Optimization of Yard Operations in
Maritime Container Terminals
by
Setareh Borjian Boroujeni

Submitted to the Department of Civil and Environmental Engineering
and the Sloan School of Management
on May 15, 2015, in partial fulfillment of the
requirements for the degrees of
Master of Science in Transportation
and
Master of Science in Operations Research

Abstract

With the continuous growth in international container shipping, many container terminals in maritime ports face congestion, particularly during peak hours of service, and when there is limited space in the storage area. Thus, there has been increasing interest in improving operations efficiency in container terminals. An efficient terminal, in general, is one that discharges containers from the ships in a timely manner and delivers containers to customers with a reasonable wait time. Moreover, a key performance measure in the storage area is the number of moves performed by yard cranes.

Due to limited space in the storage area, containers are stacked on top of each other, forming a column of containers that can be accessed by yard cranes only from the top. Therefore, in order to retrieve a container that is covered by other containers, the blocking containers must be relocated to other slots. Because such relocation moves are costly for the port operators and result in service delays, one of the main challenges in the storage area is to plan the moves such that the number of relocations is minimized. This problem is referred to as the Container Relocation Problem (CRP).

The CRP in its most simplified setting is concerned with finding a sequence of moves that retrieves all containers in a pre-defined order with a minimum number of relocations, assuming that no new containers are stacked during the retrieval process. Also, it is often assumed that the non-blocking containers cannot be relocated (i.e., repositioning moves are not allowed), an assumption that can result in a sub-optimal solution.

Other variants of the container relocation problem include the dynamic CRP and the CRP with incomplete information. The former involves minimizing the number of relocations when containers are continuously stacked in and retrieved from the storage area, and the latter refers to the case that the departure times of containers
are not fully known in advance. For example, a probabilistic distribution of container
departure orders, or approximate departure times (in the form of time windows) might
be known.

Another important efficiency metric, in addition to the number of relocations, is
customer wait times during the retrieval process. In particular, when repositioning
moves are allowed in the system, there is a trade-off between the total number of re-
locations (including repositionings) and wait times, because such repositioning moves
make the retrieval process faster for trucks arriving in the future. Also, it might be
desired to prioritize some customers so that those prioritized experience shorter wait
times. For example, in terminals with appointment systems, shorter waiting time
guarantees can be given to customers who book in advance a time slot for picking
up their containers. In this thesis, we propose optimization models that capture
service-based and cost-based objectives and study different service policies.

In the first part of this thesis, we study the CRP with complete information
using an optimization model and heuristic approach. In particular, we formulate
CRP (with no restrictive assumptions on repositioning moves) as an Integer Program
that minimizes the weighted sum of the number of relocations and the total wait
time of customers. Our integer program provides the optimal sequence of moves
for retrieving containers subject to various service policies. For example, it can be
used by port operators to minimize customer wait times, or to give different waiting
time guarantees to different customers to reflect relative priorities. Moreover,
by assigning different weight factors to the two objectives, one can use our model to plan
repositioning moves. We also extend our model to the dynamic CRP and illustrate
how the flexibility in the stacking process can be exploited to optimize jointly the
sequence of moves and the stacking position of containers. Additionally, we propose a
class of flexible retrieval policies. We demonstrate that flexible policies can result in
fewer relocations and shorter wait times, thereby benefiting both the port operators
and customers.

In the second part of this thesis, we study the CRP with incomplete information
in a 2-stage setting where the departure times of a subset of containers are initially
known and the departure times of other containers are revealed at once at a later
time. The contributions are twofold. First, we propose an approximate stochas-
tic optimization algorithm, called \textit{ASA*}, which is a branch-and-bound framework
combined with a sampling technique, and to the best of our knowledge is the first
optimization algorithm proposed for this problem. We provide theoretical bounds on
the approximation errors and present numerical results showing the computational
tractability and efficiency of our algorithm. Second, we use the \textit{ASA*} algorithm and
a myopic heuristic to study the value of information, that is, the effect of the number
of containers initially known on the number of relocations.

In the last part of this thesis, we introduce a simulator that is capable of integrated
simulation of port operations, including the retrieval process, the stacking process,
and other aspects such as allocating cranes to containers and allocating trucks to
cranes. Our simulator captures the practical details of operations that cannot be
modelled in an optimization framework and is capable of simulating long periods
(e.g. a week) of realistic-scale operations.
Thesis Supervisor: Cynthia Barnhart
Title: Chancellor
Ford Professor of Civil and Environmental Engineering

Thesis Supervisor: Patrick Jaillet
Title: Dugald C. Jackson Professor of Electrical Engineering and Computer Science
Acknowledgments

First and foremost, I would like to express my deepest gratitude to my advisors, Prof. Cynthia Barnhart and Prof. Patrick Jaillet. I am truly thankful to Cindy for her constant support and encouragement and for her invaluable guidance. She has been an exceptional role model to me as an academic adviser and as a friend. Her caring and kindness have made my journey at MIT an amazing one. I am also sincerely grateful to Patrick for his kind support and encouragement and for insightful discussions. It has been a great honor for me to work under his supervision.

I am also very grateful to Maria Marangiello, Laura Rose, and Andrew Carvalho for their administrative assistance.

I am indebted to all my teachers. My special appreciation goes to Prof. George Kocur for all I learned from him in the two most amazing courses I took with him and for his support and advices. I am also thankful to Mr. John Attanucci for his kind support from my day one at MIT as an MST student and for helping me to learn about the research opportunities at the MIT transportation program.

I would like to thank Vahideh Manshadi for her mentorship and useful discussions during my three years at MIT and for her friendship. I am also thankful to my good friend and research collaborator, Virgile, for his great company. Over the past three years, I learnt a great deal by working on assignments and course projects with my friends. I had a wonderful experience working with Krishna on Informs competition, a teamwork that led to winning the second place. I am thankful to Esther for her friendship. I am also grateful to Mina for her kindness and friendship that made my journey at MIT more enjoyable.

