neighborhood operator $\mathcal{N}_{o_3}$: First, we randomly select an assigned subblock k_1 from the critical element $\mathcal{A}^K_{i^*,p^*}$. Then, we randomly select another subblock k_2 from element $\mathcal{A}_{jp'}$, where vessel j's period p' overlaps with vessel i^*'s period p^*. Besides, subblock k_2 must also be in the preferred set K_{i^*} of vessel i^*, i.e., $k_2 \in K_{i^*} \cap K_j$. Finally, we replace all assignments of subblock k_1 with subblock k_2 in the $\mathcal{A}^K$. That is, in model M2, operator $\mathcal{N}_{o_2}$ changes the values of the decision variables $\widetilde{x}_{p^*k_1i^*}$ from 1 to 0 and the values of variables $\widetilde{x}_{p^*k_2i^*}$ from 0 to 1. Meanwhile, it changes the values of variables $\widetilde{x}_{p'k_2j}$ from 1 to 0 and the value of variables $\widetilde{x}_{p'k_2j}$ from 0 to 1.

The shaking operator

When the three neighborhood operators cannot improve the assignment by an iteration of the VNS, we make a small perturbation to assignment $\mathcal{A}^K$ through the shaking operator $\mathcal{N}_{s_1}$. The operations of $\mathcal{N}_{s_1}$ are as follows. First, we select a random element $\mathcal{A}^K_{ip}$ from assignment $\mathcal{A}^K$. Then, we select a subblock k_1 which is assigned in $\mathcal{A}^K_{ip}$ and a preferred subblock k_2 which is not assigned in $\mathcal{A}^K_{ip}$. That is, $k_1 \in \mathcal{A}^K_{ip}$, $k_2 \notin \mathcal{A}^K_{ip}$ and $k_2 \in K_i$. Finally, we swap the assignment decisions of k_1 and k_2 in $\mathcal{A}^K$.

2.4.5 Feasible adjustment of assignment $\mathcal{A}^K$

A feasible assignment $\mathcal{A}^K$ may become infeasible after the operation of a neighborhood operator or a shaking operator. We denote this infeasible assignment as $(\mathcal{A}^K)'$. In this

subsection, we design a feasible adjustment algorithm to convert $(\mathcal{A}^K)'$ into a feasible assignment by fixing its violated constraints. The feasible adjustment algorithm is invoked many times during the VNS procedure.

We only need to fix the constraints corresponding to the assignment of $\mathcal{A}^K$. As discussed in model formulation, they are (1) constraints related to subblock locations, that is, Constraints (2.3)-(2.7); and constraints related to traffic congestion in horizontal lanes, namely Constraints (2.19) and (2.20). Algorithm 3 shows the steps of the feasible adjustment algorithm. Its time complexity is $O(n)$, where n is the total number of pairs of vessel-period in the planning horizon.

Algorithm 3 The feasible adjustment algorithm of assignment $\mathcal{A}^K$

Input: an infeasible assignment $\mathcal{A}^K$

Output: a feasible assignment $(\mathcal{A}^K)'$

1: **for** each vessel-period pair (i,p) **do**
2: $\bar{K}_{ip}$ denotes the set of subblocks that cannot be assigned to (i,p)
3: **if** $i=1, p=1$ **then**
4: $\bar{K}_{ip}=\emptyset$
5: **else**
6: $\bar{K}_{ip}=\{k : k \in (j,p'), p \cap p' \neq \emptyset\}$
7: **end if**
8: **for** each subblock $k \in \mathcal{A}^K_{ip}$ **do**
9: **if** $k \in \bar{K}_{ip}$ **then**
10: remove k from $\mathcal{A}^K_{ip}$
select a new subblock $k'(k' \in K_i, k' \notin \bar{K}_{ip})$ and add it to $\mathcal{A}^K_{ip}$
11: **end if**
12: **end for**
13: **for** each horizontal lane with congestion **do**
14: remove the subblocks violating Constraints (2.19)-(2.20) from $\mathcal{A}^K_{ip}$
select new subblocks from K_i and add them to $\mathcal{A}^K_{ip}$
until Constraints (2.19)-(2.20) are satisfied
15: **end for**
16: **end for**
17: **return** $(\mathcal{A}^K)'$

Algorithm 3 traverses all elements $\mathcal{A}^K_{ip}$ $(i \in V, p \in T_i)$. In Line 8-12, we update the subblock assignment of an element according to the assignments of its preceding

elements. In Lines 13-16, we adjust the two types of subblock allocation constraints near the horizontal lanes to guarantee their feasibility. Through Algorithm 3, we can transfer $(\mathcal{A}^K)'$ to a feasible assignment after the operation of an operator.

2.4.6 Solution to container number assignment

Once the subblock allocation scheme $\mathcal{A}^K$ is obtained, the solution to container number assignment can be obtained by solving the following model M4.

$$[\text{M4}] \quad \min \sum_{i\in V}\sum_{p\in T_i}\sum_{k\in K_i}\left(\sum_{j\in V_{ip}^S} d_{jpk}^U z_{jipk} + d_{ipk}^L \sum_{j\in V_{ip}^S} z_{jipk}\right), \tag{2.42}$$

$$\text{s.t.} \quad \sum_{k\in K_i} z_{ijkp} = n_{jip}, \quad \forall i \in V, j \in V_{ip}^S, p \in T_i, \tag{2.43}$$

$$z_{jikp} \le \widetilde{y}_{pkij} M, \quad \forall i \in V, j \in V_{ip}^S, p \in T_i, k \in K_i, \tag{2.44}$$

$$\sum_{j\in V_i^L} z_{jikp} \le C_k, \quad \forall k \in K, i \in V, p \in T_i, \tag{2.45}$$

The value of subblock decisions $\widetilde{y}_{pkij}$ are known from assignment $\mathcal{A}^K$ and are parameters in model M4. The only variable z_{jikp} is a continuous value. Therefore, model M4 is a linear programming model and can be solved by a commercial solver. After solving M4, we can obtain the total traveling distance of trucks, which is the first part of the objective of this problem.

2.4.7 Solution to handling time assignment

After the solution to assignment $\mathcal{A}^K$ is obtained, the values related to subblock locations in model M3 are known. Therefore, the only decision in model M3 is the variable related to handling time. In Algorithm 4, we design a heuristic method to decide the loading and unloading time based on the subblock allocation order.

Algorithm 4 Handling time allocation algorithm

Input: location assignment $\mathcal{A}^K$

Output: handling time assignment $\mathcal{A}^U, \mathcal{A}^L$

for each vessel-period pair (i, p) **do**
  get the feasible unloading time interval $[t_1^u, t_2^u]$
  get the sequence seq of subblocks for (i, p)
  for each subblock k in sequence seq **do**
    determine the earliest feasible unloading time t
    if $t \in [t_1^u, t_2^u]$ **then**
      set t as the unloading time of subblock k
    else
      shake seq and get a new sequence seq', let $seq = seq'$
      reset all the assigned unloading time
    end if
  end for
end for
for each vessel-period pair (i, p) **do**
  update the feasible loading time $[t_1^l, t_2^l]$ of (i, p)
end for
for each vessel-period pair (i, p) **do**
  get the feasible loading time interval $[t_1^l, t_2^l]$
  get the sequence seq of subblocks for (i, p)
  for each subblock k in sequence seq **do**
    determine the earliest feasible unloading time t
    if $t \in [t_1^l, t_2^l]$ **then**
      set t as the loading time of subblock k
    else
      shake seq and get a new sequence seq', let $seq = seq'$
      reset all the assigned loading time
    end if
  end for
end for
for each vessel-period pair (i, p) **do**
  compute the time deviation of (i, p)
  update the total time deviation of all vessels
end for
return total time deviation, assignment $\mathcal{A}^U, \mathcal{A}^L$

2.5 Computational Experiments

We test the proposed CETSA algorithm on artificially generated simulation data. This section includes four parts: (1) the generation of test data; (2) the test of the effectiveness of CETSA algorithm on data sets of different sizes; (3) the test of the effectiveness of the loading and unloading time decisions; (4) the comparison between different route lengths on the solution.

2.5.1 Data generation

In this subsection, we describe the parameter setting and test data generation in our computational experiments.

First, we introduce the setting of the yard parameters. In this problem, we only consider the twenty-feet equivalent unit (TEU) and use TEU as the basic allocation unit in a block. A block has 20 TEUs in length, 6 TEUs in width, and 5 TEUs in height. Each block can be divided equally into five subblocks. The length, width, and height of a subblock contain 4 TEUs, 6 TEUs, and 5 TEUs, respectively. That is, each subblock can store up to 120 containers. The berth position of each vessel at each period is randomly generated within the feasible range of the berth. The shortest vertical distance from the berth to the yard is set to 50 meters. The maximum traffic flow of a horizontal lane is set to 1. The maximum traffic flow in one direction of a vertical lane is set to 1. That is, $w_u^U = 1$ and $w_u^L = 1$. The layout of the yard is shown in Figure 2.7.

Next, we introduce the parameters relevant to the vessels. The planning horizon in computational experiments is set to 70 days, including three periods, i.e., the 7 days period, the 10 days period, and the 14 days period. For all the vessels, the proportion of vessels with the above three periods is 60%, 20% and 20%, respectively. We set 8 hours as a time unit. Thus, each day contains three time units, and the entire planning horizon contains 210 time units. For each vessel, the start time unit of its first period is randomly generated and is guaranteed to be within the range of the first period. The arrival time of each vessel in each period is randomly generated according to the period

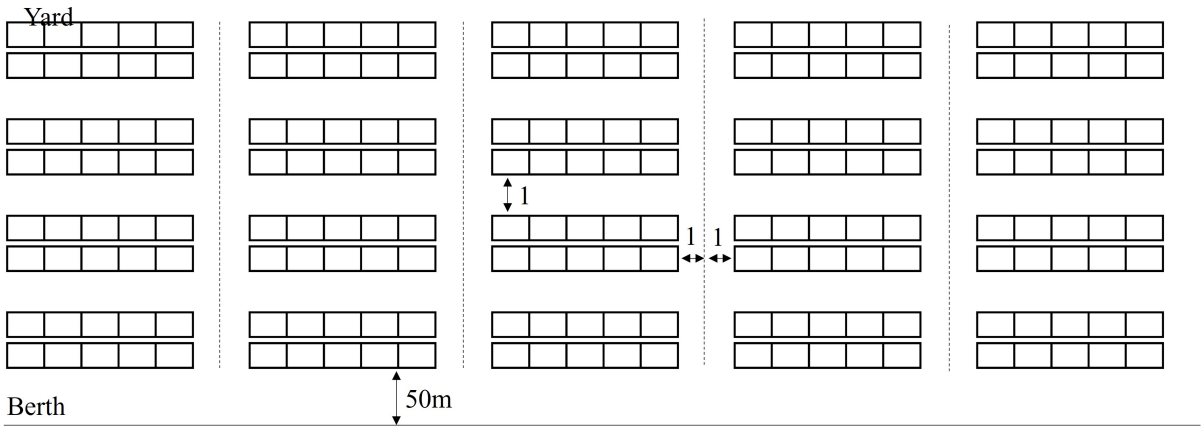


Figure 2.7: Yard layout and the relevant data

range. The preferred subblock set K_i for vessel i is also randomly generated. For vessel i in period p, the number n_{ip} of subblocks to be reserved is generated randomly with an average number of 5 subblocks. For vessel i in period p, we select vessels into its source vessel set V_{ip}^S and its destination vessel set V_{ip}^D such that each set contains 1 to 3 vessels uniformly. For a source vessel $j \in V_{ip}^S$ of vessel i in period p, we randomly generate the number of transshipment containers q_{jip}, such that q_{jip} does not exceed the total capacity of the n_{ip} reserved subblocks. Besides, the average number of transshipment containers between two vessels should be between $[500, 800]$. Based on the transshipment containers number q_{jip}, we generate the number of subblocks n_{jip} that should be reserved to vessel j satisfying that all the q_{jip} containers can be stored and $\sum_{j \in V_{ip}^S} n_{jip} = n_{ip}$. According to the number of vessels and the number of blocks, we generated ten groups of test instances, G1-G10. The parameters of the ten groups are listed in Table 2.1. For each data group, we randomly generate three test instances.

Table 2.1: Parameters of data sets

ID	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
Vessels	5	10	15	20	25	30	35	40	45	50
Subblocks	40	60	80	100	120	140	160	180	200	220
Blocks	8	12	16	20	24	28	32	36	40	44
Horizontal blocks	2	3	4	5	6	7	8	9	10	11
Vertical blocks	4	4	4	4	4	4	4	4	4	4
Berth Length	330	480	630	780	930	1080	1230	1380	1530	1680

When the parameters of the yard and the vessels are generated, the unloading route length d^U_{ipk} and the loading route length d^L_{ipk} between subblock k and vessel i can also be calculated. The coefficients η_1 and η_2 in the objective function are set to values 1×10^{-5} and 1×10^{-1} respectively.

2.5.2 Test of the effectiveness of the critical element

The core of the CETSA algorithm proposed in this chapter is the critical element. In order to test whether the local search based on the critical element has an obvious advantage over the local search based on random elements, we design an experiment for comparing the two methods. In this experiment, the random element-based three-stage assignment (RETSA) comparison algorithm is designed by replacing the critical element in CETSA with a random element while keeping other operations unchanged. That is, we only modify the method of solving assignment $\mathcal{A}^K$ and keep the method of solving assignment $\mathcal{A}^N$, $\mathcal{A}^U$ and $\mathcal{A}^L$ as described in the CETSA algorithm. We test the two algorithms on the five small-scale data groups G1-G5, and the results are listed in Table 2.2.

As can be seen from Table 2.2, for small-scale test data G1-G5, the CETSA algorithm based on the critical element can get better objective function values than the RETSA algorithm based on random elements in a given time, with an average improvement "Gap" $(= (z_2 - z_1)/z_1)$ of 4.3%. However, for instance 5_2, results of the two algorithms are equal to 31. Besides, in terms of the computational time required for obtaining the objective values, the CETSA algorithm takes slightly longer than the RETSA method. To sum up, our proposed CETSA algorithm is effective for deriving high-quality solutions on small-scale data.

Next, we compare the results of the two algorithms on the last five large data groups G6-G10. The results are listed in Table 2.3. The results of Table 2.3 show that the solutions obtained by CETSA are improved compared with the solutions obtained by RETSA on the large-scale data. The average improvement "Gap" of all the large instances

Table 2.2: Comparison results between CETSA and RETSA on small instances

Instance		CETSA		RETSA		
Group	id	z_1	Time(s)	z_2	Time(s)	Gap(%)
	5_1	45	789	47	477	4.8
G1	5_2	31	1767	31	417	0.0
	5_3	38	628	39	658	3.9
	10_1	102	677	107	1346	5.5
G2	10_2	100	507	107	520	6.8
	10_3	92	1255	96	787	4.4
	15_1	183	1459	190	996	4.2
G3	15_2	168	368	181	1246	7.7
	15_3	160	1108	162	527	1.3
	20_1	237	1688	254	777	6.9
G4	20_2	241	361	244	729	1.4
	20_3	254	1136	266	1292	4.7
	25_1	328	1483	334	1227	2.0
G5	25_2	331	547	340	353	2.7
	25_3	326	1724	351	1224	7.7
Avg.		176	1033	183	838	4.3

is 7.3%, which is larger than the improvement (4.3%) of small-scale data. This percentage improvement shows that the CETSA algorithm based on the critical element has obvious advantages in the case of large-scale data. The reason is that for large-scale data, the number of elements in each instance increases dramatically. It becomes more difficult to find a high-quality solution through a random element. Meanwhile, the critical element keeps the searching neighborhood in a promising scope. In addition, by comparing the computational time of the two algorithms on large-scale instances, we can see that the average time of obtaining the objective value of CETSA is slightly higher than that of RETSA. In comparison, the time of the two algorithms is similar in some instances, such as 30_2,45_2,50_1,50_2. The computational time of the CETSA algorithm is acceptable for obtaining a feasible solution. In addition, with the increase of data size, the disadvantage of computational time of CETSA gradually decreases.

Table 2.3: Comparison results between CETSA and RETSA on large instances

Instance		CETSA		RETSA		
Group	id	4_1	Time(s)	z_2	Time(s)	Gap(%)
	30_1	408	825	434	1430	6.3
G6	30_2	428	785	461	617	7.7
	30_3	435	1754	442	750	1.5
	35_1	519	1339	560	574	7.7
G7	35_2	525	1582	533	812	1.4
	35_3	550	970	596	696	8.4
	40_1	789	1627	837	738	6.1
G8	40_2	762	1158	842	731	10.6
	40_3	793	1860	855	1662	7.9
	45_1	701	1263	769	1366	9.7
G9	45_2	747	1472	767	780	2.7
	45_3	716	1348	789	965	10.1
	50_1	970	1428	1073	1495	10.6
G10	50_2	1152	1031	1259	1095	9.3
	50_3	1165	1751	1268	740	8.8
Avg.		711	1346	766	963	7.3

2.5.3 Effects of different assignment strategies

The models and algorithms proposed in this chapter serve the integrated decisions on subblocks and handling time assignment decisions. Our proposed CETSA is a comprehensive algorithm. In the decision-making process, CETSA iteratively searches for the two types of decisions while taking all vessels and all periods into consideration at the same time. In practice, "First Come First Serve" (FCFS) is a commonly used decision strategy. According to the FCFS strategy, we can obtain a solution by determining the assignment for all elements according to the time order. To verify the effectiveness of the comprehensive decision-making strategy of the CETSA algorithm compared with the FCFS strategy-based algorithm, we compare the two strategies based on the test data groups G4-G7. The experimental results are listed in Table 2.4.