Most importantly, I want to thank my dearest family. I owe my deepest gratitude to my mother, Fatemeh, and my father, Asghar, for their selfless love and sacrifice, and my sister, Sima, for always being there. Finally, I am truly grateful to my husband, Amir, for his love, caring, and encouragement. I cannot thank him enough for his constant support, for believing in me, and for being my best friend.
Contents

1 Introduction .......................................................... 15
   1.1 Some Definitions ................................................. 18
   1.2 Container Relocation Problem (CRP) ................................ 20
      1.2.1 Assumptions of the Basic Model of the CRP .............. 21
   1.3 Literature Review ............................................... 23
   1.4 Contributions .................................................. 25
   1.5 Thesis Outline ................................................ 27

2 CRP with Complete Information ..................................... 29
   2.1 Time-Based and Order-Based Class of Models .................. 30
      2.1.1 Handling the Idle times and Repositioning Moves .......... 32
      2.1.2 Joint Optimization of Wait time and Relocations .......... 33
      2.1.3 Dynamic Container Relocation Problem (DCRP) ............... 35
      2.1.4 Modelling Operations Flexibility .......................... 36
   2.2 Mathematical Model and Formulation for the Time-Based CRP .... 36
   2.3 Mathematical Model and Formulation for the DCRP ............... 44
   2.4 Out-of-Order Retrieval Policy .................................. 48
      2.4.1 Computational Experiments for a Flexible Retrieval Policy .......................................... 54
   2.5 The A* Algorithm ............................................... 58
      2.5.1 Lower Bound ................................................ 60
      2.5.2 Upper Bound ................................................ 63
   2.6 Heuristic Approach ............................................. 66
2.6.1 Tree Heuristic ................................................. 66
2.6.2 Numerical Experiments ........................................ 68
2.7 Concluding Remarks ............................................. 70

3 CRP with Incomplete Information 71

3.1 Problem Setting ............................................... 71
3.2 The ASA* Algorithm ............................................. 73
3.3 Heuristic Approach ............................................. 80
  3.3.1 Benchmark of the Myopic Heuristics ...................... 83
3.4 Value of Information ........................................... 86
3.5 Concluding Remarks ........................................... 89

4 Simulation Study of Port Operations 91

4.1 Simulator ....................................................... 91
4.2 User Interface .................................................. 97
4.3 Experiments .................................................... 98
4.4 Concluding Remarks ........................................... 102

5 Conclusions and Directions for Future Research 103
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Schematic diagram of a container terminal</td>
<td>17</td>
</tr>
<tr>
<td>1-2</td>
<td>Illustration of a block of container</td>
<td>17</td>
</tr>
<tr>
<td>2-1</td>
<td>Initial configuration of the bay for Example 2.1.1</td>
<td>32</td>
</tr>
<tr>
<td>2-2</td>
<td>Initial configuration of the bay for Example 2.1.2</td>
<td>34</td>
</tr>
<tr>
<td>2-3</td>
<td>Effect of traffic burst on average customer wait time</td>
<td>41</td>
</tr>
<tr>
<td>2-4</td>
<td>Initial configuration of the bay for Example 2.4.2</td>
<td>51</td>
</tr>
<tr>
<td>2-5</td>
<td>An example of an out-of-order retrieval policy</td>
<td>51</td>
</tr>
<tr>
<td>2-6</td>
<td>Out-of-order retrieval policy: relocations and customer wait times</td>
<td>55</td>
</tr>
<tr>
<td>2-7</td>
<td>Out-of-order retrieval policy: customer equity</td>
<td>57</td>
</tr>
<tr>
<td>2-8</td>
<td>The $A^*$ algorithm</td>
<td>61</td>
</tr>
<tr>
<td>2-9</td>
<td>Initial configuration of the bay for Example 2.5.1</td>
<td>65</td>
</tr>
<tr>
<td>2-10</td>
<td>An example of $A^*$ algorithm</td>
<td>66</td>
</tr>
<tr>
<td>2-11</td>
<td>The tree heuristic</td>
<td>67</td>
</tr>
<tr>
<td>3-1</td>
<td>Illustration of the error due to pruning in the $ASA^*$ algorithm</td>
<td>77</td>
</tr>
<tr>
<td>3-2</td>
<td>Tractability of the $ASA^*$ algorithm</td>
<td>81</td>
</tr>
<tr>
<td>3-3</td>
<td>Performance of the myopic heuristics when information level is $[0.25N]$</td>
<td>84</td>
</tr>
<tr>
<td>3-4</td>
<td>Performance of the myopic heuristics when information level is $[0.50N]$</td>
<td>85</td>
</tr>
<tr>
<td>3-5</td>
<td>Performance of the myopic heuristics when information level is $[0.75N]$</td>
<td>86</td>
</tr>
<tr>
<td>3-6</td>
<td>Experiment results for medium-sized instances</td>
<td>88</td>
</tr>
<tr>
<td>4-1</td>
<td>User interface for the port operations simulator</td>
<td>98</td>
</tr>
<tr>
<td>4-2</td>
<td>Percentage difference in number of relocations</td>
<td>100</td>
</tr>
</tbody>
</table>
## List of Tables

1.1 World’s busiest ports in 2010-2013 ........................................... 16

2.1 Notations ................................................................. 31

2.2 Solutions of Example 2.1.1 ........................................... 33

2.3 Solutions of Example 2.1.2 ........................................... 35

2.4 Solution for an example of the DCRP ............................. 48

2.5 Number of relocations and delay for Example 2.4.2 .......... 53

2.6 Benchmarks of heuristics ............................................. 69

2.7 Effect of parameter L on the performance of heuristic TH-L ... 69

3.1 Approximation errors of the $ASA^*$ algorithm ................... 79

4.1 Heuristics used in the simulator ................................... 96

4.2 Value of simulation parameters ................................... 99

4.3 Comparison of efficiency metrics for different levels of information 101

4.4 Comparison of average customer wait time and average stacking delay for different levels of traffic 102
Chapter 1

Introduction

As critical international logistics nodes, maritime container terminals play an important role in regional and national economies by facilitating the flow of goods between different locations all around the world. In general, three types of container transportation are handled in a maritime container terminal: (i) containers arrive on ships at the terminal and are then distributed in the nearby cities by trucks or trains (import); (ii) containers are transported by ships to a final destination (export); (iii) containers are transported by ships to an intermediate destination, then to yet another destination (transshipment)\(^1\). In each case, the collective facilities and terminals that conduct the container transportation is often referred to as a *maritime port*. Throughout this thesis, we use the terms terminal and port interchangeably.

While in general all three types of containers (import, export, and transshipment) are handled in a maritime terminal, some ports primarily facilitate the transshipment of containers. Table 1.1 shows the list of the ten busiest ports (from 2010 to 2013) and their container traffic in thousand TEUs\(^2\). Ports of Singapore, Hong Kong, and Shanghai are among the five busiest ports and are also the three busiest transshipment ports in the world.

An overview of typical operations in container terminals is as follows: On the

---

\(^1\)One possible reason for transshipment is to combine several (small) shipments into a large shipment or to divide a large shipment and transport the small shipments to different destinations.

\(^2\)The cargo capacity of container terminals and ships is often described in twenty-foot equivalent units (TEUs) representing the volume of a 20-foot-long container.
<table>
<thead>
<tr>
<th>Rank</th>
<th>Port</th>
<th>Territory</th>
<th>Container Traffic (thousand TEUs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>2013</td>
</tr>
<tr>
<td>1</td>
<td>Shanghai</td>
<td>China</td>
<td>33617</td>
</tr>
<tr>
<td>2</td>
<td>Singapore</td>
<td>Singapore</td>
<td>32240</td>
</tr>
<tr>
<td>3</td>
<td>Shenzhen</td>
<td>China</td>
<td>23280</td>
</tr>
<tr>
<td>4</td>
<td>Hong Kong</td>
<td>Hong Kong</td>
<td>22352</td>
</tr>
<tr>
<td>5</td>
<td>Busan</td>
<td>South Korea</td>
<td>17690</td>
</tr>
<tr>
<td>6</td>
<td>Ningbo-Zhoushan</td>
<td>China</td>
<td>17351</td>
</tr>
<tr>
<td>7</td>
<td>Qingdao</td>
<td>China</td>
<td>15520</td>
</tr>
<tr>
<td>8</td>
<td>Guangzhou</td>
<td>China</td>
<td>15309</td>
</tr>
<tr>
<td>9</td>
<td>Dubai</td>
<td>United Arab Emirates</td>
<td>13641</td>
</tr>
<tr>
<td>10</td>
<td>Tianjin</td>
<td>China</td>
<td>13010</td>
</tr>
</tbody>
</table>

Table 1.1: World’s busiest ports in 2010-2013 [1]

Seaside, containers are unloaded from (or loaded to) ships using quay cranes, and on the land side, they are stacked in (or retrieved from) a storage yard using smaller mobile cranes called RTGs (rubber tyred gantry cranes) \(^3\). Internal trucks transfer containers from one point to another inside the terminal, whereas external trucks transfer the containers from the storage yard to outside the terminal, or vice versa. A schematic diagram of different parts of a container terminal and the equipment is demonstrated in Figure 1-1.

In a typical container terminal, the storage yard is the area in which containers are stacked temporarily before they are loaded onto ships or external trucks. The storage yard is usually divided into several blocks. There are several rows (also referred to as bays) in each block, and several columns in each row, formed by stacking containers in tiers. The slot occupied by a container in a bay, can be specified by the tier and the column in which the container is stacked (see Figure 1-2).

Upon arrival for a retrieval, the external truck driver checks in at the terminal entrance gate with the information of the container to be picked up (which we refer to as the target container), and then drives to the block where the container is stacked.

\(^3\)Other equipment is also used occasionally for moving containers. For example, a reach stacker is a vehicle used for handling containers in areas where RTGs cannot operate. Unlike RTGs that cannot move between blocks, smaller vehicles like reach stackers are able to transport a container short distances within one block or between blocks.
The container is retrieved by an RTG and loaded onto the truck. The RTGs can access a container only from the top. So if the target container is not at the top of its column and is covered, the blocking containers must be relocated. In the next section, we present definitions of some terms that will frequently be used throughout this thesis.
1.1 Some Definitions

**Storage yard:** An area in the terminal where the containers are temporarily stacked to be later distributed in the city or to be loaded on ships for onward transportation.

**Block:** Groups of containers that are stacked together in the storage yard. Usually import, export, and transshipment containers are stacked in different blocks.

**Row (or bay):** Each block of containers in the storage yard consists of several rows or bays (20 to 60 rows depending on the block size).

**Column:** Each row or bay of containers consists of 7 to 10 columns.

**Tier:** In each column, containers are stacked on top of each other typically containing up to 4 to 6 containers.

**Slot:** Each container in the storage yard occupies one slot that can be described by its block, row, column, and tier. In this thesis we assume all containers (and all slots) are the same size.

**Quay gantry crane:** A type of crane used in the sea-side of container ports to load containers on and off ships.

**Rubber tyred gantry (RTG):** A mobile crane used in the storage yard of container terminals. An RTG straddles multiple columns of containers and moves along the rows. RTGs cannot move from one block to another; typically, one or two RTG are assigned to a block of containers depending on its size.

**Internal truck:** Internal trucks are used in the terminal to transfer containers from the sea-side to the land-side, and vice versa.

**External truck:** External trucks belong to customers; they deliver export containers to the terminal or pick up import containers to distribute them into the city.

**Target container:** A container that needs to be retrieved (and delivered to an external truck) at a given time.

**Blocking container:** A container that is on top of the target container, i.e., is in the same column as the target container but in a higher tier.

**Relocation:** The process of relocating a blocking container to a different slot (by an RTG, often) in order to access the target container.
Repositioning: The process of relocating a non-blocking container to a different slot with the aim of avoiding future relocations (see example 2.1.1 in Chapter 2).

Scheduled and actual departure times: The scheduled departure time is the earliest time that a container can be retrieved; it is the time that the external truck picking up that container arrives at the block. Similarly, the actual departure time is the time that the container is retrieved.

Scheduled and actual arrival times: The scheduled arrival time is the earliest time that a container is available to be stacked; it is the time that a container is transferred to the storage yard (after being unloaded from the ship) and is ready to be stacked. Similarly, the actual arrival time is the time that the container is stacked.

Service times: For the sake of brevity, we refer to container “scheduled departure times and arrival times” as “service times”. Depending on the setting of the problem (with complete or incomplete information), service times may be deterministic or uncertain.

Retrieval (stacking) time window: The maximum delay allowed in retrieving (stacking) a container from its scheduled departure (arrival) time.

Wait time (also referred to as retrieval delay): The amount of time that an external truck waits in the storage yard before its container is retrieved and loaded on the truck. Wait time can be measured as the difference between scheduled and actual departure time of a container.

Idle time: Any time that no truck is awaiting a retrieval or stacking and therefore the RTG is not serving an external or internal truck.

First-come-first-served (FCFS) policy: A retrieval policy that requires the containers to be retrieved in the order of truck arrivals.

Different aspects of port operations from the scheduling of trucks and cranes to optimizing the yard layout have been studied in the literature. In this thesis, we focus on the operations in the storage yard and more specifically, the Container Relocation Problem (CRP). In the next section we introduce the CRP, explain different variants of the problem, and state the assumptions for the basic models of the CRP.
In Section 1.3, we provide a brief review of the papers related to different aspects of port operations optimization.

1.2 Container Relocation Problem (CRP)

The container relocation problem arises as a result of stacking containers on top of each other; it often happens that the target container is covered by other containers. In this case, blocking containers need to be relocated, and such relocations are considered non-productive and costly moves. These relocations are costly for port operators, because they are not charged to the customer. Moreover, relocations result in retrieval delay and therefore, lower the quality of customer service.

In a container terminal, the number of relocations compared to the total number of moves (including retrievals and relocations) is an important measure of the efficiency of yard management. Thus, decreasing this ratio is of great practical interest, and it is a key objective in yard operations. Moreover, port operators desire to serve each external truck within a reasonable amount of time and keep the average wait time below a certain level. The question that yard managers face is as follows: given the estimated arrival and departure times of containers (in a certain period of time), what is the stacking, relocating, and retrieval plan that will minimize the number of relocations while retrieving each container within a reasonable time?

The CRP in its most simplified setting relies on a set of assumptions as explained in Section 1.2.1. Relaxing any of the assumptions results in variants of the problem that better represent real operations but are harder to model and solve. These variants of the CRP include:

The Static and Dynamic Container Relocation Problem (DCRP): In the static setting, it is assumed that containers have already been stacked in a block and the objective is to retrieve the containers in a pre-defined order. During the retrieval process, no new containers are stacked in the block. In the dynamic setting, it is assumed that containers are continuously stacked in and retrieved from the block.
The dynamic setting is a better representation of real operations but the resulting problem is more complicated. In section 2.3, we present an extended model that is applicable to the DCRP.