From the last column "Gap" $((z_2-z_1)/z_1)$ in Table 2.4 we can see that the improvement percentage of the comprehensive allocation strategy in CETSA compared with FCFS strategy varies from 15.3% to 31.3%. This result indicates that the comprehensive strategy outperforms the FCFS strategy. Although the computational time of CETSA is slightly longer than that of FCFS, this computational time is acceptable for obtaining a feasible solution. The FCFS strategy takes quite a short time because it only needs to loop all the vessels once in chronological order for determining the allocation scheme. The comprehensive strategy in CETSA can save about 23.6% of the objective value compared to the FCFS strategy. This side effect meets the requirements of energy saving and emission reduction and provides more reasonable arrangements for vessels.

2.5.4 The effect of handling time decisions on total distance

In this chapter, we decide each vessel's unloading time and loading time, while the handling times are often taken as parameters in other related studies. To test whether the handling time decision can improve the total traveling distance of trucks, we set up the following comparative experiment.

In this experiment, two cases are compared: one with the handling time decision and

Table 2.4: Comparison results between CETSA and FCFS strategies

Instance		CETSA		FCFS		
Group	id	z_1	Time(s)	z_2	Time(s)	Gap(%)
	20_1	237	1688	293	<5	23.5
G4	20_2	241	361	291	<5	20.8
	20_3	254	1136	293	<5	15.3
	25_1	328	1483	426	<5	30.0
G5	25_2	331	547	402	<5	21.4
	25_3	326	1724	411	<5	25.9
	30_1	408	825	506	<5	24.0
G6	30_2	428	785	562	<5	31.3
	30_3	435	1754	511	<5	17.5
	35_1	519	1339	640	<5	23.3
G7	35_2	525	1582	659	<5	25.4
	35_3	550	970	685	<5	24.6
Avg.		382	1183	473	<5	23.6

the other without the handling time decision. For the problem with the handling time decision, i.e., model M1, we find its solution using the CETSA algorithm. We use the full objective value of M1 for each iteration but report only the value of the total traveling distance obj_{route} as the results of this experiment. For the problem without the handling time decision, we solve it according to the following steps. First, we randomly generate the unloading time and loading time of each reserved subblock for each vessel-period pair. Second, we set the relevant values of loading and unloading time variables in model M2 to the time generated in the first step. Finally, we solve model M2 with the CETSA algorithm and obtain assignment $\mathcal{A}^K$, assignment $\mathcal{A}^N$, and the total traveling distance obj_{route}. For large-scale data groups G8-G10, we compare the two cases. The experimental results are listed in Table 2.5.

It can be seen from Table 2.5 that the objective value obj^1_{route} with loading and unloading time decision is smaller than the objective value obj^2_{route} without loading and unloading time decision which shows an average 16.8% reduction of the total traveling distance. This indicates that the handling time decision can affect the subblock allocation

decision. This finding is reasonable because the allocation of the subblock can be made more flexible by adjusting the loading and unloading time. The subblocks near the berth are more likely to be allocated so that traffic congestions can be avoided. For the case without the handling time decision, traffic congestion can only be limited by allocating subblocks at the farther storage locations, resulting in longer traveling distances of trucks. From the computational time in Table 2.5, we note that when there is a handling time decision, the CETSA algorithm costs more time to allocate the loading and unloading time, which is more time-consuming than the case without the time decision.

Table 2.5: Comparison results between cases with and without handling time assignment

Instance		TIme decision(M1)		No time decision(M2)		
Group	id	z_1	Time(s)	z_2	Time(s)	Gap(%)
	40_1	58616975	1627	66635033	841	13.7
G8	40_2	61905672	1158	72683046	819	17.4
	40_3	61749857	1860	72349297	727	17.2
	45_1	69966238	1263	80258234	950	14.7
G9	45_2	73852968	1472	84629877	1044	14.6
	45_3	71495786	1348	82340865	780	15.2
	50_1	78970969	1428	95957772	1052	21.5
G10	50_2	80503207	1031	92199323	646	14.5
	50_3	77541588	1751	94661759	575	22.1
Avg.		70511473	1438	82412801	826	16.8

2.5.5 Effects of different route length strategies

In the related studies that minimize the total traveling distance of trucks, Zhen et al. (2011) uses the average distance between the middle berth position and all subblocks. That is, in model M1, they set $d^U_{ipk} = \bar{d}_{bk}$ and $d^L_{ipk} = \bar{d}_{bk}$. To test whether different route length calculation strategies impact the solution, we compare two route length calculation strategies based on test data groups G1-G10. The two strategies are as follows. (1) Strategy one uses the actual route length between berth b and subblock k, that is, the strategy used in this problem. (2) Strategy 2 uses the average route length $\bar{d}_{bk}$ used in

Zhen et al. (2016). That is, $d^U_{ipk} = d^L_{ipk} = \bar{d}_{bk} = (\sum_{k \in K} d_{\bar{b}k})/|K|$, where $\bar{d}_{bk}$ is the distance between the middle berth position $\bar{b}$ and a subblock k. The experimental results are listed in Table 2.6.

Table 2.6: Comparison results for different route length strategies

group	id	$route_1$	$route_2$	ratio	group	id	$route_1$	$route_2$	ratio
	5_1	4521096.88	10099224	2.2		30_1	40798891	67187248	1.6
G1	5_2	3056491.6	8139600	2.7	G6	30_2	42819791	70204032	1.6
	5_3	3767892.32	9429696	2.5		30_3	43522580	70686652	1.6
	10_1	10174326.7	20377184	2.0		35_1	51889654	85350720	1.6
G2	10_2	10037418.2	19346560	1.9	G7	35_2	52379265	85236480	1.6
	10_3	9218055.84	19394720	2.1		35_3	54708121	86819040	1.6
	15_1	18258720	31987872	1.8		40_1	58616975	97531540	1.7
G3	15_2	16778431.2	29512448	1.8	G8	40_2	61905672	1.02E+08	1.6
	15_3	15997485.7	28512096	1.8		40_3	61749857	1.05E+08	1.7
	20_1	23732833.5	40997880	1.7		45_1	69966238	1.18E+08	1.7
G4	20_2	24008084.2	41952328	1.7	G9	45_2	73852968	1.22E+08	1.6
	20_3	25400456.7	43587456	1.7		45_3	71495786	1.18E+08	1.7
	25_1	32765900.6	55137880	1.7		50_1	78970969	1.34E+08	1.7
G5	25_2	33084136.4	55098152	1.7	G10	50_2	80503207	1.39E+08	1.7
	25_3	32488916.2	53215656	1.6		50_3	77541588	1.38E+08	1.8
Avg.									1.8
Stdev.p									0.3

In Table 2.6, Column "$route1$" represents the objective value of the actual traveling distance of trucks calculated using strategy one. Column "$route_2$" represents the objective value of the approximate traveling distance of trucks calculated using strategy two. Column "gap%" represents the difference in percentage between $route_1$ and $route_2$ and is computed as $gap = (route_2 - route_1)/route_1 \times 100\%$. As can be seen from Column "gap%", the traveling distance computed with the average route length is larger than that computed with the actual distance. The average value of "gap%" is 1.8%. Besides, the standard deviation of values of "gap%" is 0.3, which indicates that when we use the average route length to estimate the actual route length, the results are likely to increase in a relatively consistent proportion.

2.6 Summary

In this chapter, we focus on yard space allocation and handling time assignment for transshipment vessels at a seaport yard. Vessels from different liners have a heterogeneous periodical pattern of arrival and departure. The problem is heterogeneous and periodical in nature in the time dimension, which aims to minimize the total traveling distance of yard trucks and the total time deviation of all vessels. The primary space allocation unit is a size-fixed subblock. To avoid traffic congestion in the yard, two important decisions are to be made, i.e. the allocation of subblock locations and the assignment of unloading and loading time of each subblock. Each transshipment vessel needs to unload to subblocks reserved for it and load containers from subblocks reserved for its source vessels. Therefore, when deciding the subblock location, it is necessary to determine both the locations of reserved subblocks for the unloading containers and the loading containers. Meanwhile, a second decision on the unloading and loading time for each subblock is also made to improve flexibility in the allocation of subblock locations.

We propose a comprehensive MIP model which determines both the location, unloading time and loading time of each subblock for both the unloading and loading process of a transshipment vessel. Since the two decisions are mutually related in the time dimension, the MIP model cannot be decomposed into independent sub-problems. The MIP model cannot be solved by the commercial solver or the column generation method. Besides, the four related decisions increase the magnitude of the number of variables of the model. Thus, we design a heuristic method to solve it. First, we divide the MIP model into two sub-models. Based on the solutions of the two sub-models, we propose a three-stage heuristic method based on the so-called critical element, that is, the CETSA algorithm. The numerical results verify the effectiveness of the CETSA algorithm. Meanwhile, by comparing different space allocation methods, we can see that the integrated allocation method proposed in this chapter can obtain a solution with a better objective value. In addition, we verify the effects of loading and unloading time decisions and the route length strategies on the total traveling distance of yard trucks.

CHAPTER 3

A NEW DYNAMIC SHAPE ADJUSTMENT AND PLACEMENT ALGORITHM FOR 3D YARD ALLOCATION PROBLEM WITH TIME DIMENSION

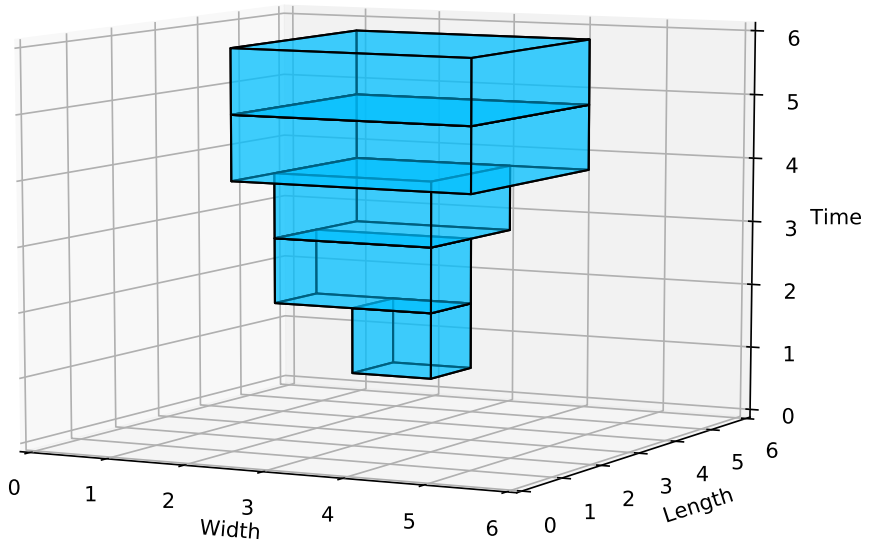
3.1 Introduction

Container terminals are large and critical components of complex logistic networks and are considered as strategic assets. Usually, export containers arrive at a container terminal through several modes of transportation, such as trucks and barges. Containers are dropped off at these terminals and are stored for a brief period of time. After this period, the containers are taken from the stack by cranes and loaded onto ships for delivery. This leads to the well-known yard allocation problem (YAP) faced by terminal operators (Fu et al., 2007; Jin et al., 2016; Zhen et al., 2016). Over the past few decades, technology has facilitated automation of container handling at terminals with tactical-level coordination in respect of allocation of terminal yard blocks, assignment of quay crane jobs and routes for the AGVs (automated guided vehicles) that travel between the quay sides and the yard blocks. Meanwhile, new decision support systems have also helped lower-level operations that determine whether and how containers should be stored within a specific yard block with limited space.

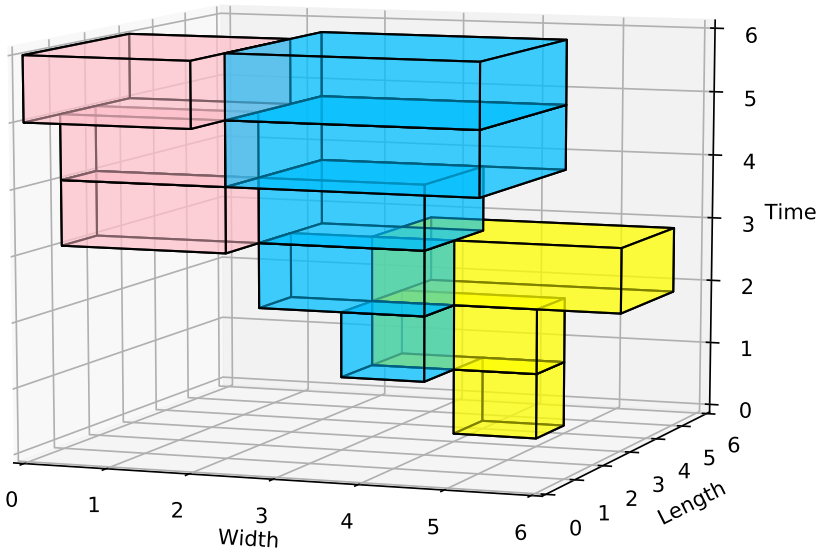
This study focuses on yard space allocation for export containers. We study the 3D yard allocation problem with time dimension (3DYAPT), which was first introduced in Fu et al. (2007). The 3DYAPT is defined for a yard block, simply referred to as a yard in the rest of this study, served by a few quay cranes and is assumed to be empty at the start of the planning horizon. A request is defined as the requirement from several batches of containers to be transported to the same destination. Each request has a

storage time requirement and needs a certain set of yard space. Containers of the same request are usually stored adjacent to each other, and therefore containers in each request occupy a particular rectangular storage space. This rectangular storage space is not predetermined and is decided on arrival of the containers of different batches at different arrival times. From a time perspective, the area of this rectangular storage space should be non-decreasing, i.e., during the arrival of container batches, the area of the rectangular storage space either increases or remains unchanged as time progresses until all containers in the particular request are loaded for delivery to the same destination. As an example, Figure 3.1a shows a storage solution to one valid request that spans from time point 2 to 6. It can be seen that the rectangular storage space increases from time point 2 to 5 because more containers to be taken to the same destination arrive during this period. From time point 5 to 6, the storage space remains unchanged. This is possible because newly arrived containers can utilize the unused space in the storage solution up to time point 5. It is also easy to see that the rectangular storage spaces of different requests should be non-overlapping. For example, Figure 3.1b shows a feasible storage solution to three requests where there is no storage space overlap. To optimize space utilization, the objective of the 3DYAPT is to minimize the largest storage space ever used throughout the planning horizon.

In this work, we consider a more general rectangular storage space in the 3DYAPT. Unlike Fu et al. (2007), we relax the square shape restriction on storage spaces, allowing rectangular shapes with adjustable lengths and widths. The idea is that rectangular storage shapes with adjustable lengths and widths offer much larger flexibility when fitting them into narrow spaces, compared to squares, thus having great potential to generate more efficient yard allocation solutions. We prove that with the square restriction relaxed, the problem is still NP-Complete. We present a new integer linear programming formulation, and then propose an effective approach based on the simulated annealing algorithm and a dynamic shape adjustment placement procedure (SA-DSAP) that packs rectangular storage spaces into the yard to minimize the largest occupied area over the planning horizon. Our SA-DSAP consists of two stages. In the first stage, sequences of



(a) A valid request



(b) Three valid requests

Figure 3.1: Examples of the three-dimensional requests

all requests are generated by SA. In the second stage, given a particular sequence, the storage space solution is computed using a dynamic programming (DP) procedure. Because, the DP procedure is to be frequently invoked, to further improve computational

efficiency, we also develop some speed-up techniques eliminating storage spaces that are unlikely to be included in an optimal solution before they are evaluated by DP. Extensive computational experiments show that SA-DSAP is able to obtain optimal solutions for small-scale instances, and the solutions for large-scale instances are of much higher quality than the Bottom-Left heuristic used in Fu et al. (2007).

The remainder of this study is organized as follows. In Section 3.2, we present the problem in greater detail and formulate the problem as an integer linear programming model. In Section 3.3, the general framework of the SA-DSAP is described, followed by details of its most critical component, the DSAP, in Section 3.4. Experiments and computational results are discussed in Section 3.5, and conclusions are drawn in Section 3.6.

3.2 Problem Description and Formulation for the 3DYAPT

3.2.1 Problem description

To describe the 3DYAPT, let us consider a rectangular terminal yard that consists of $W \times L$ rectangular slots, where W and L represent the number of slots contained by the width and length of the yard, respectively. In each slot, it is possible to stack at most H standard containers. For ease of description, the rectangular yard area can be represented by the area $(0, 0, W, L)$, with coordinates $(0, 0)$ indicating the left-bottom corner point and the coordinates (W, L) indicating the right-top corner point.

The considered time horizon T is composed of a series of evenly distributed discrete time points $T = \{1, 2, \ldots, |T|\}$. The arrival and departure of containers occur at a given time point. In case there are both arrival and departure during the same time unit, we assume that departure containers are handled first and then the arrival containers.