**The CRP with Complete and Incomplete Information:** In the CRP with complete information, it is assumed that the departure order and arrival order of containers are known in advance. In the CRP with incomplete information, port operators have no information or partial information about these orders in advance (for example, a probabilistic distribution of the departure and arrival orders may be known). In Chapter 3, we explain the setting of the CRP with incomplete information and present an approximate stochastic algorithm to solve the problem.

**The Restricted and Non-Restricted CRP:** The restricted CRP is the setting where only the blocking containers in the same column as the target container can be relocated, i.e., repositioning moves are not allowed. In the non-restricted setting, any container, whether blocking the target container or not, can be moved. The mathematical model and formulation presented in Chapter 2 are applicable to the restricted and non-restricted CRP instances. The algorithms in Section 2.5 and 3.2 and the heuristics presented for the CRP with complete and incomplete information are based on the restricted setting of the CRP.

### 1.2.1 Assumptions of the Basic Model of the CRP

The assumptions for the most simplified setting of the CRP are as follows:

1. Containers have already been stacked in the block and no new containers are stacked during the retrieval process (referred to as the static CRP).
2. Containers are retrieved in a pre-defined order.
3. The departure order of containers is known in advance (referred to as the CRP with complete information).
4. Only the blocking containers on top of the target container can be relocated. In other words, no repositioning move is performed (referred to as the restricted CRP).
5. There is one RTG per block of containers. So at any given time, only one move (retrieval or relocation) can be performed.

6. Delay and wait times are not accounted for. The objective is only to minimize the number of relocation moves.

Assumption 6 is the critical assumption behind the basic model of the CRP and is the common assumption of all existing models and heuristics in the literature. A model based on this assumption essentially abstracts away the notion of time and does not take into account the time of the day that each move is performed or the required amount of time for completing each move. We refer to this class of models as order-based. A common aspect of all order-based models and heuristics is that they focus only on minimizing container relocations, and do not capture truck wait times or crane idle times when no truck is awaiting a retrieval.

In an order-based model, the information about the departure times is translated into a departure order, which becomes an input to the model. Ignoring the departure time (and arrival time) of the containers and not using them in the model, results in some limitations. Namely the objective of providing a particular level of service or of measuring wait times for external trucks cannot be captured. Service-based goals, therefore, are not considered and instead, the focus is only on cost-minimizing objectives for the port operators. Moreover in many terminals, repositioning moves are performed during the off-peak hours in order to make the retrieval process faster for trucks arriving in the future. Such moves cannot be modelled by an order-based model either. Lastly, specifying a retrieval and stacking order for the DCRP is not straightforward in an order-based model, because it is not clear which of the two tasks should be prioritized. Moreover, the stacking order is often more flexible than the retrieval order. For example, typically a group of containers is discharged from the ship and is available for stacking with no difference in priority among the containers in the group. We use this flexibility in our model to reduce the number of relocations.

One of the main contributions of this thesis is to introduce a time-based class of models. Considering that estimates of the departure times and arrival times of containers is often evaluated from historical operations data, we assume that service
times and service time windows for stacking and retrieving each container are given. Based on these inputs, we present a time-based model and a mathematical program that jointly considers the cost-based objective of minimizing the number of relocation moves (including repositioning moves) and the service-based objective of minimizing wait times, and provides the optimal sequence of moves for retrieving containers subject to various service policies. In Section 2.2, we illustrate examples arising in real operations and show that they can be addressed only using time-based models. Moreover in Chapter 2, we present our time-based mathematical program and Integer Programming (IP) formulation for the CRP and an extended model for the DCRP.

1.3 Literature Review

The CRP has been widely studied in the literature. For a recent classification scheme and a comprehensive literature review of the CRP and similar problems that appear in other practical applications, see Lehnfeld and Knust [18]. For a general survey on ports operations literature, see Stahlbock and Voß [23].

Literature on the relocation problem in container terminals follows two main directions. The first approach is to design heuristics and evaluate their performance through numerical experiments on a set of random instances. Many heuristics have been developed for the CRP. Several authors consider heuristics based upon a set of rules that are designed to minimize the number of relocations. Kim and Hong [14] suggest a decision rule that uses an estimate of the expected number of additional relocations. Caserta et al. [7] propose a heuristic that aims to postpone future relocations as much as possible. Extending the objective function to the weighted sum of the crane working time and the number of relocations, Lee and Lee [17] present a three-phase heuristic that generates an initial feasible sequence of moves in the first phase and reduces the number of moves and crane's working time in phases two and three. Also, Caserta et al. [5] propose a binary description of bay configurations and use it in a four-step heuristic.

In other papers, authors use local search methods. Caserta et al. [6] propose
a meta-heuristic based on the corridor method to find the best slot for relocating a container, and Forster and Bortfeldt [10] use a tree search procedure to improve a greedy initial solution. Finally, Petering and Hussein [21] describe a look-ahead heuristic that takes into account the possibility of performing repositioning moves.

Some papers study the stacking problem, i.e., where to put an arriving container. For example, Kim et al. [15] propose a methodology to determine the stacking location of an arriving export container considering its weight. Borgman et al. [2] determine the stacking location by the estimated departure time and the distance from the exit point, and Dekker et al. [9] simulate different stacking policies for containers in automated terminals. In a more recent work, Gharehgozli et al. [11] propose a decision-tree heuristic to minimize the expected number of relocations when arriving containers are stacked in a bay with an arbitrary number of columns.

The second approach in the literature is to develop a mathematical model and formulation for the CRP, and either solve the program or use it to design heuristics. In one of the very first papers, Kim and Hong [14] present a formulation and provide a branch-and-bound algorithm for obtaining the optimal locations for relocated containers. Ünlüyurt and Aydin [25] also use a branch-and-bound approach to solve the problem optimally and propose several heuristics that give near-optimal solutions. Caserta et al. [7] show that the CRP is NP-hard and propose two binary Integer Programming (IP) formulations. Their first IP provides an optimal solution to the non-restricted CRP (i.e., taking into account the repositioning moves) whereas their second formulation is applicable to the restricted CRP. Petering and Hussein [21] propose a mixed IP formulation for the restricted CRP and show that their formulation has fewer decision variables and better runtime performance.

Some papers address the DCRP, i.e. the setting where the containers are stacked and retrieved continuously. Wan et al. [26] propose an IP formulation and apply heuristics based on the static IP formulation to a dynamic setting. Hakan Akyüz and Lee [12] present a binary IP for the DCRP and then propose IP-based heuristics. Rei and Pedroso [22] propose two heuristic methods for the DCRP assuming there is no height limit for the columns. Borjian et al. [4] propose a generalized model and
IP formulation that capture the cost-based objective (number of relocations) and the service-based objective (truck wait time). They present an extension of their model for the DCRP, and also introduce and evaluate a class of flexible retrieval policies.

In addition to mathematical programming formulations, another method that has been used to solve the CRP is the $A^*$ algorithm, which has a branch-and-bound framework. It was first applied to the CRP by Zhang et al. [28] and has also been studied by Zhu et al. [30]. Moreover, Tanaka and Takii [24] propose a new lower bound for the algorithm, and Borjian et al. [3] provide a detailed study of the properties of different bounds used in the $A^*$ algorithm.

A couple of papers perform an average case analysis of the CRP when the bay size grows asymptotically. Olsen and Gross [20] provide an average case analysis when the number of columns grows to infinity and the maximum height of each column can also grow arbitrarily. They show that there exists a polynomial time algorithm that solves this problem close to optimality with high probability. Borjian et al. [3] study the asymptotic behavior of the CRP where the number of columns grows and show that the optimum number of relocations converges to a simple lower-bound.

The literature on CRP with incomplete information is limited. Zhao and Goodchild [29] evaluate the use of truck arrival information to reduce relocations. They study the impact of information quality and bay configuration on the number of relocations through numerical experiments. Borjian et al. [3] study the CRP with incomplete information in a two-stage setting and propose an approximate stochastic algorithm based on the $A^*$ algorithm. They use their algorithm and a myopic heuristic to study the value of information. Ku [16] also proposes a stochastic dynamic programming model and a heuristic considering the information from a truck appointment system.

1.4 Contributions

In this thesis, we study two main variants of the container relocation problem in container terminals: the CRP with complete information, and the CRP with incomplete
For the CRP with complete information, one of our main contributions is a new mathematical model that incorporates practical details, and captures customer-centric service elements, such as wait times. To this end, we develop a new IP formulation that jointly minimizes the number of relocation moves (including repositioning moves) and customer wait times, and provides the sequence of moves for retrieving containers for external trucks based on various service policies. For example, our model can be used by port operators to minimize external truck wait time, or to give different waiting time guarantees to different customers to reflect relative priorities. Moreover, our model expands the capabilities of existing approaches by determining repositioning moves that reduce overall service time, rather than the number of container relocations, thus moving from a focus of cost minimization for the port operators to one of maximization of customer service levels.

In addition, we extend our model and formulation to the DCRP, i.e., the setting where containers are stacked and retrieved continuously. In our model, we relax the strict stacking ordering, and jointly optimize the ordering and the slot for stacking incoming containers.

In the context of the CRP with complete information, we also propose and study a new class of retrieval policies. We relax the first-come-first-served (FCFS) policy and allow for *out-of-order* retrievals. We show this flexible retrieval policy is surprisingly efficient and aligned with the benefits of port operators and customers. We demonstrate the impacts of this policy by measuring the number of relocations and service times. We prove and verify through experiments that such a policy results in fewer relocations and reduced overall service times. Moreover, our experiments show that flexible retrieval planning results in equitable customer services, an intuitively clear notion that we will define in our context.

Our contribution to the literature of the CRP with incomplete information is twofold. First, we develop a 2-stage approximate stochastic optimization algorithm, called $ASA^*$, which to the best of our knowledge is the first optimization algorithm introduced and implemented for the CRP with incomplete information. We also give
theoretical bounds on the approximation errors and show through numerical experiments that our algorithm is fast and efficient for medium-sized instances. Second, we study the value of information using the ASA* algorithm and the myopic heuristics that we design based on existing heuristics for the CRP with complete information. Our experiments show that when the retrieval order of 50% of containers is known at the beginning, the loss from the missing information is negligible and the average number of relocations is very close to that of the CRP with complete information.

1.5 Thesis Outline

This thesis has 5 chapters and is structured as follows.

Chapter 2 is focused on the CRP with complete information. In Section 2.1, we describe the container relocation problem and explain two model classes, time-based and order-based. In Section 2.2, we present a mathematical model and integer programming formulation for the time-based CRP and extend the formulation to the DCRP in Section 2.3. In Section 2.4, we relax the FCFS assumption in the retrieval process and introduce a new class of flexible retrieval policies. In Section 2.5, we review the A* algorithm and its implementation for the CRP. In Section 2.6, we introduce and benchmark a new heuristic for the CRP with complete information. We provide the concluding remarks of Chapter 2 in Section 2.7.

Chapter 3 is focused on the CRP with incomplete information. In Section 3.1 we describe the setting of the problem, and in Section 3.2, we present a stochastic optimization algorithm. In Section 3.3, we introduce three myopic heuristics and evaluate their performance through numerical experiments. Lastly, in Section 3.4, we use our stochastic optimization algorithm and a myopic heuristic to assess the value of information. We provide the concluding remarks of Chapter 3 in Section 3.5.

In Chapter 4 we describe a port operations simulator and provide some simulation results. Finally in Chapter 5, we provide some concluding remarks and directions for future studies.
Chapter 2

CRP with Complete Information

In this chapter we describe the container relocation problem in detail. In Section 2.1, we explain and compare two model classes (time-based and order-based) and illustrate the cases where the order-based model fails to capture the practical details of operations. In Section 2.2 we present a time-based mathematical model and integer programming formulation for the CRP with complete information, that is, the case where departure times of all containers are known in advance and the containers are required to be retrieved in the predefined order dictated by their departure times. We present an extension of the model in Section 2.3 for the dynamic container relocation problem, where containers are stacked and retrieved continuously from the bay. In Section 2.4 we relax the assumption of retrieving containers in a pre-defined order and extend the model for CRP and DCRP to the case that some containers may be retrieved out-of-order. We show that the out-of-order retrieval policy can result in fewer relocations and shorter wait times for external trucks and therefore, is beneficial for both port operators and customers.

In the last two sections of this chapter, we introduce an algorithm and heuristic based on the order-based model for the CRP with complete information. In Section 2.5, we review the $A^*$ algorithm and its implementation for the CRP. In Section 2.6, we introduce a new heuristic for the CRP and evaluate its performance, through numerical experiments, compared to existing heuristics.
2.1 Time-Based and Order-Based Class of Models

We define the CRP as follows. We are given a bay with $C$ columns and $P$ tiers, where containers are stacked on top of each other up to height $P$ (typically four or five) in each column. In a given time horizon (say a day), $N$ containers need to be retrieved from this bay and delivered to external trucks that arrive at the terminal to pick up containers. The departure time of each container is the earliest time that it can be retrieved, and is the same as the arrival time of its corresponding truck. In the CRP, the departure times are assumed to be known beforehand. A container can only be retrieved from the topmost tier, which means that if it is blocked by another container, the blocking container should be relocated to another slot in the bay.  

The solution to the container relocation problem gives the optimal sequence of moves (retrieval, relocation, and repositioning moves) that retrieves all the containers and minimizes the total number of non-productive, that is, relocation plus repositioning, moves.