3.2.1.1 Request

We are given a set of Q containers whose arrivals and departures happen within time points in T. Containers for the same destination are collected in a group which

we call as a *request*. Note that each container is included in exactly one request, and containers belonging to a request leave the yard area at the same departure time. Let $R = \{1, 2, \ldots, N\}$ $(N = |R|)$ denote the set of requests, where request $r \in R$ consists of n_r containers $(\sum_{r \in R} n_r = Q)$, and the departure time of each container involved is denoted by $t_r \in T$. We assume that the loading process of containers in a request can be completed in a short period, and the space occupied by request r can be released fully only after its departure time t_r. The arrival time of request r (denoted by a_r) is defined as the earliest arrival time among all arriving containers in request. Thus, we can define $T_r = \{a_r, a_r + 1, \ldots, t_r\}$ a subset of T, to represent the discretized time points occupied by request r. $n_{r,t}$ is defined as the total number of arrived containers in request r by time t. The storage space required by request r at time point t can be determined by $n_{r,t}$. t_r is a known input data which can be obtained from the output by Iris et al. (2018). Because containers may arrive at any time point $t \in T_r$ and are stored in the yard up to the delivery time of the request, $n_{r,t}$ must be non-decreasing, i.e., $n_{r,a_r} \leq n_{r,a_r+1} \leq \ldots \leq n_{r,t_r} = n_r$.

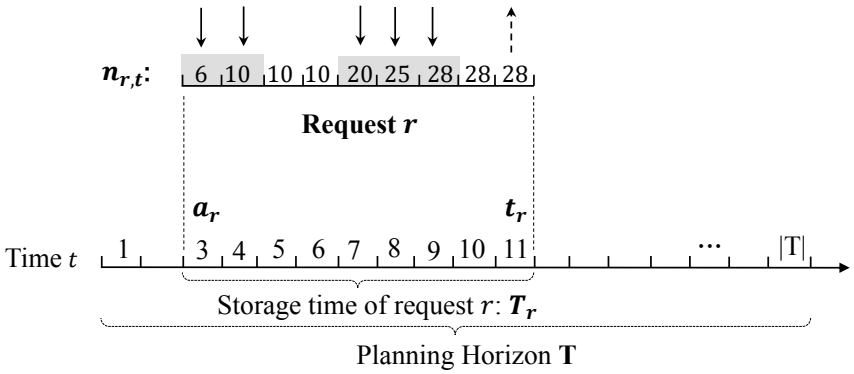


Figure 3.2: Illustration of a request

Figure 3.2 shows an example of handling of a given request r within planning horizon T. Containers in this request r are stored in the yard from time point 3 $(a_r = 3)$ to time point 11 $(t_r = 11)$. In the beginning, 6 containers arrive at time point 3 followed by 4 more containers at time point 4. No container arrives between time points 5 and 6. At time points 7, 8 and 9, 10, 5 and 3 containers arrive at the yard respectively. The accumulated number of arrived containers $n_{r,t}$ from time point 3 to 11 is depicted in Figure 3.2. We can see that, request r has a total of 28 arrived containers $(n_r = 28)$ and all of them are

released together at time point 11.

3.2.1.2 Feasible storage space

Given a request $r \in R$, we need to determine a rectangular space within the yard area to store the containers of request r, covering all time points in T_r. We denote $s_{r,t} = (x^-_{r,t}, y^-_{r,t}, x^+_{r,t}, y^+_{r,t})$ as the rectangular storage space at time $t \in T_r$, where $(x^-_{r,t}, y^-_{r,t})$ and $(x^+_{r,t}, y^+_{r,t})$ are coordinates of the left-bottom and the right-top corner points, respectively. At time t, the storage space $s_{r,t}$ is feasible when the following four conditions are satisfied.

(1) The rectangular storage space $s_{r,t}$ is fully contained in yard area, i.e., $0 \leq x^-_{r,t} < x^+_{r,t} \leq W$ and $0 \leq y^-_{r,t} < y^+_{r,t} \leq L$ must be satisfied simultaneously;

(2) The area occupied by storage space $s_{r,t}$ has enough slots for request r accumulated at time point t. This requirement can be expressed by $n_{r,t} \leq H(x^+_{r,t} - x^-_{r,t})(y^+_{r,t} - y^-_{r,t})$, where H is the maximum number of containers that can be stacked for each unit slot.

(3) Due to restraints imposed on yard cranes, the width $(x^+_{r,t} - x^-_{r,t})$ and the length $(y^+_{r,t} - y^-_{r,t})$ of a storage space need to be bounded by prescribed ranges $[\underline{w}, \overline{w}]$ and $[\underline{l}, \bar{l}]$, i.e., $\underline{w} \leq x^+_{r,t} - x^-_{r,t} \leq \overline{w}$ and $\underline{l} \leq y^+_{r,t} - y^-_{r,t} \leq \bar{l}$.

(4) To restrict the shape of the rectangular storage space, the difference between length $l_{r,t}(= y^+_{r,t} - y^-_{r,t})$ and width $w_{r,t}(= x^+_{r,t} - x^-_{r,t})$ of a feasible storage space $s_{r,t}$ should be within a predetermined range d, i.e., $|l_{r,t} - w_{r,t}| \leq d$. It then should be noted that the square storage space case presented in Fu et al. (2007) corresponds to the case when $d = 0$, i.e., $|l_{r,t} - w_{r,t}| \leq 0$, or $l_{r,t} = w_{r,t}$.

As shown in Figure 3.1a, there is a request r from time point 2 to 6, and at each time point, there is a corresponding three-dimensional feasible storage space. The yard area is assumed to be (0,0,6,6) and the height limit is 6 units, i.e., $L = 6$, $W = 6$ and $H = 6$. At time point 3, we assume there are 11 arriving containers ($n_{r,3} = 11$). Then, a feasible storage space at time point 3 is drawn as (2,2,3,3), ranging from the left-bottom point (2,2) to the right-top point (3,3). This storage space is fully contained in the yard and is large enough to accommodate 12 ($l \times w \times h$=(4-2)×(3-2)×6) containers at a time.

Finally, the assumed length and width restrictions, $[\underline{w}, \overline{w}] = [1, 6]$ and $[\underline{l}, \overline{l}] = [1, 6]$, are also satisfied since the storage space has a length of 2 units and a width of 1 unit.

Because there exist different storage arrangements for the $n_{r,t}$ containers at each time t, we use $S_{r,t}$ to denote the set of all feasible storage spaces for request r at time t. Using an approach similar to Buhrkal et al. (2011) and Iris et al. (2015), we simply generate $S_{r,t}$ for all requests at different time points a priori.

3.2.1.3 Constraints and objective

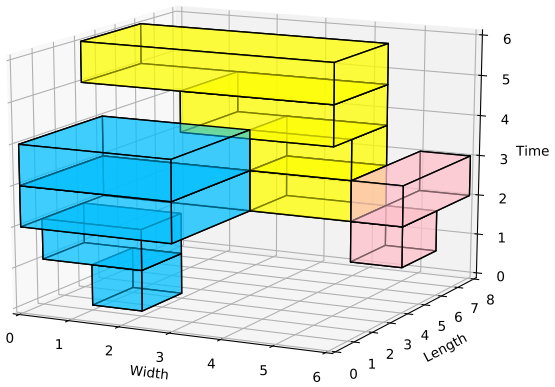
Let $\mathbf{s} = \{\mathbf{s}_1, \ldots, \mathbf{s}_N\}$ denote a storage solution vector to all requests, where $\mathbf{s}_r = \{s_{r,a_r}, \ldots, s_{r,t_r}\}$ $(s_{r,t} \in S_{r,t}, r \in R, t \in T_r)$ is the storage solution to request r. We aim to obtain a feasible storage solution vector $\mathbf{s}$ by allocating exactly one feasible storage space $s_{r,t}$ for each request r and time point t, such that the following constraints are satisfied:

(1)*Non-decreasing constraints.* As discussed earlier, for each request r, the rectangular spaces developed from time a_r to t_r must expand continuously to cover the arriving containers. Thus, given request $r \in R$ and given each two consecutive time points $t, t+1 \in T_r$, the two feasible storage spaces, $s_{r,t}$ and $s_{r,t+1}$, must satisfy $x^-_{r,t} \geq x^-_{r,t+1}$, $x^+_{r,t} \leq x^+_{r,t+1}$, $y^-_{r,t} \geq y^-_{r,t+1}$, and $y^+_{r,t} \leq y^+_{r,t+1}$, simultaneously;

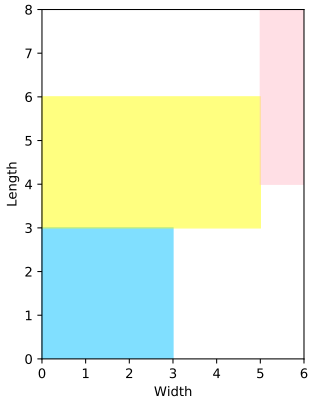
(2)*Non-overlapping constraints.* These constraints stipulate that the feasible storage spaces of two different requests cannot share any area in the yard at any time points in T. Mathematically, given time $t \in T$, two requests r and r' such that $t \in T_r \cap T_{r'}$, the feasible storage spaces, $s_{r,t}$ and $s_{r',t}$, must satisfy at least one of the following: (i) request r' is completely on the left side of request r, i.e., $x^-_{r',t} \geq x^+_{r,t}$; (ii) request r' is completely on the right side of request r, i.e., $x^+_{r',t} \leq x^-_{r,t}$; (iii) request r' is completely above request r, i.e., $y^-_{r',t} \geq y^+_{r,t}$; and (iv) request r' is completely under request r, i.e., $y^+_{r',t} \leq y^-_{r,t}$.

In Figure 3.1b, the storage solution to the three requests is feasible. For each request, its feasible storage spaces increase or keep unchanged, which is in accord with the non-decreasing constraints. The feasible storage spaces do not overlap at each time point.

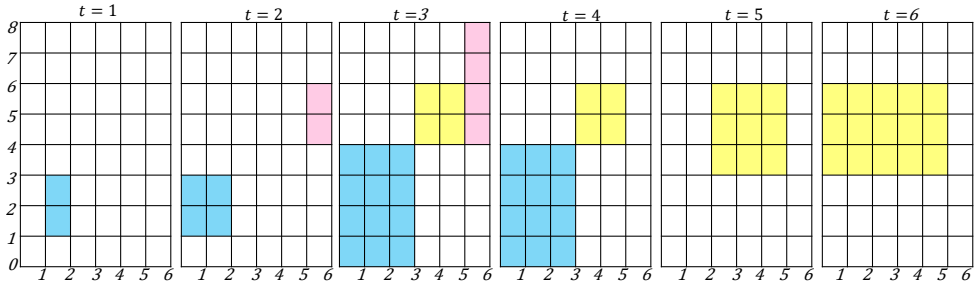
The 3DYAPT is then described as follows. Given the incoming containers grouped



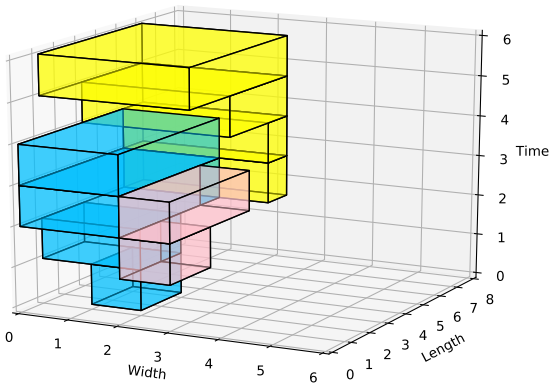
(a) Feasible solution



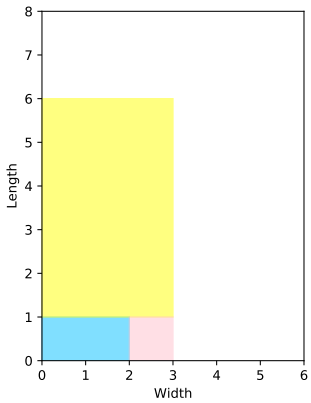
(b) Feasible area



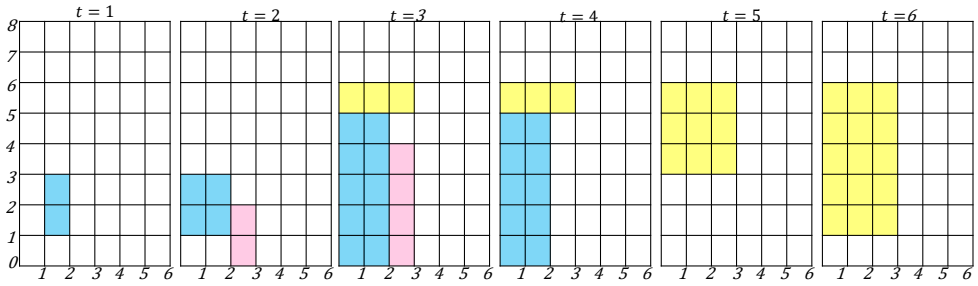
(c) 2D views of Figure 3.3a



(d) Optimal solution



(e) Optimal area



(f) 2D views of Figure 3.3d

Figure 3.3: Illustration of storage space solutions to three requests

by different requests in R, the arrival and departure time points $a_r, t_r \in T_r$ for each request $r \in R$, and the sets $S_{r,t}$ $(r \in R, t \in T_r)$ of feasible storage spaces, the problem aims to select a series of feasible storage spaces $s_{r,t}$ $(r \in R, t \in T_r)$, such that the determined storage solution vector $\mathbf{s} = \{(s_{1,a_1}, \ldots, s_{1,t_1}), \ldots, (s_{N,a_N}, \ldots, s_{N,t_N})\}$ satisfies both the non-decreasing constraints and the non-overlapping constraints. Our objective, which follows the definition of objective in Fu et al. (2007), is to minimize the area of the minimum rectangular cover s_C so as to maximize the utility of yard space. This minimum rectangular cover s_C can be represented by $(x_C^-, y_C^-, x_C^+, y_C^+)$, where $x_C^- = \min_{r \in R, t \in T_r} x_{r,t}^-$, $y_C^- = \min_{r \in R, t \in T_r} y_{r,t}^-$, $x_C^+ = \max_{r \in R, t \in T_r} x_{r,t}^+$, and $y_C^+ = \max_{r \in R, t \in T_r} y_{r,t}^+$.

Figure 3.3a illustrates a feasible storage solution for the three requests. Figure 3.3b shows the rectangular cover associated with the storage solution with an area of 48 (with=6, length=8). It can be observed that this rectangular cover is not optimal. It should also be noted that in 3DYAPT, containers are not required to be piled along the block side because it is hard to maintain such a requirement when requests keep arriving and getting released during the whole planning horizon. In Figure 3.3d, the optimal storage solution to the three requests is shown. The area of the optimal rectangular cover (Figure 3.3e) is reduced to 18 (width=3, length=6).

3.2.2 Formulation

We now formulate the 3DYAPT as an integer linear programming model as follows.

3.2.2.1 Notations

Indices:

r: request, $r \in R$;

t: time point, $t \in T$;

p: position in the yard, $p \in P$;

s: rectangular storage space, $s := s_{r,t} \in S_{r,t}$. In the following passage, we use s to represent a storage solution $s_{r,t}$ as aforementioned.

Sets:

R: set of requests, $R = \{1, 2, \ldots, N\}$;

T: set of time points of the planning horizon, $T = \{1, 2, \ldots, |T|\}$;

T_r: set of time points of request r, $T_r = \{a_r, a_r + 1 \ldots, t_r\}$, $r \in R$;

P: set of all positions in the yard, $P = \{(x_p, y_p) : x_p \in \{1, 2, \ldots, W\}, y_p \in \{1, 2, \ldots, L\}\}$;

$S_{r,t}$: set of candidate storage solutions for request r at time point t, $S_{r,t} = \{s_{r,t}\} = \{s\}$, $r \in R$, $t \in T_r$;

R_t: set of requests that requires storage space at time point t, $R_t = \{r | r \in R, t \in T_r\}$, $R_t \subset R$, $t \in T$.

Parameters:

x_p: x-coordinate of position p;

y_p: y-coordinate of position p;

$\theta_{p,s}$: equals one if position p is in rectangular storage space s; otherwise, equals zero, $p \in P$, $s \in S_{r,t}$.

Decision variables:

$\pi_p \in \{0, 1\}$: set to one if position p is the point at the right-top corner of rectangular cover s_C, and zero otherwise, $p \in P$. $s_C = (x_C^-, y_C^-, x_C^+, y_C^+)$ is the minimum rectangular cover as defined in Subsection 3.2.1.3.

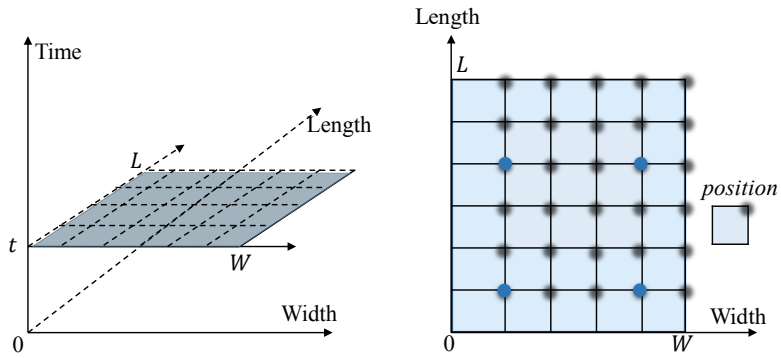
$\lambda_{r,t,s} \in \{0, 1\}$: set to one if the rectangular storage space selected for request r at time t is s, and zero otherwise, $r \in R$, $t \in T_r$, $s \in S_{r,t}$.

Here, P is the set of positions in the yard. Because the yard itself is a two-dimensional rectangular area from left-bottom point $(0, 0)$ to right-top point (W, L) (Figure 3.4a), considering the unit container size, coordinates for all occupied rectangular areas can be defined as integers. Furthermore, we use the right-top corner point of a rectangular area to indicate its position in the yard (Figure 3.4b). Thus, set P is denoted as $P = \{(x_p, y_p) : x_p \in \{1, 2, \ldots, W\}, y_p \in \{1, 2, \ldots, L\}\}$. It is obvious that the right-top corner point (x_s^+, y_s^+) of any feasible storage space s belongs to set P.

R_t is the set of requests that are stored in the yard at time point t, i.e., $R_t = \{r :$

$r \in R, a_r \leq t \leq t_r\}$. Hence, R_t is a subset of the overall request set R, i.e., $R_t \subset R$. $\theta_{p,s}$ is an indicator which equals 1 when position p is covered by storage space s. Recall that $s = (x_s^-, y_s^-, x_s^+, y_s^+)$. The value of $\theta_{p,s}$ can be obtained from coordinates $p = (x_p, y_p)$ and coordinates of the left-bottom and right-top corner points (x_s^-, y_s^-), (x_s^+, y_s^+) of s as follows:

$$\theta_{p,s} = \begin{cases} 1 & \text{if } x_s^- < x_p \leq x_s^+, y_s^- < y_p \leq y_s^+, \\ 0 & \text{otherwise.} \end{cases}$$



(a) (b)

Figure 3.4: Illustration of positions

3.2.2.2 Mathematical model

The objective of the 3DYAPT is to minimize the area of the rectangular cover s_C for all requests R in time horizon T. As defined, the rectangular cover is $s_C = (x_C^-, y_C^-, x_C^+, y_C^+)$, and the points at the left-bottom and the right-top corners of s_C are $p^- = (x_C^-, y_C^-)$ and $p = (x_C^+, y_C^+)$ respectively. Noting that the left-bottom point p^- is fixed to $(0, 0)$ by minimization, we then can formulate the objective with only the right-top point p ($p \in P$) as follows:

$$\min \sum_{p \in P} x_p y_p \pi_p. \tag{3.1}$$

Objective (3.1) minimizes the yard space ever used by all requests during the whole planning horizon.

Each request has to satisfy the non-decreasing constraint, that is its rectangular storage space increases or keeps unchanged with time. Considering two consecutive time

points t and $t+1$ of request r ($r \in R, t \in \{1,\ldots,t_r-1\}, t+1 \in \{2,\ldots,t_r\}$), we have two consecutive rectangular spaces s and s'. The equation $\sum_{s\in S_{r,t}} \lambda_{r,t,s} = 1$ ($r \in R, t \in T_r$) guarantees that one feasible storage space is exactly assigned to a request at a time point. To ensure that the storage space s assigned to request r at time t is occupied until departure time t_r, the storage space s' at time $t+1$ should be larger or at least equal to s. That is, the left edge of s' should be left to that of s, the right edge of s should be right to that of s, the bottom edge of s' should be lower than that of s, and the top edge of s' should be higher than that of s. So the non-decreasing constraints are formulated as follows:

$$\sum_{s\in S_{r,t}} \lambda_{r,t,s} = 1, \quad r \in R, t \in T_r, \tag{3.2}$$

$$\sum_{s'\in S_{r,t+1}} x^-_{s'}\lambda_{r,t+1,s'} \leq \sum_{s\in S_{r,t}} x^-_s \lambda_{r,t,s}, \quad r \in R, t \in T_r \setminus \{t_r\}, \tag{3.3}$$

$$\sum_{s'\in S_{r,t+1}} y^-_{s'}\lambda_{r,t+1,s'} \leq \sum_{s\in S_{r,t}} y^-_s \lambda_{r,t,s}, \quad r \in R, t \in T_r \setminus \{t_r\}, \tag{3.4}$$

$$\sum_{s'\in S_{r,t+1}} x^+_{s'}\lambda_{r,t+1,s'} \geq \sum_{s\in S_{r,t}} x^+_s \lambda_{r,t,s}, \quad r \in R, t \in T_r \setminus \{t_r\}, \tag{3.5}$$

$$\sum_{s'\in S_{r,t+1}} y^+_{s'}\lambda_{r,t+1,s'} \geq \sum_{s\in S_{r,t}} y^+_s \lambda_{r,t,s}, \quad r \in R, t \in T_r \setminus \{t_r\}, \tag{3.6}$$

Constraints (3.2) ensure that one storage space is assigned to a request at a time. Constraints (3.3)-(3.6) are the non-decreasing constraints ensuring that the storage space for each request either increases or remains unchanged during its storage time interval.

Constraints (3.7) are the non-overlapping constraints ensuring that at a given time, each storage space can be used by at most one request. According to the non-overlapping constraint, at time point t, any two different requests r and r' ($r, r' \in R_t$) cannot occupy the same space simultaneously. Thus, the sets of positions of their storage spaces $s_{r,t}$ and $s_{r',t}$ should be disjoint. That is, at time point t, for any position $p \in P$, it must satisfy either of the following two conditions: 1) p is covered by $s_{r,t}$ or $s_{r',t}$, but not both; 2) p is neither covered by $s_{r,t}$ nor $s_{r',t}$. Then the non-overlapping constraints are formulated as

follows:

$$\sum_{r\in R_t}\sum_{s\in S_{r,t}:\theta_{p,s}=1}\lambda_{r,t,s}\leq 1,\quad t\in T, p\in P, \tag{3.7}$$

where the left-side expression represents the total number of times that point p is covered by all rectangular storage spaces at time t that cover point p.

$$x_s^+\lambda_{r,t,s}\leq\sum_{p\in P}x_p\pi_p,\quad r\in R, t\in T_r, s\in S_{r,t}, \tag{3.8}$$

$$y_s^+\lambda_{r,t,s}\leq\sum_{p\in P}y_p\pi_p,\quad r\in R, t\in T_r, s\in S_{r,t}, \tag{3.9}$$

$$\sum_{p\in P}y_p\pi_p\leq L, \tag{3.10}$$

$$\sum_{p\in P}x_p\pi_p\leq W, \tag{3.11}$$

$$\sum_{p\in P}\pi_p=1, \tag{3.12}$$

Constraints (3.8) and (3.9) ensure that the rectangular cover s_C is valid to cover the storage spaces of all requests during the whole planning horizon. Constraints (3.10)-(3.11) guarantee that all requests are stored within the yard area. Constraint (3.12) determines the size of the rectangular cover s_C. The left-bottom corner point of rectangular cover s_C is restricted to point $(0,0)$ by Constraints (3.8)-(3.9) and Objective (3.1).

$$\lambda_{r,t,s}\in\{0,1\},\quad r\in R, t\in T_r, s\in S_{r,t}, \tag{3.13}$$

$$\pi_p\in\{0,1\},\quad p\in P, \tag{3.14}$$

Constraints (3.13)-(3.14) define integral decision variables. As a result, we obtain a mathematical model **M1** with objective (3.1) and constraints (3.3)-(3.14) for the 3DYAPT.

As mentioned in Subsection 3.2.1.2, it is easy to see that the squared storage space case presented in Fu et al. (2007), which is strongly NP-hard, is a special case of the 3DYAPT. Therefore, the 3DYAPT is strongly NP-hard.

3.3 Solving the 3DYAPT by Simulated Annealing

In this section, we present a simulated annealing (SA)-based heuristic solution approach for the 3DYAPT. The main component of this solution approach is a dynamic programming algorithm based on the dynamic shape adjustment placement (DSAP) procedure. DSAP determines the optimal storage spaces for arriving containers from a set of feasible storage spaces.

According to the model presented in Section 3.2, the 3DYAPT consists of two decisions: (1) the storage shape-related decision, and (2) the storage position-related decision for each request at each storage time point. Both decisions are represented by the binary variables $\lambda_{r,t,s}$, $(r \in R, t \in T_r, s \in S_{r,t})$. Other decision variables π_p $(p \in P)$ regarding the objective value are dependent on $\lambda_{r,t,s}$. In our heuristic solution approach, decision $\lambda_{r,t,s}$ is determined by deciding a feasible storage space $s_{r,t} \in S_{r,t}$ for each request r at time point t. We present each feasible solution to our 3DYAPT by a storage solution vector $\mathbf{s} = \{\mathbf{s}_1, \ldots, \mathbf{s}_N\}$ where $\mathbf{s}_r = \{s_{r,a_r}, \ldots, s_{r,t_r}\}$ $(r \in R, s_{r,t} \in S_{r,t}, t \in T_r)$ is composed of the decided feasible storage spaces as defined in Section 3.2.1.

3.3.1 Preprocessing: generating set $S_{r,t}$

We first introduce the preprocessing step which converts the accumulated number of arrived containers $n_{r,t}$ $(r \in R, t \in T_r)$ into a set of feasible storage spaces $S_{r,t}$ for each request r and time point t. These feasible storage spaces in set $S_{r,t}$ are to be directly used in the solution approach. We define parameter $\Omega_{r,t}^{min}$ as the minimum yard area needed to store $n_{r,t}$ containers for request r at time t, which is computed as $\Omega_{r,t}^{min} = \lceil n_{r,t}/H \rceil$. Accordingly, the set of feasible storage spaces $S_{r,t}$ can be generated by an enumeration procedure, which enumerates all feasible storage spaces to satisfy the following three conditions: (1) each feasible storage space $s_{r,t}$ should not exceed the yard limits; (2) the area of a storage space $s_{r,t}$ should be no less than $\Omega_{r,t}^{min}$; and (3) each $s_{r,t} \in S_{r,t}$ should not be dominated by any other storage space in set $S_{r,t}$.

Algorithm 5 Framework of SA-DSAP

Procedure: SA-DSAP($n_{r,t}, R, W, L$)

Preprocessing: convert $n_{r,t}$ to set $S_{r,t}$ ($r \in R, t \in T_r$);
Generate an initial sequence p_0 of the N requests;
$\mathbf{s}_0, \Omega_0 \leftarrow$ DSAP(p_0, R, W, L);
Let $p, p^* \leftarrow p_0$, $\Omega, \Omega^* \leftarrow \Omega_0$ and $\mathbf{s}, \mathbf{s}^* \leftarrow \mathbf{s}_0$;
Generate initial temperature $\tau_0 = \alpha \times \Omega$;
Let $\tau = \tau_0$;
while $\tau \geq 0.001$ **do**
  Add τ to temperate schedule $\mathcal{T}$;
  $\tau = \beta \times \tau$;
end while
for each temperature τ in $\mathcal{T}$ **do**
  Let $K = K_{max}, I = 0$;
  while $K > 0$ **do**
    Call a two-swap operator to generate p' from p;
    $\mathbf{s}', \Omega' \leftarrow$ DSAP(p', R, W, L);
    if $\Omega' < \Omega^*$ **then**
      $I = 0$
    else
      $I = I + 1$
    end if
    if $I \geq I_{max}$ **then**
      break;
    end if
    if $\Omega' < \Omega$ **then**
      $p \leftarrow p'$, $\Omega \leftarrow \Omega'$ and $\mathbf{s} \leftarrow \mathbf{s}'$;
      if $\Omega < \Omega^*$ **then**
        $p^* \leftarrow p$, $\Omega^* \leftarrow \Omega$ and $\mathbf{s}^* \leftarrow \mathbf{s}$;
      end if
    else
      if $random[0, 1] < exp[(\Omega - \Omega')/\tau]$ **then**
        $p \leftarrow p'$, $\Omega \leftarrow \Omega'$ and $\mathbf{s} \leftarrow \mathbf{s}'$;
      end if
    end if
  end while
end for
return $\mathbf{s}^*, \Omega^*$

3.3.2 General framework of SA-DSAP

The simulated annealing based dynamic shape adjustment placement algorithm, namely SA-DSAP, is presented in Algorithm 5. It iteratively explores and adjusts sequences of requests within the SA framework. The sequence of requests is the sequence for allocating yard space. For each sequence of requests explored, it applies the dynamic shape adjustment placement algorithm (DSAP) to determine the feasible storage solutions $\mathbf{s}_r$ ($r \in R$) for every single request, one by one. For each explored sequence p of requests, a feasible storage solution vector $\mathbf{s} = \{\mathbf{s}_1, \ldots, \mathbf{s}_N\}$ and its relevant objective value a are computed by the DSAP algorithm. Details of the DSAP are described in Section 3.4. The results obtained by DSAP are utilized by the simulated annealing for further exploration of new sequences of requests.

We generate an initial sequence p_0 of requests by the following two rules: (1) Generate the sequence in decreasing order in terms of the number of containers since request with more containers usually takes more storage space; (2) For two or more requests with the same number of containers, the sequence is generated in increasing order of the size of set $S_{r,t}$. The intuition behind the rules is that requests with smaller sizes are considered as having less flexibility in determining the required storage space. Hence, they are placed in the front of the sequence p.

In SA-DSAP, we adopt a SA-based framework to iteratively adjust the sequence of requests. The initial temperature is set to the product of parameter α (=0.382) and the objective value of the initial sequence of requests. Each following temperature is set to the current temperature value multiplied by a decaying parameter β (=0.618). The annealing process stops when the temperature drops below 0.001. For each temperature τ, we apply a random two-swap operator to sequence p to get a neighbor p'. The storage solution vector of p' is accepted when its objective Ω' outperforms the objective Ω of current sequence p ($\Omega' < \Omega$). When p' is not better than p ($\Omega' \geq \Omega$), it is still to be accepted with a probability $exp[(\Omega - \Omega')/\tau]$. At each temperature, the number of iterations is limited to K_{max} (=500) times. To balance search efficiency and solution

quality, this searching process is terminated if the objective value has no improvement after I_{max} (=400) consecutive iterations.

3.4 Detailed Design of DSAP

3.4.1 Main idea

Given a sequence of requests $p = \{p_1, p_2, \ldots, p_N\}$, DSAP determines storage solutions $\{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_N\}$ for requests one by one, where $\mathbf{s}_i$ is the storage solution to request r ($r = p_i$) and is composed of a series of storage space decisions at each of its handling time point t, i.e., $\mathbf{s}_i = \{s_{r,t} : r = p_i, a_r \leq t \leq t_r\}$.

As shown in Algorithm 6, DSAP solves the N requests sequentially by using N iterations following sequence p. Each iteration determines the storage solution for a request by minimizing the yard area used up to this request. In each iteration i, the algorithm takes the storage solutions $\mathbf{s}_1, \ldots, \mathbf{s}_{i-1}$ of its preceding requests as input, and then generates the storage solution $\mathbf{s}_i$ for request p_i. To make the algorithm consistent, we use a dummy solution $\mathbf{s}_0$ to represent the empty yard before request p_1 is assigned. After solving the ith iteration, the obtained storage solution $\mathbf{s}_i$ of request p_i is saved before proceeding to the next iteration, and the available yard spaces at each time point are updated (Line 12). At each time t, we use a $W \times L$ binary matrix to indicated two-dimensional yard status. Each element of the binary matrix represents a container space unit, which is set to 1 if the space is occupied at time t and set to 0 otherwise. In DSAP, once the storage spaces s_i for the ith request in sequence p is determined, the yard space status in then updated by setting all the space units of s_i to 1.

In the ith iteration, storage solution $\mathbf{s}_i$ is deduced by selecting a feasible storage solution which leads to the minimum rectangular yard area a_i used by requests $p_1, \ldots, p_i$. The area a_i is computed as $L_i \times W_i$, where L_i and W_i are the yard length and yard width ever reached by these requests. We use a three-step procedure to generate $\mathbf{s}_i$. First, the used width W_i is artificially fixed to a feasible value w in range $[W_{i-1}, W]$, where W_{i-1} is the yard width reached by requests $p_1, \ldots, p_{i-1}$ and W is the yard width boundary. The

Algorithm 6 Generating storage solutions by DSAP given sequence p

Procedure: DSAP(p, R, W, L)

1: $L_0 = 0, W_0 = 1$
2: $\mathbf{s}_0$: empty yard
3: **for** $i \leftarrow 1$ **to** N **do**
4: **for** $w \leftarrow W$ **to** W_{i-1} **do**
5: $L_i(w), \mathbf{s}_i(w) \leftarrow$ SolveRequest$(i, p, R, w, L_{i-1}, \{\mathbf{s}_0, \ldots, \mathbf{s}_{i-1}\})$
6: $\Omega_i(w) = L_i(w) \times w$
7: **end for**
8: $\Omega_i = \min_{W_{i-1} \le w \le W} \Omega_i(w)$
9: $W_i = \arg \min_{W_{i-1} \le w \le W} \Omega_i(w)$
10: $L_i = L_i(W_i)$
11: $\mathbf{s}_i = \mathbf{s}_i(W_i)$
12: UpdateYardStatus
13: **end for**
14: $\mathbf{s} = \{\mathbf{s}_1, \ldots, \mathbf{s}_N\}$, $\Omega = \Omega_N$
15: **return** $\mathbf{s}, \Omega$

value w is reversely looped from W to W_{i-1} such that yard space is assigned according to width before according to length (Line 4). Second, given that W_i is fixed to value w, the yard length $L_i(w)$ ever reached by requests $p_1, \ldots, p_i$ is solved by the `SolveRequest` procedure based on dynamic programming. At the same time, the storage solution $\mathbf{s}_i(w)$ is obtained (Line 5). The area used by $\mathbf{s}_1, \ldots, \mathbf{s}_{i-1}, \mathbf{s}_i(w)$ is computed as $\Omega_i(w) = L_i(w) \times w$ (Line 6). Finally, a_i, W_i, L_i and storage solution $\mathbf{s}_i$ for request p_i are derived according to the w value that leads to the minimum area as W_i, i.e., $W_i = \arg\min_{W_{i-1} \le w \le W} \Omega_i(w)$ (Line 8~11). After N iterations, the storage solutions $\{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_N\}$ and yard area used Ω ($\Omega = \Omega_N$) following sequence p can be obtained (Line 14).