As mentioned in Section 1.2.1, the critical assumption of order-based models is that they do not take into account the time of the day that each move is performed or the required amount of time for completing each move. Such models take the departure order of containers (as opposed to departure times) as input and focus only on minimizing the number of non-productive moves. In particular, they do not capture service-based objectives like truck wait times. Moreover, order-based models do not capture crane idle times when no truck is awaiting a retrieval. This can be significant in that such idle time allows for container repositioning that will result in increased timeliness of service (reduced wait times) when trucks do arrive.

Our time-based model relies on actual service times rather than departure orders of containers. In order to have the arrival times of the external trucks as an input to our model, we discretize the planning horizon into time steps, where one time step is the minimum time needed to complete one move; a move includes stacking, relocating, repositioning, or retrieving one container. We then translate the arrival times of the external trucks to these time steps.

\footnote{In the non-restricted CRP, a non-blocking container can also be relocated; i.e., a repositioning move is allowed.}
time of the trucks to time steps. For example if the planning horizon is from 9am to 11am and each move takes on average 4 minutes to complete, there will be 30 time steps in the planning horizon. In this example if an external truck arrives at 9:15, its arrival time step would be 4. Note that if a truck arrives in the middle of one time step (say at 9:07, that is, during the second time step), we assign it to the next (third) time step.

In the rest of this section, we explain the flexibility of the time-based model and in Sections 2.1.1-2.1.4, we illustrate cases that arise operationally and that can be handled only by our time-based model. Table 2.1 summarizes the notations that will be used throughout this thesis for describing the bay configuration.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bay size</td>
<td>$C \times P$ A bay with $C$ columns and $P$ tiers.</td>
</tr>
<tr>
<td>Container</td>
<td>$(c_i, d_n)$ A container with label $c_n \in {c_1, c_2, c_3, \ldots }$. The time step at which the external truck corresponding with container $c_n$ arrives (i.e., scheduled departure time of $c_n$), is denoted by $d_n \in {1, 2, 3, \ldots }$. ²</td>
</tr>
<tr>
<td>Slot</td>
<td>$[i, j]$ The slot in the $i^{th}$ column and $j^{th}$ tier of the bay where $1 \leq i \leq C$ and $1 \leq j \leq P$. Tiers are indexed from the bottom such that the lowest and topmost tiers are represented by $j = 1$ and $j = P$, respectively.</td>
</tr>
<tr>
<td>Schedule</td>
<td>$S \in R^N$ The arrival time steps of external trucks that pick up containers $c_1, c_2, \ldots, c_N$.</td>
</tr>
</tbody>
</table>

²Label of containers have no implication about arrival time or order of their corresponding trucks. What imposes the retrieval order is the scheduled departure times: if $d_n < d_{n'}$, then $c_n$ will be retrieved before $c_{n'}$; however for simplicity and without loss of generality, we assume that in such case $n < n'$. 

Table 2.1: Notations
2.1.1 Handling the Idle times and Repositioning Moves

Given the arrival times of external trucks, in the time-based model we can specify the time step that represents the earliest retrieval time for each container. This is important because in practice, port operators perform repositioning moves, often during off-peak hours. Repositioning moves might not decrease the total number of relocations but can result in less wait times and fewer relocations during peak hours. Such moves cannot be modelled in an order-based model because it is not possible to specify the time intervals between the arrival time of the trucks. Example 2.1.1 illustrates a simple case where knowing truck arrival times results in obtaining a solution that includes a repositioning move during the crane idle time.

**Example 2.1.1. Repositioning move:** Consider a $3 \times 3$ bay with 6 containers, with the initial configuration given in Figure 2-1. Let $S_1 = (1, 2, 3, 4, 5, 6)$ and $S_2 = (1, 3, 4, 5, 6, 7)$ be two different schedules for the arrival times of the trucks that will pick up containers $c_1$ to $c_6$, respectively. $S_1$ is a tight schedule in the sense that there is no gap between the arrival times of the trucks. Note that a time step is by definition the minimum time to perform one move. This implies that if two or more moves are required for retrieving a container, all subsequent containers will be retrieved with some delay). $S_2$ is a more flexible schedule because there is one time step gap between arrival times $c_2$ and $c_3$. Solving this example for schedule $S_1$ and for schedule $S_2$, we obtain the following two different solutions:

\[
\begin{array}{ccc}
(c_1, 1) & (c_4, 4) & (c_2, 2) \\
(c_5, 5) & (c_3, 3) & (c_6, 6)
\end{array}
\quad
\begin{array}{ccc}
(c_1, 1) & (c_4, 5) & (c_2, 3) \\
(c_5, 6) & (c_3, 4) & (c_6, 7)
\end{array}
\]

(a) \quad (b)

Figure 2-1: Initial configuration of the bay for Example 2.1.1 where arrival schedule is a) $S_1$, and b) $S_2$

In both cases of Example 2.1.1, one relocation is required to retrieve $c_3$. The sequence of moves is different though, because in the case of $S_2$, the relocation can be performed as a repositioning move at time 2 (before retrieving $c_2$). As shown in Table
2.2, a time-based model would provide different solutions for the two arrival schedules, whereas an order-based model does not distinguish between $S_1$ and $S_2$ and will give the same sequence of moves for both cases. In a solution given by an order-based model, the repositioning move will not be performed at time 2 for schedule $S_2$ and as a result, the external trucks picking up containers $c_3, \ldots, c_6$ will each experience 1 unit of delay. In a solution given by a time-based model, however, all containers are retrieved with no delay.

<table>
<thead>
<tr>
<th>Container</th>
<th>From</th>
<th>To</th>
<th>Container</th>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>($c_1, 1$)</td>
<td>[1,2]</td>
<td>out</td>
<td>($c_1, 1$)</td>
<td>[1,2]</td>
<td>out</td>
</tr>
<tr>
<td>($c_2, 2$)</td>
<td>[3,2]</td>
<td>out</td>
<td>relocate ($c_4, 5$)</td>
<td>[2,2]</td>
<td>[1,2]</td>
</tr>
<tr>
<td>($c_4, 4$)</td>
<td>[2,2]</td>
<td>[1,2]</td>
<td>retrieve ($c_2, 3$)</td>
<td>[3,2]</td>
<td>out</td>
</tr>
<tr>
<td>($c_3, 3$)</td>
<td>[2,1]</td>
<td>out</td>
<td>retrieve ($c_3, 4$)</td>
<td>[2,1]</td>
<td>out</td>
</tr>
<tr>
<td>($c_4, 4$)</td>
<td>[1,2]</td>
<td>out</td>
<td>retrieve ($c_4, 5$)</td>
<td>[1,2]</td>
<td>out</td>
</tr>
<tr>
<td>($c_5, 5$)</td>
<td>[1,1]</td>
<td>out</td>
<td>retrieve ($c_5, 6$)</td>
<td>[1,1]</td>
<td>out</td>
</tr>
<tr>
<td>($c_6, 6$)</td>
<td>[3,1]</td>
<td>out</td>
<td>retrieve ($c_6, 7$)</td>
<td>[3,1]</td>
<td>out</td>
</tr>
</tbody>
</table>