3.4.2 Computing the minimum used length

As mentioned in Line 5 of Algorithm 6, the `SolveRequest` procedure computes the yard length $L_i(w)$ ever reached by requests $p_1, \ldots, p_i$ when storage solutions of requests $p_1, \ldots, p_{i-1}$ are fixed to $\mathbf{s}_1, \ldots, \mathbf{s}_{i-1}$ and the yard width W_i reached by requests $p_1, \ldots, p_i$ is fixed to w. In this subsection, we present details of the dynamic programming (DP)

procedure in the `SolveRequest`, illustrating how the storage solution $\mathbf{s}_i(w)$ for request r ($r = p_i$) is determined (Algorithm 7).

Algorithm 7 Generating solution $\mathbf{s}_i$ to the ith request

Procedure: SolveRequest$(i, p, R, w, L_{i-1}, \{\mathbf{s}_0, \ldots, \mathbf{s}_{i-1}\})$

1: Get information of request r: $a_r, t_r, S_{r,t}$
2: **for** stage $k = |T_r|$ **to** 1 **do**
3: **for** decision $s \in S_k$ **do**
4: Computing state value $h(k, s)$
5: **end for**
6: **end for**
7: $L_i(w) = \min_{\forall s \in S_1} h(1, s)$
8: $\mathbf{s}_i(w) = \{s_1^*, \ldots, s_{|T_r|}^*\} \leftarrow \arg\min_{s \in S_1} h(1, s)$
9: **return** $L_i(w), \mathbf{s}_i(w)$

Definition of stage. Since request r covers $|T_r|$ ($|T_r| = t_r - a_r + 1$) time points, we define $|T_r|+1$ stages in the dynamic programming, that is, from stage 0 to stage $|T_r|$. While stage 1 to stage $|T_r|$ correspond to time point a_r to t_r, stage 0 is a dummy stage representing a time point that is after the last request is scheduled and before the current request that is to be scheduled.

Definition of decision. At each stage, a storage space needs to be determined. The decision s at stage k, defined as a feasible storage space $s_{r,k+a_r-1}$ ($r = p_i$), determines the storage space assigned to the request at stage k. Accordingly, the set of decisions S_k at stage k is the set of all storage spaces $S_{r,t}$ at time t, i.e., $S_k = S_{r,k+a_r-1}$.

Definition of state. For request p_i, a state (k, s) is defined as taking decision s at stage k. We define the state value $h(k, s)$ as the minimum yard length to satisfy the space requirement of request p_i for time points $k + a_r - 1, \ldots, t_r$ for state (k, s).

In the DP, the state values are computed backward from stage $|T_r|$ to stage 1. At the beginning of stage k, all the states at stages $k+1, \ldots, |T_r|$ are known. In order to obtain state values $h(k, s)$ for each state at stage k, we consider the state transition from states $(k+1, s)$ ($s \in S_{k+1}$) to state (k, s). So, the computation of the state value $h(k, s)$ is based on the following cases:

Case 1: State value $h(0, s)$ at the dummy stage 0. The decision set of dummy stage 0 is empty, and the state value at stage 0 is equal to the minimum used yard length after requests $p_1, \ldots, p_{i-1}$ are scheduled, that is $h(0, s) = L_{i-1}$. L_0 is simply set as zero for convenience.

Case 2: State value $h(k, s)$ of infeasible decisions at stage k ($k \geq 1$). In the preprocessing stage of the heuristic method, the storage space candidates set $S_{r,t}$ is generated without checking its feasibility. Hence, decisions in set S_k may violate the non-overlapping constraints or the non-decreasing constraints. To overcome this issue, a two-step preprocessing procedure is applied before they are used to compute $h(k, s)$.

- Step 1: We first consider the non-overlapping constraints. If the storage space corresponding to decision s overlaps with the scheduled yard space at that time point, $h(k, s)$ is then set to a very large positive integer M.
- Step 2: We next consider the non-decreasing constraints. First, the decisions at stage $|T_r|$ all satisfy the non-decreasing constraints. Next, for decision s at stage $|T_r|-1$, if there exists a decision s' at stage $|T_r|$ such that the storage space of s' covers the storage space of s, decision s is then a feasible decision at stage $|T_r|-1$. We define set $S_{|T_r|}(s)$, namely the cover set of decision s, as the set of all decisions at stage $|T_r|$ that cover decision s at stage $|T_r|-1$. It is easy to see that $S_{|T_r|}(s)$ is a subset of the decision set $S_{|T_r|}$. Generally, for a decision s at stage k ($k = 1, \ldots, |T_r|-1$), its cover set is $S_{k+1}(s)$ ($S_{k+1}(s) \subset S_{k+1}$). If $S_{k+1}(s)$ is empty, decision s is infeasible and the value $h(k, s)$ is set to a very large positive integer M.

In Case 3 and Case 4, computation of $h(k, s)$ only involves feasible decisions, which is also processed backward from stage $|T_r|$ to stage 1.

Case 3: State value $h(|T_r|, s)$ of feasible decisions at stage $|T_r|$. At stage $|T_r|$, the state value $h(|T_r|, s)$ of choosing decision s takes the greater value of $h(0, s)$ and $y^+_{r,t}$, that is $h(|T_r|, s) = \max\{L_{i-1}, y^+_{r,t}\}$. $h(0, s)$ is equal to L_{i-1} as defined in Case 1. $y^+_{r,t}$ is the yard length corresponding to the feasible storage space when decision $s = (x^-_{r,t}, y^-_{r,t}, x^+_{r,t}, y^+_{r,t})$.

Case 4: State value $h(k,s)$ of feasible decisions at stage k ($k < |T_r|$). As defined in Case 2, a feasible decision s at stage k has a cover set $S_{k+1}(s)$ at stage $k+1$. From decision s' ($s' \in S_{k+1}(s)$) at stage $k+1$ to decision s at stage k, the used yard length by s is equal to $h(k+1,s)$ due to the non-decreasing constraints. $h(k,s)$ takes the minimum value among all transitions from s' to s. Therefore, from all feasible decisions at stage $k+1$ to decision s at stage k, it follows that $h(k,s) = \min_{s' \in S_{k+1}(s)} h(k+1,s')$, and the selected decision s^*_{k+1} at stage $k+1$ equals $arg \min_{s' \in S_{k+1}(s)} h(k+1,s')$.

Based on the above four cases, noting that the feasible storage space s is defined as $(x^-_{r,t}, y^-_{r,t}, x^+_{r,t}, y^+_{r,t})$, the recurrent equation to determine the value of $h(k,s)$ is as follows:

$$h(k,s) = \begin{cases} L_{i-1}, & \text{if } k = 0, \\ M, & \text{if overlap or decreasing,} \\ \max\{L_{i-1}, y^+_{r,t}\}, & \text{if } k = |T_r|, \\ \min\limits_{s' \in S_{k+1}(s)} h(k+1,s'), & \text{if } k = 1,\ldots,|T_r|-1, \end{cases}$$

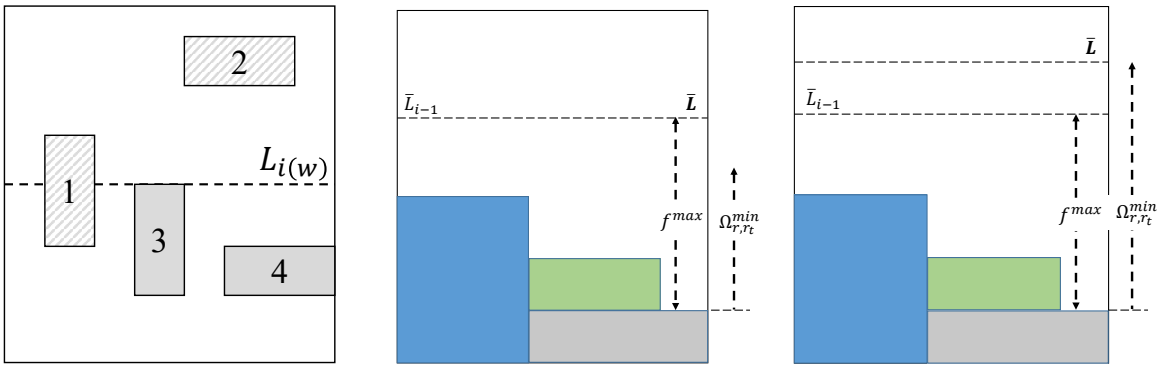
After obtaining all state values of $h(1,s)$ ($s \in S_1$) at stage 1, we set the minimum of them as the final objective value $L_i(w)$ for a given yard width w (Line 7 in Algorithm 7). The storage solution $\mathbf{s}_i(w) = \{s^*_1, \ldots, s^*_{|T_r|}\}$ is then derived through backtracking of states from stage 1 to stage $|T_r|$ (Line 8).

3.4.3 Speed-up techniques

In the aforementioned DP, we compute $|S_k|$ states at each stage k ($1 \le k \le |T_r|$). Recall that set S_k equals the storage space set $S_{r,t}$ ($t = a_r + k - 1$). Thus, to get the minimum used yard length $L_i(w)$ when W_i is fixed to w, DP has to compute the state values for $\sum_{k=1}^{|T_r|} |S_k| = \sum_{t=t_r}^{a_r} |S_{r,t}|$ times. Set $S_{r,t}$ contains a huge number of feasible storage spaces for the current request at time point t following the preprocessing procedure discussed in Section 3.3. However, it is easy to see that some feasible storage spaces $s_{r,t}$ are not good enough to generate a small yard length, so the DP procedure can be accelerated simply by ignoring such storage spaces. As an example, in Figure 3.5, storage spaces 1

and 2 should be ignored due to their inefficiency while storage spaces 3 and 4 should be retained.

We then compute an estimated value $\bar{L}$ for $L_i(w)$ to help determine whether a feasible storage space should be evaluated or ignored. If this estimated value $\bar{L} < L_i(w)$, the DP procedure would be a waste of time searching the reduced set without finding any feasible solution. If $\bar{L}$ is much larger than $L_i(w)$, the reduced set would still possibly be too large. Therefore, we aim to set $\bar{L}$ to a value larger than $L_i(w)$ by an appropriate amount. The following three steps are then taken.



(a) (b) (c)

Figure 3.5: Illustration of the speed-up process

(1) Check the available yard spaces according to the minimum length $\bar{L}_{i-1}$ for the preceding requests at time point t_r. We only consider available yard spaces at t_r because the minimum yard length at any other time point t ($a_r \leq t \leq t_r - 1$) is bounded by that at time point t_r due to the non-decreasing constraints. Yard length $\bar{L}$ could be reached by a particular feasible storage space in set S_{r,t_r}.

(2) If the special feasible storage space s (width=1, length=Ω^{min}_{r,t_r}) can be assigned to the maximum available yard space f^{max} ($\Omega^{min}_{r,t_r} \leq f^{max}$), yard length would not change, i.e., $\bar{L} = \bar{L}_{i-1}$ as illustrated by Figure 3.5b. Otherwise, $\bar{L}$ is set to a minimum possible value such that s can be placed into the available yard space (Figure 3.5c).

(3) Run the DP with the reduced storage spaces. If no feasible solution is found, increase $\bar{L}$ by 1, until a feasible solution is obtained.

3.5 Computational Experiments and Results

In this section, we first discuss test instance generation. Then, a simple lower bound (LB) to the objective value for each test instance is introduced. Next, we compare the performance of our solution approach with the simulated annealing based bottom-left (SA-BL) algorithm used in Fu et al. (2007) on both small, medium and large-scale instances. Finally, our solutions are compared with the optimal solutions obtained by a general optimization solver, IBM ILOG CPLEX, for small-scale instances. All experiments are implemented in Java and are conducted on a CentOS Linux workstation equipped with 16GB memory and Intel Pentium processor running at 3.5 GHz. The reported computational times of SA-DSAP include the processing time of generating set $S_{r,t}$. We use IBM ILOG CPLEX 12.6.1 for solving the ILP models.

3.5.1 Test data generation

Eight instances were developed and tested in Fu et al. (2007), but they are no longer available. Following the description in Fu et al. (2007) and taking new practical data (Port of Hong Kong in Figures 2019 Edition) into consideration, we generate 13 groups of new test instances, G1~G13, and each group consists of five test instances. These instances are all generated in a yard block with fixed width ($W = 6$), length ($L = 15$). The maximum stack height H is set to 5.

The planning horizon considered is set as seven days. Each day consists of 24 time points for one hour each. In total, there are 168 time points for the planning horizon. The number of requests ranges from 5 to 65 requests among Instance Groups G1 to G13. A request r is randomly generated as follows: (1) The length of storage time interval $[a_r, t_r]$ uniformly ranges from 4 to 20 time points. (2) The arrival time interval between two consecutive requests follows a Poisson distribution with mean equal to 4. We assume that the arrival time of the first request a_1 is 1, and then obtain the arrival time points a_r of all requests. (3) The departure time t_r of a request cannot exceed 168 (the maximum planning horizon). (4) The total number of containers $n_{r,t}$ by time t is generated according

to $\sum_{b\in B_r:a_r^b\le t}\sum_{t'\in[a_r^b,t]} n_r^{b,t'}$ where the size of set B_r is uniformly generated from 1 to 4. In order to simulate real cases, each element $b \in B_r$ represents a batch of containers arriving from the same origin, and containers from the same batch b are unloaded into the yard within a time interval $[a_r^b, t_r^b]$, where a_r^b and t_r^b are the time points when batch b starts and finishes unloading. We define $a_r = \min_{b\in B_r} a_r^b$ and $t_r = \max_{b\in B_r} t_r^b$. At each time point $t \in [a_r^b, t_r^b]$, there are $n_r^{b,t}$ arriving containers and $n_r^t = \sum_{b\in B_r} n_r^{b,t}$. The value of a_r^b, t_r^b and $n_r^{b,t}$ are generated as follows: (a) The length of interval $[a_r^b, t_r^b]$ is generated uniformly from 1 to the maximum feasible length $(t_r - a_r^b)$ for batch b. (b) The arrival time points a_r^b between two consecutive batches follow Poisson distribution with a mean equal to 4 time units. (c) At each time point t with an arrival, the number of arriving containers $n_r^{b,t}$ follows an exponential distribution with a mean equal to 5 units. Before generating data, we define a metric, *load intensity (LI)*, to show the complexity of a instance. LI is defined as $LI = \frac{\sum_{r\in R}\sum_{i\in r} d_i}{L\times W\times H\times T}$ where d_i is the storage duration of container i of request r. For each instance group, we generate five instances with a different total number of containers and different load intensity.

Table 3.1: Test instance groups

Scale	Group	N	Q (TEUs)	LI	T
small	G1	5	100 400	1%-4%	168
	G2	10	400-800	4%-10%	168
	G3	15	700-1300	8%-15%	168
	G4	20	900-1500	9%-19%	168
	G5	25	1100-1800	11%-20%	168
medium	G6	30	1600-2800	19%-33%	168
	G7	35	2000-2500	20%-35%	168
	G8	40	2200-2900	22%-36%	168
	G9	45	2700-3100	30%-34%	168
large	G10	50	2900-3300	30%-40%	168
	G11	55	3400-3900	38%-43%	168
	G12	60	3400-3900	36%-46%	168
	G13	65	3600-4300	39%-49%	168

As shown in Table 3.1, we have generated thirteen instance groups (G1~G13), classified into small, medium and large-scale groups based on the number of requests (N), the number of containers (Q) and load intensity (LI). The last Column "T" in Table 3.1 denotes the length of the planning horizon considered. A small-scale instance has less

than 1800 containers from up to 25 requests. A medium-scale instance has around 1600-3100 containers from 30-45 requests, while a large-scale instance has around 2900-4300 containers from 50-65 requests.

3.5.2 Lower bound

Since the problem objective is to find the minimal area of the rectangular cover s_C of all requests during the planning horizon T, a simple yet intuitive LB can be developed. The basic idea of the lower bound is to relax the non-decreasing constraints in the model, thus allowing containers in a request to be placed freely in the yard across time. With this relaxation, containers in different requests are to be placed closely adjacent to each other during each time unit, and the overall occupied storage space becomes extremely compact. Thus, this LB is calculated as the maximum sum of minimal storage space occupied by requests at each time among all time points during the planning horizon:

$$LB = \max_{0 \le t \le |T|} \min_{s \in S_{r,t}} \sum_{r \in R_t} \Omega_{r,t},$$

where R_t is the set of requests that require storage space at time point t as defined in Section 3.2.2 and $\Omega_{r,t}$ is the area of storage space required by request r at time point t as defined in Subsection 3.3.1.

3.5.3 Comparison between SA-DSAP and SA-BL

Experiments are conducted on small, medium and large instances using both SA-DSAP and simulated annealing based bottom-left (SA-BL) (Fu et al., 2007). The values of initial yard lengths for the two algorithms are set based on different strategies. Yard length in the BL algorithm is set to a positive integer that is large enough to accommodate the containers required to be stored in the yard. It is also found that the value of yard length has little influence on the computational time required for BL algorithm. For the DSAP algorithm, the value of yard length is dynamically set and updated for each request following strategies introduced in Section 3.4.3. The computational time includes the time to generate the set $S_{r,t}$ $(r \in R, t \in T_r)$.

Table 3.2: Comparison between SA-BL and SA-DSAP algorithms for small-scale instances

Instances				SA-BL			SA-DSAP				
Group	ID	LI	LB	z_1	Time (s)	Iterations	z_2	Time (s)	Iterations	Gap%	Imp%
G1	5_1	1.2%	10	12	0.5	7200	10	26.3	7203	0.0%	20.0%
	5_2	2.1%	12	18	0.3	7935	12	0.1	7214	0.0%	50.0%
	5_3	3.5%	16	24	0.3	16845	18	197.8	15314	12.5%	37.5%
	5_4	3.3%	18	30	0.3	16875	18	0.4	15341	0.0%	66.7%
	5_5	2.8%	24	36	0.3	28114	26	209.2	23428	8.3%	41.7%
	Avg.									4.2%	43.2%
G2	10_1	4.1%	24	36	0.4	34628	26	252.7	31480	8.3%	41.7%
	10_2	6.3%	30	42	0.4	34962	30	12.0	31784	0.0%	40.0%
	10_3	6.2%	36	48	0.4	43762	39	536.7	39784	8.3%	25.0%
	10_4	8.6%	42	60	0.4	57941	44	668.1	48284	4.8%	38.1%
	10_5	9.8%	48	72	0.5	62902	52	600.4	57184	8.3%	41.7%
	Avg.									6.0%	37.3%
G3	15_1	7.6%	30	48	0.4	71729	30	332.7	65208	0.0%	60.0%
	15_2	12.8%	36	42	0.5	72265	36	34.1	65695	0.0%	16.7%
	15_3	11.1%	42	60	0.5	78010	45	578.5	74295	7.1%	35.7%
	15_4	13.4%	48	60	0.5	91515	54	583.5	83195	12.5%	12.5%
	15_5	14.2%	54	78	0.5	101305	56	971.4	92095	3.7%	40.7%
	Avg.									4.7%	33.1%
G4	20_1	18.1%	30	42	0.5	105043	30	176.5	93788	0.0%	40.0%
	20_2	16.0%	36	48	0.5	110982	39	948.9	102288	8.3%	25.0%
	20_3	9.0%	42	60	0.6	119094	44	1131.4	110888	4.8%	38.1%
	20_4	12.6%	48	66	0.6	132113	50	723.3	119776	4.2%	33.3%
	20_5	15.7%	54	78	0.6	154507	56	929.6	128745	3.7%	40.7%
	Avg.									4.2%	35.4%
G5	25_1	11.5%	30	48	0.6	153266	33	930.8	136845	10.0%	50.0%
	25_2	11.6%	36	54	0.6	157801	39	795.4	145439	8.3%	41.7%
	25_3	17.7%	42	54	0.6	156217	42	2.4	145453	0.0%	28.6%
	25_4	18.2%	48	72	0.6	169738	52	1111.5	153888	8.3%	41.7%
	25_5	20.5%	54	84	0.8	163341	60	1330.2	162804	11.1%	44.4%
	Avg.									7.6%	41.3%

Table 3.2 compares SA-DSAP and SA-BL on the small-scale instance groups. We can see that SA-DSAP significantly outperforms SA-BL. SA-DSAP obtains high-quality solutions whose values are only about 5% higher than the lower bounds. The averages of "Gap%" (=(z_2-LB)/LB) are 4.2%, 6.0%, 4.7%, 4.2% and 7.6% for G1 to G5 respectively. SA-DSAP also outperforms SA-BL by 43.2%, 37.3%, 33.1%, 35.4% and 41.3% for G1 to G5 respectively as shown by "Imp%" (=(z_1-z_2)/LB). Column "Time (s)" and "Iterations" in Table 2 give the total CPU runtime and the number of actual iterations executed in the simulated annealing procedure of both algorithms. It can be observed that SA-BL performs considerably faster than SA-DSAP in terms of CPU runtime. This is reasonable because SA-BL adopts a simple greedy strategy when allocating space to requests of containers while SA-DSAP employs a more time-consuming dynamic programming method. The planning time horizon is usually set to at least a week, so the computational time of half an hour is acceptable. From Column "Iterations", we can see that the number of iterations that SA has explored in both algorithms are comparable. Besides, it can seen that some solutions by SA-DSAP equal to the LBs in Table 3.2. This implies that there is greater chance for instances with low load intensity LI to reach optimal solutions. The reasons are from two aspects. First, the lower bound is calculated based on the relaxation of the non-decreasing constraints, so a solution simply adopts the most compact way for placement of containers from different requests independently at each time point. Second, for small-scale instances in Table 3.2, when the "Load Intensity" is low, the yard space becomes sufficiently large and hence the requests can be easily placed in the most compact way when the storage space is allocated. As a result, for the small-scale instances with low "Load Intensity", objective values of the feasible solutions are more likely to be equal to the lower bounds.

Table 3.3 illustrates the comparison between SA-DSAP and SA-BL on the medium-scale instance groups (G6~G9). The average values of "Imp%" of the four groups are 29.4%, 31.7%, 41.1%, and 31.7% respectively. The average of "Gap%" is around 10% for medium-scale instances, which indicates that the results of SA-DSAP are close to the estimated lower bounds.

Table 3.3: Comparison between SA-BL and SA-DSAP algorithms for medium-scale instances

Instances				SA-BL			SA-DSAP				
Group	ID	LI	LB	z_1	Time (s)	Iterations	z_2	Time (s)	Iterations	Gap%	Imp%
G6	30_1	19.8%	36	48	0.7	183742	39	1558.0	171241	8.3%	25.0%
	30_2	19.2%	42	60	0.8	195738	48	1286.1	179741	14.3%	28.6%
	30_3	19.7%	48	66	0.8	192414	50	908.9	188641	4.2%	33.3%
	30_4	21.8%	54	66	0.8	240514	56	1450.8	197143	3.7%	18.5%
	30_5	33.2%	60	90	1.0	223006	65	1800.2	205915	8.3%	41.7%
	Avg.									7.8%	29.4%
G7	35_1	20.6%	40	60	0.8	245580	42	1701.3	214335	5.0%	45.0%
	35_2	21.7%	48	66	0.9	254640	52	1800.0	222807	8.3%	29.2%
	35_3	24.9%	54	72	1.0	265987	60	998.6	231763	11.1%	22.2%
	35_4	27.4%	60	84	1.1	275033	65	1800.1	240638	8.3%	31.7%
	35_5	33.1%	66	90	1.1	284946	70	1718.7	249730	6.1%	30.3%
	Avg.									7.8%	31.7%
G8	40_1	21.8%	36	54	1.0	296998	39	1215.1	258262	8.3%	41.7%
	40_2	30.9%	42	72	1.0	304466	44	1800.2	265334	4.8%	66.7%
	40_3	24.2%	48	72	0.9	314062	52	1680.1	273934	8.3%	41.7%
	40_4	35.2%	54	78	1.2	324753	65	1749.1	282847	20.4%	24.1%
	40_5	24.6%	60	84	1.0	334121	65	1469.7	291846	8.3%	31.7%
	Avg.									10.0%	41.1%
G9	45_1	30.0%	42	54	1.2	344358	48	1661.7	300546	14.3%	14.3%
	45_2	31.3%	48	72	1.2	353040	52	1800.1	308906	8.3%	41.7%
	45_3	33.8%	54	84	1.4	364649	60	1792.3	317730	11.1%	44.4%
	45_4	31.6%	60	90	1.3	373350	66	1543.4	326660	10.0%	40.0%
	45_5	29.9%	66	84	1.2	383047	72	1316.7	335707	9.1%	18.2%
	Avg.									10.6%	31.7%

Table 3.4 shows comparison between SA-DSAP and SA-BL on the large-scale groups (G10~G13). It can be observed that all the twenty solution values by SA-DSAP (Column 2) are within the preset yard area limit ($W \times L = 90$) while six of twenty solutions by SA-BL exceed this limit of 90. This suggests that SA-DSAP can cope with the situation well when high usage of yard space is expected. From Column "Gap%", it can be found that SA-DSAP obtains high-quality solutions whose values are only about 15% higher than the lower bounds. The averages of "Gap%" are 11.7%, 15.5%, 11.8%, and 14.0% for G10 to G13 respectively. Column "Imp%" shows that SA-DSAP outperforms SA-BL by 28.6%, 27.7%, 32.6% and 31.7% respectively. In general, SA-DSAP is capable of finding high-quality storage solutions when facing a large number of requests involving a large number of containers during the planning horizon.

Table 3.4: Comparison between SA-BL and SA-DSAP algorithms for large-scale instances

Instances				SA-BL			SA-DSAP				
Group	ID	*LI*	LB	z_1	Time (s)	Iterations	z_2	Time (s)	Iterations	Gap%	Imp%
G10	50_1	35.7%	48	66	1.4	393857	60	1800.0	342488	25.0%	12.5%
	50_2	29.9%	54	78	1.3	403212	56	1779.9	351389	3.7%	40.7%
	50_3	34.9%	60	90	1.4	413494	70	1540.7	360662	16.7%	33.3%
	50_4	38.3%	66	90	1.6	422555	72	1800.1	368029	9.1%	27.3%
	50_5	39.1%	72	96†	1.5	432009	75	1507.0	377348	4.2%	29.2%
	Avg.									11.7%	28.6%
G11	55_1	38.1%	48	72	1.6	439437	60	1800.1	383528	25.0%	25.0%
	55_2	42.5%	54	84	1.6	447513	65	1800.2	391569	20.4%	35.2%
	55_3	38.0%	60	84	1.5	459767	66	1800.1	400610	10.0%	30.0%
	55_4	40.4%	66	96†	1.7	468511	75	1553.9	409921	13.6%	31.8%
	55_5	42.5%	72	90	1.6	479327	78	1641.1	420088	8.3%	16.7%
	Avg.									15.5%	27.7%
G12	60_1	36.2%	48	66	1.5	492881	54	1383.2	428597	12.5%	25.0%
	60_2	42.7%	54	90	1.8	502272	66	1789.7	437717	22.2%	44.4%
	60_3	39.1%	60	90	1.6	511482	66	1800.2	446130	10.0%	40.0%
	60_4	40.5%	66	78	1.8	520919	70	1800.2	453700	6.1%	12.1%
	60_5	45.2%	72	108†	1.8	530219	78	1800.0	463132	8.3%	41.7%
	Avg.									11.8%	32.6%
G13	65_1	39.3%	55	72	1.7	539319	66	1800.2	470702	20.0%	10.9%
	65_2	49.0%	60	84	1.8	548775	70	1677.0	480172	16.7%	23.3%
	65_3	45.8%	66	102†	1.9	558017	72	1800.1	486218	9.1%	45.5%
	65_4	38.8%	72	102†	1.9	567417	84	1800.1	496458	16.7%	25.0%
	65_5	41.5%	78	126†	2.0	576687	84	1800.0	505415	7.7%	53.8%
	Avg.									14.0%	31.7%

Note: Objective value marked by † exceeds maximum yard area (90) and is infeasible.

3.5.4 Comparison with optimal solutions on small-scale instances

In this subsection, we conduct additional computational experiments to verify the effectiveness of our approach. SA-DSAP is benchmarked on small-scale instances (G1~G5) solvable to optimality based on a reduced ILP model (**M2**). The reduced ILP model (**M2**) is developed based on the original ILP model discussed in Section 3.2.2 and solutions obtained in Subsection 3.5.3; we incorporate two additional constraints:

$$\pi_p = 0, \text{ if } x_p y_p < \text{LB or } x_p y_p > \text{UB}, \forall p \in P, \tag{3.15}$$

$$\lambda_{r,t,s} = 0, \text{ if } x_s^+ y_s^+ > \text{UB}, \forall s \in S_{r,t}, \forall r \in R, \forall t \in T_r, \tag{3.16}$$

where LB is the estimated lower bound (Section 3.5.2) and UB is the feasible solution, also an upper bound, obtained by SA-DSAP. As we find out, the solution space in M2 has been greatly reduced after introduction of lower and upper bounds, thus making the model solvable by CPLEX, though it may take more than 24 hours to get the optimal solution for a single instance.

In Table 3.5, Column "CPLEX (**M2**)" shows the optimal objective values solved by

CPLEX. Column "SA-DSAP" shows the objective values by SA-DSAP within 30 minutes. Column "Diff%" (=$(z_2 - z_1)/z_1$) shows the gap between the solutions by SA-DSAP to the solutions by the general optimization solver CPLEX.

From Table 3.5, we observe that SA-DSAP finds the optimal solution in 23 out of 25 instances. For instance 20_5, the result of SA-DSAP is only slightly worse than the solved optimum by 1 unit (1.82%) in yard length, as shown in Column "Diff%". While the computing times for the already reduced ILP model (**M2**) are very long, requiring up to a few days using CPLEX, while SA-DSAP takes less than a few minutes. For instance 25_5 where CPLEX does not reach optimal, the objective value of SA-DSAP is equal to that of the feasible solution obtained by **M2** with much shorter computational time. The overall results of the experiments are encouraging, since high-quality solutions can be generated within a short period of time by SA-DSAP.

Table 3.5: Comparison between solutions by SA-DSAP and CPLEX

			CPLEX (**M2**)		SA-DSAP		
Group	ID	LB	z_1	Time (s)	z_2	Time (s)	Diff%
G1	5_1	10	**10**	0.76	**10**	0.25	0.00%
	5_2	12	**12**	0.43	**12**	0.23	0.00%
	5_3	16	**18**	1.50	**18**	0.93	0.00%
	5_4	18	**18**	1.03	**18**	0.91	0.00%
	5_5	24	**26**	12.92	**26**	0.24	0.00%
G2	10_1	24	**26**	85.08	**26**	3.13	0.00%
	10_2	30	**30**	11.89	**30**	4.20	0.00%
	10_3	36	**39**	61.98	**39**	0.08	0.00%
	10_4	42	**44**	438.47	**44**	1.27	0.00%
	10_5	48	**52**	763.48	**52**	8.52	0.00%
G3	15_1	30	**30**	38.08	**30**	0.71	0.00%
	15_2	36	**36**	44.23	**36**	13.65	0.00%
	15_3	42	**45**	149.78	**45**	2.44	0.00%
	15_4	48	**52**	719.91	**52**	93.68	0.00%
	15_5	54	**56**	599.09	**56**	32.39	0.00%
G4	20_1	30	**30**	62.16	**30**	54.14	0.00%
	20_2	36	**39**	594.15	**39**	20.58	0.00%
	20_3	42	**44**	1977.01	**44**	378.91	0.00%
	20_4	48	**50**	34691.52	**50**	76.22	0.00%
	20_5	54	**55**	2370.67	56	180.07	1.82%
G5	25_1	30	**33**	628.40	**33**	0.14	0.00%
	25_2	36	**39**	371.49	**39**	79.11	0.00%
	25_3	42	**42**	104.77	**42**	11.24	0.00%
	25_4	48	**52**	14631.70	**52**	34.26	0.00%
	25_5	54	60	86400.00	60	168.23	0.00%

3.5.5 Sensitivity analysis

In this subsection, we conduct a set of sensitivity analysis. We first test the effect of container quantity per request on storage solutions. We select three instances from the small, medium and large-scale data, 10_4, 30_2, 50_2. For each instance, we increase the number of containers by 5% to 50% and keep the storage time of the containers unchanged, resulting in 10 new instances as a group. We then apply the SA-DSAP algorithm on these three instance groups. The results are listed in Table 3.6 as follows.

Table 3.6: Results on "container number per request" factor

ID	per	z	Delta	ID	per	z	Delta	ID	per	z	Delta
10_4	0%	44	0	30_2	0%	48	0	50_2	0%	56	0
10_4_5	5%	51	7	30_2_5	5%	55	7	50_2_5	5%	70	14
10_4_10	10%	51	7	30_2_10	10%	55	7	50_2_10	10%	70	14
10_4_15	15%	54	10	30_2_15	15%	60	12	50_2_15	15%	72	16
10_4_20	20%	55	11	30_2_20	20%	60	12	50_2_20	20%	75	19
10_4_25	25%	60	16	30_2_25	25%	60	12	50_2_25	25%	78	22
10_4_30	30%	60	16	30_2_30	30%	65	17	50_2_30	30%	84	28
10_4_35	35%	63	19	30_2_35	35%	70	22	50_2_35	35%	84	28
10_4_40	40%	63	19	30_2_40	40%	70	22	50_2_40	40%	90	34
10_4_45	45%	66	22	30_2_45	45%	70	22	50_2_45	45%	90	34
10_4_50	50%	66	22	30_2_50	50%	70	22	50_2_50	50%	90	34

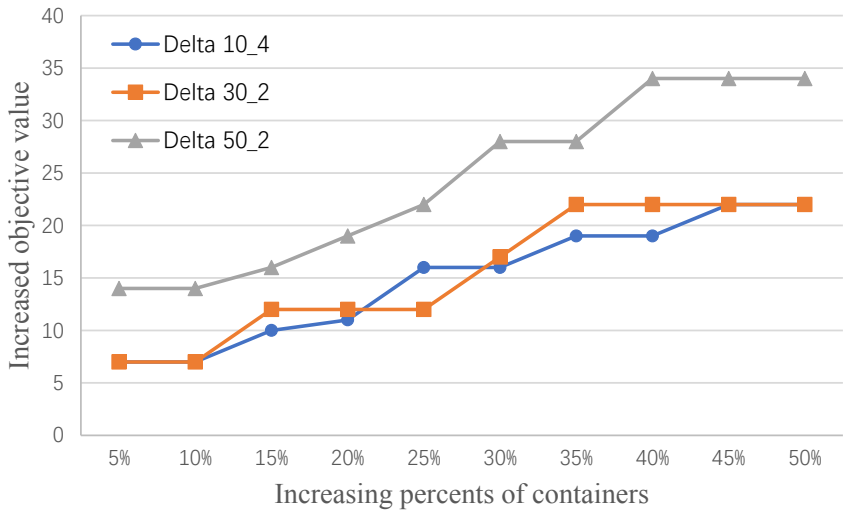


Figure 3.6: Illustration of results when container number increases

In Table 3.6, Column "ID", Column "per" and Column "z" represents the case name, increase in the number of containers and the objective function value, respectively. Column "Delta" represents the increase of objective value as compared with the original

objective value. From Table 2, we can see that in general the more containers a request needs to process, the more space required (see Figure 3.6), but for some instances such as 10_4_45 to 10_4_50, the overall space required does not change. The reason for the latter case, as we find out, is that there is much unused space in the original solution due to an uneven distribution of containers in the requests, and thus with the increase in the number of containers per request, the unused space is occupied first and the objective value does not change.