Table 2.2: Solutions of Example 2.1.1 given by time-based model for two different arrival schedules

### 2.1.2 Joint Optimization of Wait time and Relocations

An important efficiency metric, in addition to the number of relocations, is the wait time of external trucks (or the retrieval delay). In an order-based model, the truck wait times cannot be measured or minimized because the arrival time of external trucks is not an input to the model. In a time-based model, the wait time can be measured as the difference between the time that a container is delivered to an external truck (actual departure time) and the time that the truck arrives (scheduled departure time). Also, the wait time can be included in the objective function. For example, the objective function can be defined as the weighted sum of the number of relocation moves and total wait times. Note that these two efficiency measures do not necessarily move in the same direction, meaning that minimizing the number of relocation moves is not necessarily equivalent to minimizing the total wait times.
(or even the maximum wait time for a truck). For some bay configurations, there
might exist multiple solutions that achieve different trade-offs between the number of
relocations and total wait times. Example 2.1.2 illustrates such a trade-off for a $3 \times 3$
block. To jointly optimize both efficiency metrics, we define the objective function as
the weighted sum of total wait times and relocations.

**Example 2.1.2. Joint optimization of relocation and delay:** Consider a $3 \times 3$
bay with 6 containers, with the initial configuration given in Figure 2-2. Let $S =
(1, 3, 4, 5, 6, 7)$ be the arrival times of the trucks that pick up containers $c_1$ to $c_6$. For
this example, the solution minimizing the number of relocations differ from the one
minimizing delay, as shown in Table 2.3.

<table>
<thead>
<tr>
<th>$(c_1, 1)$</th>
<th>$(c_2, 3)$</th>
<th>$(c_5, 6)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(c_4, 5)$</td>
<td>$(c_6, 7)$</td>
<td>$(c_3, 4)$</td>
</tr>
</tbody>
</table>

Figure 2-2: Initial configuration of the bay for Example 2.1.2

In the relocation-minimizing solution, no move is done in the second time step
whereas in the delay-minimizing solution, relocation of $c_5$ is performed during this
time step. This results in less delay for $c_3$ later in the retrieval process but requires
$c_5$ to be relocated again, resulting in an additional relocation compared to solution 1
in Table 2.3.

With the objective function being the weighted sum of the relocation moves and
total delay, the optimal solution would depend on the relative weights assigned to the
two metrics. For example, if we assign equal weights to both terms in the objective
function, both solutions are optimal as they would have the same objective function
value. If we assign a larger weight to relocation than delay, the first solution (with 1
relocation and 4 delay) is optimal and if we assign a larger weight to delay, the second
solution is optimal.
<table>
<thead>
<tr>
<th>Container From To</th>
<th>Container From To</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t=1) retrieve ((c_1, 1)) [1,2] out</td>
<td>retrieve ((c_1, 1)) [1,2] out</td>
</tr>
<tr>
<td>(t=2) idle</td>
<td>relocate ((c_5, 6)) [3,2] [1,2]</td>
</tr>
<tr>
<td>(t=3) retrieve ((c_2, 3)) [2,2] out</td>
<td>retrieve ((c_2, 3)) [2,2] out</td>
</tr>
<tr>
<td>(t=4) relocate ((c_5, 6)) [3,2] [2,2]</td>
<td>relocate ((c_3, 4)) [3,1] out</td>
</tr>
<tr>
<td>(t=5) retrieve ((c_3, 4)) [3,1] out</td>
<td>retrieve ((c_3, 4)) [3,1] out</td>
</tr>
<tr>
<td>(t=6) retrieve ((c_5, 6)) [1,1] out</td>
<td>relocate ((c_5, 6)) [1,2] [2,2]</td>
</tr>
<tr>
<td>(t=7) retrieve ((c_5, 6)) [2,2] out</td>
<td>retrieve ((c_5, 6)) [2,2] out</td>
</tr>
<tr>
<td>(t=8) retrieve ((c_6, 7)) [2,1] out</td>
<td>retrieve ((c_6, 7)) [2,1] out</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Relocations</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total delay</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 2.3: Solutions of Example 2.1.2

2.1.3 Dynamic Container Relocation Problem (DCRP)

One of the critical assumptions of the CRP is that all containers are already in the bay and we only need to retrieve them. In practice, the retrieval and stacking processes can overlap. In other words, at the same time that some containers are retrieved, some other containers need to be stacked in the bay. This is in fact an "integrated retrieval and stacking problem". However, in order to be consistent with the literature, we refer to it as the dynamic container relocation problem.

It is not clear how to model the DCRP using an order-based model, because a single order cannot be specified for all containers (some of which need to be retrieved and some stacked). Even defining two separate orders (for retrieval and stacking) would not work. First, it would not be clear which of the two tasks (retrieval or stacking) should be prioritized when there is an overlap between the two processes. Second, the stacking process is more flexible because there is usually a group of containers that is to be stacked and there is no priority for stacking one container before the others.

In our model, we relax the strict ordering of stacking, and jointly optimize the sequence of moves (which will determine the order of stacking) and the slot selected

---

3 A special case of the DCRP is to find the best slots for stacking incoming containers such that they can be retrieved in the future with a minimum number of relocations.
for incoming containers. Retrievals are performed according to the predefined order. In the stacking process, however, we impose pre-defined orderings among groups of containers (rather than for each container). The rationale for this is that typically, a group of containers is discharged from the vessel and is available for stacking with no difference in priority among the containers in the group. In Section 2.3 we present our extended mathematical model and formulation for the DCRP. We also present an example to illustrate how our DCRP model finds the optimal sequence of stacking and retrieval moves, and exploits the flexibility of the stacking process.

2.1.4 Modelling Operations Flexibility

Time-based models allow for studying different service policies (other than first-come first-served). For example, it is possible to specify in the model the maximum amount of delay for serving each truck. This is especially important if it is desired to serve some trucks more quickly than others. Also, because the model is not based on the order of arrival, it is possible to serve some trucks (retrieve some containers) out of order. Out-of-order retrieval is discussed in detail in Section 2.4.

2.2 Mathematical Model and Formulation for the Time-Based CRP

In this section, we present the mathematical model for the time-based CRP. We also develop an IP formulation that expands on the BRP-I formulation presented in [7].

For describing the bay configuration and the moves, we use the same variables as those in BRP-I. However, unlike BRP-I in which the time index, $t$, is merely for determining the order of the moves, in our model it represents the actual time of the day that each move is performed. Moreover, the input of the model is the arrival time of the external trucks rather than the departure order of containers. The four sets of variables are as follows:
Given the arrival time of the external trucks, we can specify the time-step at which each container is ready to be retrieved. We denote the departure time of container $c_n$ (i.e., the time-step that $c_n$ is ready to be retrieved) by $d_n$. Further, we denote by $\delta_n$ the maximum amount of delay (measured in time-step) that is allowed in retrieving container $c_n$. Note that retrieval delay is the same as the wait time of the external truck, and we will use these two terms interchangeably. Given $d_n$ and $\delta_n$, we define a retrieval time window for each container as $[d_n, d_n + \delta_n]$, and require that containers be retrieved within these time windows. Also, if the departure time of a container is unknown or beyond the planning horizon, we allow it to stay in the bay arbitrarily long by setting its departure time to infinity.