We then test the effect of yard length on the solutions. The current yard has a length of 20 units, and we reduce this yard length and keep other parameters unchanged. We select three instances, small, medium and large-scale from data, i.e., 10_4, 30_2, 50_2, and decrease the yard length by one unit each time from 20 to 10, resulting in a group of 10 new instances for each selected instance. We apply SA-DSAP on these new instance groups and present the results in Table 3.7.

Table 3.7: Results on "yard length" factor

ID	L	z	ID	L	z	ID	L	z
10_4	20	44	30_2	20	48	50_2	20	56
	19	44		19	48		19	56
	18	44		18	48		18	56
	17	44		17	48		17	56
	16	44		16	48		16	60
	15	44		15	48		15	60
	14	44		14	48		14	60
	13	44		13	48		13	60
	12	44		12	48		12	60
	11	44		11	48		11	60
	10	45		10	48		10	60

In Table 3.7, Column "ID", Column "L" and Column "z" represents the case name, the yard length and the objective function value, respectively. From Table 3.7, we observe that for small and medium-scale instances (10_4 and 30_2), the solution remains almost the same as the yard length decreases. For large-scale instances (50_2), the solution starts to change when the yard length decreases by a significant value (in this case decreased by 4 units to 16), as shown in Figure 3.7. The reason for this is that there is a large quantity of containers and there is a significant decrease in the yard length, the flexibility in placing containers in all the requests in the available yard space also decreases significantly.

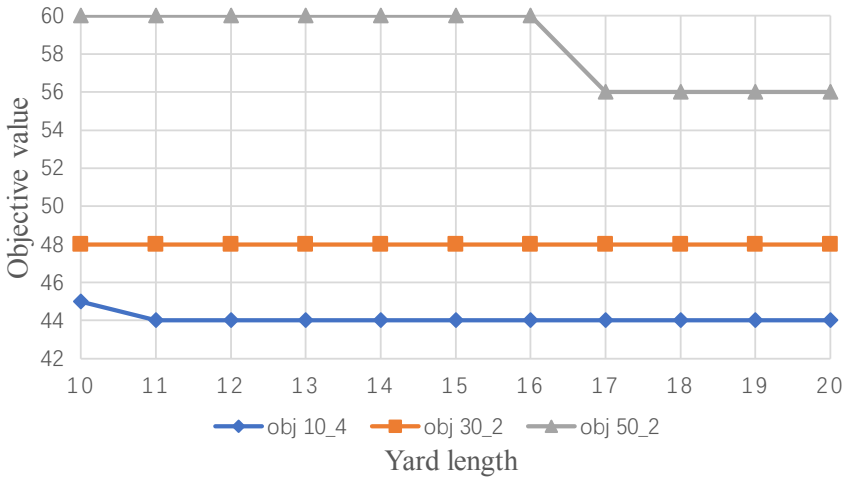


Figure 3.7: Illustration of results when yard length increases

Therefore, the objective values obtained by the proposed SA-DSAP algorithm become larger.

Finally, we test the effect of the difference d in length and width of the storage space on the solutions. We take a small-scale instance 10_4 and change the value of d from 0 to 15. SA-DSAP is then applied to this group of 16 instances. The results are presented in Table 3.8.

Table 3.8: Results on "storage space shape" factor

ID	d	z	d	z	d	z	d	z
10_4	0	60	4	45	8	44	12	44
	1	48	5	44	9	44	13	44
	2	48	6	44	10	44	14	44
	3	45	7	44	11	44	15	44

In Table 3.8, Column "ID", Column "d" and Column "z" represents the case name, the value of the difference d and the objective function value, respectively. As we observe from the table, for the square storage space ($d = 0$), our results are consistent with Fu et al. (2007). As the d-value increases, the objective value decreases initially and then remains unchanged. This implies that SA-DSAP algorithms are good at finding efficient storage solutions when there is more flexibility (larger d-values) in the shape of storage space. However, when the d-value exceeds a certain limit, the advantage of this flexibility appears less significant (Figure 3.8).

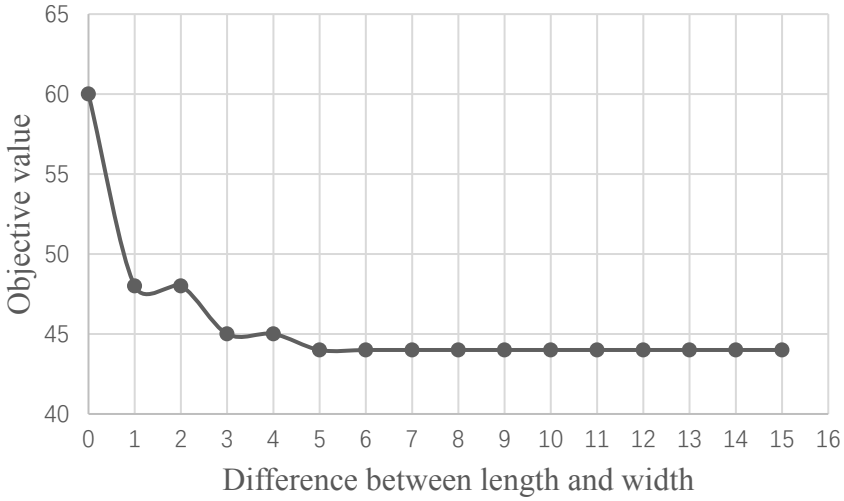


Figure 3.8: Illustration of the effect of d-value

3.5.6 Improvement on space utilization by SA-DSAP

To compare the unused space in solutions of SA-DSAP and SA-BL, we define $ratio_{unused}$ to indicate the proportion of unused sparse storage spaces in a solution $\mathbf{s}$, as follows:

$$ratio_{unused} = 1 - \frac{area_{\mathbf{s}}}{z_{\mathbf{s}} \times |T_{\mathbf{s}}|},$$

where $area_{\mathbf{s}}$ is the total storage space used in solution $\mathbf{s}$. $z_{\mathbf{s}}$ is the objective value of solution $\mathbf{s}$, i.e., the area of the rectangular cover in solution $\mathbf{s}$. $|T_{\mathbf{s}}|$ is the length of time for which containers are stored in the yard during the planning horizon. Computational experiments are conducted for both SA-DSAP and SA-BL on instance groups with size 15, 35 and 55 (small, medium and large-scale instances). The results are reported in the following Table 3.9.

As shown by the last column in Table 3.9, the unused space ratio $ratio_{unused}$ has been improved significantly by SA-DSAP compared with SA-BL. For small-scale instances, the improvement of $ratio_{unused}$ is between 2% and 13%. For medium-scale instances, the improvement of $ratio_{unused}$ is between 5% and 10%. For large-scale instances, the improvement of $ratio_{unused}$ is between 2% and 10%. We are then able to conclude that SA-DSAP can generate more efficient yard allocation solutions with less unused space compared with SA-BL.

Table 3.9: Results of unused space ratio on small, medium and large-scale data

		SA-BL			SA-DSAP			
ID	T_s	z	$area_{total}$	$ratio_{unused}(\%)$	z	$area_{total}$	$ratio_{unused}(\%)$	Imp(%)
15_1	67	48	870	73%	30	801	60%	13%
15_2	95	42	1456	64%	36	1400	59%	4%
15_3	98	60	1573	73%	45	1478	66%	7%
15_4	57	60	1527	55%	54	1449	53%	2%
15_5	69	78	1702	68%	56	1598	59%	10%
35_1	150	60	2672	70%	42	2506	60%	10%
35_2	156	66	2742	73%	52	2559	68%	5%
35_3	117	72	2878	66%	60	2734	61%	5%
35_4	133	84	3574	68%	65	3295	62%	6%
35_5	149	90	3823	71%	70	3654	65%	7%
55_1	165	72	4569	62%	60	4418	55%	6%
55_2	162	84	5195	62%	65	5057	52%	10%
55_3	157	84	4004	70%	66	3835	63%	7%
55_4	162	96	4794	69%	70	4621	59%	10%
55_5	165	90	4885	67%	78	4507	65%	2%

3.5.7 Non-empty yard

To test the effectiveness of the SA-DSAP algorithm in solving these cases with initially non-empty yards, we select three instance groups G4, G8, G12 (size=20, 40, 60) from small-scale to medium-scale and large-scale instances. For each instance, we randomly generate a non-empty initial yard status, that is, some requests are in the middle of being processed in the yard at the beginning of the planning time horizon, and therefore some portion of yard space has already been occupied. All these instances are solved by SA-DSAP. Results are shown in Table 3.10.

Table 3.10: Comparison results on initial yard status

ID	z^E	z^{NE}	Delta	ID	z^E	z^{NE}	Delta	ID	z^E	z^{NE}	Delta
20_1	30	30	0	40_1	39	40	1	60_1	54	54	0
20_2	39	39	0	40_2	44	45	1	60_2	66	66	0
20_3	44	46	2	40_3	52	52	0	60_3	66	66	0
20_4	50	54	4	40_4	65	65	0	60_4	70	70	0
20_5	56	60	4	40_5	65	65	0	60_5	78	80	2

In Table 3.10, z^E and z^{NE} represent the objective function value obtained from the initially empty instances and the initially non-empty instances. Delta is defined as $z^{NE} - z^E$. The results show that even if the yard is initially non-empty, our proposed algorithm can still obtain very competitive results in terms of yard usage that do not differ very

much from the initial empty instances.

3.5.8 Effects on load intensity

In this subsection, we conduct experiments to analyze how the SA-DSAP algorithm is affected by the load intensity (LI) of an instance. In order to find the threshold LI value that SA-DSAP cannot find a feasible solution, we gradually increase LI for a few instances, i.e., 61_1 65_5 from the largest data set G13 in Table 3.11 and generate five new data sets. We then apply the SA-DSAP algorithm on all instances.

In Table 3.11, Column "LI" and Column "z" represent the "Load Intensity" and the objective values. It can be seen that for the 65_1 instance, when LI increases to 55.07%, the SA-DSAP algorithm cannot obtain a feasible solution. Similarly, the threshold LI values for instances 65_2, 65_3, 65_4 and 65_5 are found to be 57.31%, 57.23%, 54.47%, and 57.99% respectively. Therefore, it is observed from the experiments that the SA-DSAP algorithm is unlikely to find a feasible solution for instances in the largest data set G13 when the load intensity is above 55%.

Table 3.11: Results of threshold LI values based on instance group G13 (size=65)

65_1		65_2		65_3		65_4		65_5	
LI	z	LI	z	LI	z	LI	z	LI	z
50.28%	84	55.84%	88	55.34%	90	46.66%	88	49.73%	90
52.12%	88	56.33%	90	**57.23%**	**92**	**54.47%**	**100**	**57.99%**	**100**
55.07%	**92**	56.82%	87	58.83%	90	62.28%	112	66.26%	110
57.78%	96	**57.31%**	**96**	60.78%	102	70.10%	128	74.52%	115
58.42%	96	57.80%	96	64.12%	102	77.91%	144	82.78%	132

3.6 Summary

In this chapter, we study the 3D yard allocation problem with time dimension (3DYAPT) at the operational level. Containers in each request are stacked in the yard following the consignment strategy. We aim to minimize the area of a rectangular yard space ever occupied by all requests within the planning horizon while satisfying non-decreasing constraints and the non-overlapping requirements. An integer linear programming model is

formulated for the problem, but it cannot be solved directly especially for medium-scale and large-scale instances due to NP-hardness of the 3DYAPT.

We propose a SA-DSAP algorithm that dynamically adjusts the shape of the stack of containers for each request at each time point. To balance the search time of the DSAP algorithm and the solution quality, a customized initial yard length strategy is proposed for each request based on both the information of the request in process and the instant layout information of the yard. Our work has its merit in practice because computational experiments show that the SA-DSAP algorithm outperforms the algorithm proposed by Fu et al. (2007) by 27.7% to 32.6% for large-scale groups based on average values, which means marked economic implications for ports. Besides, the SA-DSAP obtains optimal solutions within a short time compared with the general optimization solver CPLEX. Therefore, the proposed SA-DSAP algorithm is effective for solving the 3DYAPT.

CHAPTER 4

CONCLUSIONS AND FUTURE RESEARCH

4.1 Conclusions

This thesis studies two optimization problems for yard space allocation that consider the time dimension. The first problem aims to optimize total traveling distance and time deviation by taking into account the location and handling time of each subblock reserved for a transshipment vessel. The second problem aims to improve the space utilization of a single block by flexible allocation of the storage space for each request.

The first problem focuses on space allocation within a seaport yard. The primary space allocation unit is set to a size-fixed subblock. We study subblock allocation and handling time assignment for the transshipment vessels with multiple periods during a planning horizon. To avoid traffic congestion in the yard, we need to properly set the subblock location and the loading and unloading time of the subblocks in this study. We build a mixed-integer programming model which aims to minimize the total traveling distance of yard trucks. The MIP model is decomposed into two sub-models, and then we propose a three-stage heuristic algorithm, namely CETSA, based on the so-called critical element (CE) . The numerical results verify the effectiveness of the CETSA algorithm. Meanwhile, by comparing different space allocation methods, we can see that the integrated allocation method can obtain solutions with better objective values. In addition, we verify the influence of loading and unloading time decisions and the route length strategies on the total traveling distance of yard trucks.

In the second research problem of this thesis, we study the 3D yard allocation problem with time dimension (3DYAPT) at the operational level. Containers in each request are stacked in the yard following the consignment strategy. We aim to minimize the area of a rectangular yard space ever occupied by all requests within the planning horizon

while satisfying non-decreasing constraints and the non-overlapping requirements. An integer linear programming model is formulated for the problem, but it cannot be solved directly especially for medium-scale and large-scale instances due to NP-hardness of the 3DYAPT. We propose a SA-DSAP algorithm that dynamically adjusts the shape of the stack of containers for each request at each time point. To balance the search time of the DSAP algorithm and the solution quality, a customized initial yard length strategy is proposed for each request based on both the information of the request in process and the instant layout information of the yard. Our work has its merit in practice because computational experiments show that the SA-DSAP algorithm outperforms the algorithm proposed by Fu et al. (2007) by 27.7% to 32.6% for large-scale groups based on average values, which means marked economic implications for ports. Besides, the SA-DSAP obtains optimal solutions within a short time compared with the general optimization solver CPLEX. Therefore, the proposed SA-DSAP algorithm is effective for solving the 3DYAPT.

4.2 Limitations and Future Research

As for future studies, there are several directions in which this thesis are not considered and can be further improved. In the first problem, we assume that each truck serves unloading or loading containers of one vessel in one route. This assumption simplifies the traffic flows in the yard and accelerates the handling process of the vessel. However, the truck is empty in about half of the route, which inevitably reduces the utilization of the trucks. Lee et al. (2009) investigated the vehicle routing problem that a truck serves two vessels in one route from the operational level. It is possible to extend the first topic by considering the truck service order for one loading container and one loading container at a time. Such modification has the potential to reduce the number of trucks in the yard since each truck is utilized more efficiently. The energy consumption would also be reduced.

In the second problem, first, it is possible to extend the 3DYAPT by modifying con-

straints or objectives to address new practical aspects or requirements. For example, the rectangular storage shape assumption could be relaxed. Storage spaces with irregular boundaries, such as the flexible bay boundary shared by two neighboring segments (Zhou et al., 2020b), can be studied. Constraints that limit container tier difference can be considered. It is possible to incorporate this requirement by adding a post-processing procedure to the SA-DSAP algorithm that adjusts the height of container tiers based on a given strategy. Other requirements such that containers should be piled along the yard crane moving direction or in the length dimension, coupled with "container reshuffling", during the whole planning horizon can also be studied. The problem itself can also be extended to integrate yard space allocation with crane scheduling, thus taking reduction of energy consumption of crane operations (Iris and Lam, 2019) as one of the dual objectives into consideration. Second, there is currently still room for improvement in the solution approach. The proposed SA-DSAP algorithm is a heuristic method, in which requests are evaluated one by one according a pre-assigned priority sequence. Extensions can be made to consider allocation of two or more requests together in the dynamic programming in order to make the algorithm more efficient. For some simple or restricted cases of the 3DYAPT, it is also possible to develop approximation algorithms with certain performance guarantee.

Bibliography

Bazzazi, M., Safaei, N., Javadian, N., 2009. A genetic algorithm to solve the storage space allocation problem in a container terminal. Computers & Industrial Engineering 56, 44–52.

Buhrkal, K., Zuglian, S., Ropke, S., Larsen, J., Lusby, R., 2011. Models for the discrete berth allocation problem: A computational comparison. Transportation Research Part E: Logistics and Transportation Review 47, 461–473.

Cao, J., Shi, Q., Lee, D.H., 2008. A decision support method for truck scheduling and storage allocation problem at container. Tsinghua Science Technology 13, 211–216.

Cao, J.X., Lee, D.H., Chen, J.H., Shi, Q., 2010. The integrated yard truck and yard crane scheduling problem: Benders' decomposition-based methods. Transportation Research Part E: Logistics and Transportation Review 46, 344–353.

Carlo, H.J., Vis, I.F., Roodbergen, K.J., 2014. Storage yard operations in container terminals: Literature overview, trends, and research directions. European journal of operational research 235, 412–430.

Chang, D., Jiang, Z., Yan, W., He, J., 2011. Developing a dynamic rolling-horizon decision strategy for yard crane scheduling. Advanced Engineering Informatics 25, 485–494.

Chen, L., Langevin, A., Lu, Z., 2013. Integrated scheduling of crane handling and truck transportation in a maritime container terminal. European Journal of Operational Research 225, 142–152.

Chen, P., Fu, Z., Lim, A., 2002. The yard allocation problem, in: AAAI/IAAI, pp. 3–8.

Chen, X., He, S., Zhang, Y., Tong, L., Shang, P., Zhou, X., 2020. Yard crane and agv scheduling in automated container terminal: A multi-robot task allocation framework. Transportation Research Part C: Emerging Technologies 114, 241–271.

Chu, F., He, J., Zheng, F., Liu, M., 2019. Scheduling multiple yard cranes in two adjacent container blocks with position-dependent processing times. Computers Industrial Engineering 136, 355–365.

Fu, Z., Li, Y., Lim, A., Rodrigues, B., 2007. Port space allocation with a time dimension. Journal of the Operational Research Society 58, 797–807.

Galle, V., Barnhart, C., Jaillet, P., 2018. Yard crane scheduling for container storage, retrieval, and relocation. European Journal of Operational Research 271, 288–316.

Gharehgozli, A., Yu, Y., de Koster, R., Du, S., 2019. Sequencing storage and retrieval requests in a container block with multiple open locations. Transportation Research Part E: Logistics and Transportation Review 125, 261–284.

Gharehgozli, A.H., Laporte, G., Yu, Y., de Koster, R., 2015. Scheduling twin yard cranes in a container block. Transportation Science 49, 686–705.

Gharehgozli, A.H., Yu, Y., de Koster, R., Udding, J.T., 2014. An exact method for scheduling a yard crane. European Journal of Operational Research 235, 431–447.

Guo, X., Huang, S.Y., 2012. Dynamic space and time partitioning for yard crane workload management in container terminals. Transportation Science 46, 134–148.

Guo, X., Huang, S.Y., Hsu, W.J., Low, M.Y.H., 2011. Dynamic yard crane dispatching in container terminals with predicted vehicle arrival information. Advanced Engineering Informatics 25, 472–484.

Han, Y., Lee, L.H., Chew, E.P., Tan, K.C., 2008. A yard storage strategy for minimizing traffic congestion in a marine container transshipment hub. OR Spectrum 30, 697–720.

He, J., Huang, Y., Yan, W., Wang, S., 2015. Integrated internal truck, yard crane and quay crane scheduling in a container terminal considering energy consumption. Expert Systems with Applications 42, 2464–2487.

He, J., Tan, C., 2018. Modelling a resilient yard template under storage demand fluctuations in a container terminal. Engineering Optimization 51, 1547–1566.

He, J., Tan, C., Yan, W., Huang, W., Liu, M., Yu, H., 2020. Two-stage stochastic programming model for generating container yard template under uncertainty and traffic congestion. Advanced Engineering Informatics 43.

Hu, X., Guo, J., Zhang, Y., 2019. Optimal strategies for the yard truck scheduling in container terminal with the consideration of container clusters. Computers Industrial Engineering 137.

Hu, Z.H., Sheu, J.B., Luo, J.X., 2016. Sequencing twin automated stacking cranes in a block at automated container terminal. Transportation Research Part C: Emerging Technologies 69, 208–227.

Ibaraki, T., Nakamura, K., 2006. Packing problems with soft rectangles, in: International Workshop on Hybrid Metaheuristics, Springer. pp. 13–27.

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.

Iris, Ç., Lam, J.S.L., 2019. A review of energy efficiency in ports: Operational strategies, technologies and energy management systems. Renewable and Sustainable Energy Reviews 112, 170–182.

Iris, Ç., Pacino, D., Ropke, S., Larsen, A., 2015. Integrated berth allocation and quay crane assignment problem: Set partitioning models and computational results. Transportation Research Part E: Logistics and Transportation Review 81, 75–97.

Ji, J., Jeng, M., 1990. Bin-packing adjustable rectangles and applications to task scheduling on partitionable parallel computers, in: Proceedings of the Second IEEE Symposium on Parallel and Distributed Processing 1990, IEEE. pp. 312–315.

Jiang, X., Chew, E.P., Lee, L.H., Tan, K.C., 2012a. Flexible space-sharing strategy for storage yard management in a transshipment hub port. OR Spectrum 35, 417–439.

Jiang, X., Chew, E.P., Lee, L.H., Tan, K.C., 2014. Short-term space allocation for storage yard management in a transshipment hub port. OR Spectrum 36, 879–901.

Jiang, X., Lee, L.H., Chew, E.P., Han, Y., Tan, K.C., 2012b. A container yard storage strategy for improving land utilization and operation efficiency in a transshipment hub port. European Journal of Operational Research 221, 64–73.

Jiang, X.J., Jin, J.G., 2017. A branch-and-price method for integrated yard crane deployment and container allocation in transshipment yards. Transportation Research Part B: Methodological 98, 62–75.

Jin, J.G., Lee, D.H., Cao, J.X., 2016. Storage yard management in maritime container terminals. Transportation Science 50, 1300–1313.

Jin, J.G., Lee, D.H., Hu, H., 2015. Tactical berth and yard template design at container transshipment terminals: A column generation based approach. Transportation Research Part E: Logistics and Transportation Review 73, 168–184.

Kaveshgar, N., Huynh, N., 2015. Integrated quay crane and yard truck scheduling for unloading inbound containers. International Journal of Production Economics 159, 168–177.

Kress, D., Dornseifer, J., Jaehn, F., 2019. An exact solution approach for scheduling cooperative gantry cranes. European Journal of Operational Research 273, 82–101.

Lee, B.K., Kim, K.H., 2010. Comparison and evaluation of various cycle-time models for yard cranes in container terminals. International Journal of Production Economics 126, 350–360.

Lee, D.H., Cao, J.X., Shi, Q., Chen, J.H., 2009. A heuristic algorithm for yard truck scheduling and storage allocation problems. Transportation Research Part E: Logistics and Transportation Review 45, 810–820.

Li, M.K., Yip, T.L., 2013. Joint planning for yard storage space and home berths in container terminals. International Journal of Production Research 51, 3143–3155.

Li, W., Wu, Y., Petering, M.E.H., Goh, M., Souza, R.d., 2009. Discrete time model and algorithms for container yard crane scheduling. European Journal of Operational Research 198, 165–172.

Lim, A., Xu, Z., 2006. A critical-shaking neighborhood search for the yard allocation problem. European Journal of Operational Research 174, 1247–1259.

Liu, C., 2020. Iterative heuristic for simultaneous allocations of berths, quay cranes, and yards under practical situations. Transportation Research Part E: Logistics and Transportation Review 133, 101814.

Liu, C., Zhang, C., Zheng, L., 2016. A bi-objective model for robust yard allocation scheduling for outbound containers. Engineering Optimization 49, 113–135.

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.

Ma, H.L., Chung, S.H., Chan, H.K., Cui, L., 2017. An integrated model for berth and yard planning in container terminals with multi-continuous berth layout. Annals of Operations Research 273, 409–431.

Martello, S., Pisinger, D., Vigo, D., 2000. The three-dimensional bin packing problem. Operations Research 48, 256–267.

Martin Alcalde, E., Kim, K.H., Marchán, S.S., 2015. Optimal space for storage yard considering yard inventory forecasts and terminal performance. Transportation Research Part E: Logistics and Transportation Review 82, 101–128.

Murata, H., Fujiyoshi, K., Nakatake, S., Kajitani, Y., 1996. Vlsi module placement based on rectangle-packing by the sequence-pair. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 15, 1518–1524.

Ng, W., Mak, K., Li, M., 2010. Yard planning for vessel services with a cyclical calling pattern. Engineering Optimization 42, 1039–1054.

Ng, W.C., 2005. Crane scheduling in container yards with inter-crane interference. European Journal of Operational Research 164, 64–78.

Niu, B., Xie, T., Tan, L., Bi, Y., Wang, Z., 2016. Swarm intelligence algorithms for yard truck scheduling and storage allocation problems. Neurocomputing 188, 284–293.

Nossack, J., Briskorn, D., Pesch, E., 2018. Container dispatching and conflict-free yard crane routing in an automated container terminal. Transportation Science 52, 1059–1076.

Park, T., Choe, R., Hun Kim, Y., Ryel Ryu, K., 2011. Dynamic adjustment of container stacking policy in an automated container terminal. International Journal of Production Economics 133, 385–392.

Petering, M.E.H., Wu, Y., Li, W., Goh, M., de Souza, R., Murty, K.G., 2016. Real-time container storage location assignment at a seaport container transshipment terminal: dispersion levels, yard templates, and sensitivity analyses. Flexible Services and Manufacturing Journal 29, 369–402.

Robenek, T., Umang, N., Bierlaire, M., Ropke, S., 2014. A branch-and-price algorithm to solve the integrated berth allocation and yard assignment problem in bulk ports. European Journal of Operational Research 235, 399–411.

Sharif, O., Huynh, N., 2013. Storage space allocation at marine container terminals using ant-based control. Expert Systems with Applications 40, 2323–2330.

Sharif, O., Huynh, N., Chowdhury, M., Vidal, J.M., 2012. An agent-based solution framework for inter-block yard crane scheduling problems. International Journal of Transportation Science and Technology 1, 109–130.

Silva, E.F., Wauters, T., et al., 2019. Exact methods for three-dimensional cutting and packing: A comparative study concerning single container problems. Computers & Operations Research 109, 12–27.

Speer, U., Fischer, K., 2017. Scheduling of different automated yard crane systems at container terminals. Transportation Science 51, 305–324.

Tadumadze, G., Boysen, N., Emde, S., Weidinger, F., 2019. Integrated truck and workforce scheduling to accelerate the unloading of trucks. European Journal of Operational Research 278, 343–362.

Tan, C., He, J., Wang, Y., 2017. Storage yard management based on flexible yard template in container terminal. Advanced Engineering Informatics 34, 101–113.

Tan, C.M., He, J.L., 2016. Integrated yard space allocation and yard crane deployment problem in resource-limited container terminals. Scientific Programming 2016, 1–12.

Tang, L., Zhao, J., Liu, J., 2014. Modeling and solution of the joint quay crane and truck scheduling problem. European Journal of Operational Research 236, 978–990.

Tao, Y., Lee, C.Y., 2015. Joint planning of berth and yard allocation in transshipment terminals using multi-cluster stacking strategy. Transportation Research Part E: Logistics and Transportation Review 83, 34–50.

Vis, I.F.A., Carlo, H.J., 2010. Sequencing two cooperating automated stacking cranes in a container terminal. Transportation Science 44, 169–182.

Wang, K., Zhen, L., Wang, S., Laporte, G., 2018. Column generation for the integrated berth allocation, quay crane assignment, and yard assignment problem. Transportation Science 52, 812–834.

Wang, Y., Jiang, X., Lee, L.H., Chew, E.P., Tan, K.C., 2017. Tree based searching approaches for integrated vehicle dispatching and container allocation in a transshipment hub. Expert Systems with Applications 74, 139–150.

Wei, L., Oon, W.C., Zhu, W., Lim, A., 2011. A skyline heuristic for the 2d rectangular packing and strip packing problems. European Journal of Operational Research 215, 337–346.

Won, S.H., Zhang, X., Kim, K.H., 2011. Workload-based yard-planning system in container terminals. Journal of Intelligent Manufacturing 23, 2193–2206.

Wu, Y., Li, W., Goh, M., de Souza, R., 2010. Three-dimensional bin packing problem with variable bin height. European journal of operational research 202, 347–355.

Wu, Y., Li, W., Petering, M.E.H., Goh, M., Souza, R.d., 2015. Scheduling multiple yard cranes with crane interference and safety distance requirement. Transportation Science 49, 990–1005.

Yang, Y., Zhong, M., Dessouky, Y., Postolache, O., 2018. An integrated scheduling method for agv routing in automated container terminals. Computers Industrial Engineering 126, 482–493.

Yu, H., Ge, Y.E., Chen, J., Luo, L., Liu, D., Tan, C., 2017. Incorporating container location dispersion into evaluating gcr performance at a transhipment terminal. Maritime Policy Management 45, 770–786.

Yu, M., Qi, X., 2013. Storage space allocation models for inbound containers in an automatic container terminal. European Journal of Operational Research 226, 32–45.

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, 883–903.

Zhang, C.Q., Wan, Y.W., Liu, J.Y., Linn, R.J., 2002. Dynamic crane deployment in container storage yards. Transportation Research Part B-Methodological 36, 537–555.

Zhen, L., 2014. Container yard template planning under uncertain maritime market. Transportation Research Part E: Logistics and Transportation Review 69, 199–217.

Zhen, L., 2016. Modeling of yard congestion and optimization of yard template in container ports. Transportation Research Part B: Methodological 90, 83–104.

Zhen, L., Chew, E.P., Lee, L.H., 2011. An integrated model for berth template and yard template planning in transshipment hubs. Transportation Science 45, 483–504.

Zhen, L., Hu, H., Wang, W., Shi, X., Ma, C., 2018. Cranes scheduling in frame bridges based automated container terminals. Transportation Research Part C: Emerging Technologies 97, 369–384.

Zhen, L., Xu, Z., Wang, K., Ding, Y., 2016. Multi-period yard template planning in container terminals. Transportation Research Part B: Methodological 93, 700–719.

Zhou, C., Chew, E.P., Lee, L.H., 2018. Information-based allocation strategy for grid-based transshipment automated container terminal. Transportation Science 52, 707–721.

Zhou, C., Lee, B.K., Li, H., 2020a. Integrated optimization on yard crane scheduling and vehicle positioning at container yards. Transportation Research Part E: Logistics and Transportation Review 138.

Zhou, C., Wang, W., Li, H., 2020b. Container reshuffling considered space allocation problem in container terminals. Transportation Research Part E: Logistics and Transportation Review 136, 101869.

APPENDIX A

NP-HARD PROOF OF THE 3DYAPT IN CHAPTER 3

The NP-hardness of 3DYAPT is proved by reducing it from the well-known NP-hard Bin Packing Problem (BPP). The decision version of BPP can be described as follows: Let $\mathcal{R}$ be a set of n rectangles $R_1, R_2, \ldots, R_n$ with fixed orientations, whose lengths and widths are fixed integers, i.e., $R_i = (l_i, w_i)$, l_i, w_i are integers. A packing of $\mathcal{R}$ is the non-overlapping orthogonal placement of these rectangles into a rectangular sheet B with given length L and width W. The objective of this decision version BPP is to decide whether $\mathcal{R}$ can be packed into the given sheet B. Murata et al. (1996) proved this decision version BPP to be NP-hard.

We can reduce the BPP to the 3DYAPT.A special case of the 3DYAPT is designed as follows: All requests arrive at the same time and leave at the same departure time. Each request occupies the yard space for only one time unit. Accordingly, the total length of the planning horizon is equal to one time unit. Thus, the time dimension of the problem can be ignored in this special case. Also, the non-decreasing constraint can be reduced without the time dimension. We only consider the non-overlapping and the yard boundary constraints in this special case. For each request R_i at time t, it has a stack S_i^t which is corresponding to a projected area A_i^t after reducing the height dimension, i.e., $l_i^t \times w_i^t \geq A_i^t$, l_i^t, w_i^t is the length and width of stack S_i^t. In this problem, the shape of a stack can be modified, that is, for each stack S_i^t and its corresponding projected area A_i^t, the length and width of the stack is changeable, i.e., $\underline{l_i^t} \leq l_i^t \leq \overline{l_i^t}$, $\underline{w_i^t} \leq w_i^t \leq \overline{w_i^t}$. In this special case, the lower bound and upper bound are set equal for each l_i^t and w_i^t, i.e., $\underline{l_i^t} = l_i^t = \overline{l_i^t}$, $\underline{w_i^t} = w_i^t = \overline{w_i^t}$, which indicates that the shape of stacks is fixed for each request. Thus, the objective of this special case is to find the minimum rectangular space that can cover all the requests. Given a specified area a, we can obtain a rectangular

space S_a with width W_a and length L_a in the enclosed yard. The width and length are in range $\underline{W_a} \leq W_a \leq \overline{W_a}$ and $\underline{L_a} \leq L_a \leq \overline{L_a}$ such that $W_a \times L_a = a$. It follows that the special case of the problem reaches the minimum area ever used by all requests if and only if the BPP has a feasible packing, and therefore this particular case is NP-hard. Hence, the 3DYAPT in this paper which considers different arrival and departure times, a longer planning horizon and flexible shapes of container stacks is also NP-hard.