Let $T$ be the total number of time-steps required to retrieve the containers. Because each retrieval task has a due time determined by its service time window, $T$ can be expressed as follows:

$$T = \max_{1 \leq n \leq N, d_n < \infty} \{d_n + \delta_n\}. \quad (2.5)$$

The set of constraints that ensures the configuration is feasible at each time-step and at most one move is performed at each time-step is given by (2.6)-(2.11), which are similar to the constraints in $BRP - I$ [7]. We add Constraints (2.12)-(2.15) to ensure the containers are retrieved in the order given by their indices in the bay, and
within their service time windows.

Constraints (2.6) ensure that at each time-step, each container is either in the yard, or has been retrieved. Constraints (2.7) and (2.8) ensure that each slot is occupied by at most one container, and there are no empty slots between containers in a column. Constraints (2.9) require that at each time-step, at most one move takes place. Constraint (2.10) update the configuration of the yard at time-step \( t \) based on the configuration and the move that took place in time-step \( t-1 \). Constraints (2.11) capture the relations between retrieval variables and their corresponding configuration variables.

Constraints (2.12) ensure that at most \( n-1 \) retrievals are performed before retrieving each container, \((c_n, d_n)\). (Proposition 2.4.1 in Section 2.4 shows that satisfying Constraints (2.12) imply that containers are retrieved in the prescribed order). Constraints (2.13) ensure that each container is retrieved only after the arrival of its corresponding external truck. Constraints (2.14) ensure that containers are retrieved within their allowable time windows, and Constraints (2.15) require that containers are not retrieved at some time beyond their allowed time window.

\[
\sum_{i=1}^{C} \sum_{j=1}^{P} b_{ijnt} + v_{nt} = 1, \quad n = 1, \ldots, N, \quad t = 1, \ldots, T
\]  
(2.6)

\[
\sum_{n=1}^{N} b_{ijnt} \leq 1, \quad i = 1, \ldots, C, \quad j = 1, \ldots, P, \quad t = 1, \ldots, T
\]  
(2.7)

\[
\sum_{n=1}^{N} b_{ijnt} \geq \sum_{n=1}^{N} b_{ij+1nt}, \quad i = 1, \ldots, C, \quad j = 1, \ldots, P - 1, \quad t = 1, \ldots, T
\]  
(2.8)

\[
\sum_{i,k=1}^{C} \sum_{j,l=1}^{P} x_{ijklnt} + \sum_{i=1}^{C} \sum_{j=1}^{P} \sum_{n=1}^{N} y_{ijnt} \leq 1, \quad t = 1, \ldots, T
\]  
(2.9)

\[
b_{ijnt} = b_{ijnt-1} - \sum_{k=1}^{C} \sum_{l=1}^{P} x_{ijklnt-1} + \sum_{k=1}^{C} \sum_{l=1}^{P} x_{klijnt-1} - y_{ijnt-1}
\]  
(2.10)

\[
v_{nt} = \sum_{i=1}^{C} \sum_{j=1}^{P} \sum_{t'=1}^{t-1} y_{ijnt'}, \quad n = 1, \ldots, N, \quad t = 1, \ldots, T
\]  
(2.11)
\( (t - n) \sum_{i=1}^{C} \sum_{j=1}^{P} y_{ijnt} + \sum_{i=1}^{C} \sum_{j=1}^{P} \sum_{n=1}^{N} \sum_{t'=1}^{t-1} y_{ijnt'} \leq t - 1 \)  \( t = 1, \ldots, T, \ n = 1, \ldots, N \) and \( d_1 < d_2 < \cdots < d_N \)

\begin{align*}
\sum_{i=1}^{C} \sum_{j=1}^{P} \sum_{n=1}^{N} y_{ijnt} &= 0, \quad n = 1, \ldots, N \\
\sum_{i=1}^{C} \sum_{j=1}^{P} \sum_{t=d_n}^{d_n+\delta_n} y_{ijnt} &= 1, \quad n = 1, \ldots, N \\
\sum_{i=1}^{C} \sum_{j=1}^{P} \sum_{t=d_n+\delta_n+1}^{T} y_{ijnt} &= 0, \quad n = 1, \ldots, N
\end{align*}

The objective is to jointly minimize the relocations and delay (difference between the time that a container is delivered to an external truck and the time that the truck arrives). We define the objective function as the weighted sum of these two efficiency metrics. The total number of relocation moves can be computed as \( \sum_{i,k=1}^{C} \sum_{j,l=1}^{P} \sum_{n=1}^{N} \sum_{t=1}^{T} x_{ijklnt} \).

Also note that the earlier we retrieve container \( n \), the larger is the summation \( \sum_{t=d_n}^{T} v_{nt} \).

Thus we use the negative of this quantity as a measure of delay in retrieving containers. We also define parameters \( w_{rel} \) and \( w_r \) as the weight factors for relocations and retrieval delays, respectively. The weight factors are set by port operators based on their port policies and sensitivity to the number of relocations and total delay.

The optimization model for the time-based CRP is as follows:

\[
\min \ w_{rel} \sum_{i,k=1}^{C} \sum_{j,l=1}^{P} \sum_{n=1}^{N} \sum_{t=1}^{T} x_{ijklnt} - w_r \sum_{n=1}^{N} \sum_{t=d_n}^{T} v_{nt} \quad (2.16)
\]

\[
s.t. (2.6) - (2.15) \quad \text{and} \quad b, x, y, v \geq 0
\]

One of the main features of the time-based model is that the arrival time of external trucks can be specified. This is important because the trucks do not necessarily arrive uniformly over time (especially during off-peak hours), resulting in crane idle times, i.e., times that the yard crane is not busy with the retrieval process. In prac-
tice, during idle times, port operators may reposition the containers, in order to avoid delay in the retrieval process in the future.

Incorporating “time” into the model, we can account for the idle times and plan repositioning moves based on truck arrival schedules and bay configurations. A way to plan repositioning moves is by considering their effect, on the number of relocations and service delay for future truck arrivals. Through jointly minimizing these two objectives, our time-based model can decide whether to do repositioning at any time, and which repositioning moves are most beneficial. The decision depends on the relative weights of the two objectives ($w_{rel}$ and $w_r$). If $w_{rel}$ is large compared to $w_r$, the optimal decision will be to avoid repositioning or perform only repositioning moves that reduce the total number of relocations. Conversely, if $w_{rel}$ is low compared to $w_r$, it might be optimal to perform some additional relocations (during the repositioning process) in order to further decrease service delay. Example 2.1.2 in Section 2.1.2 illustrates this trade-off between relocation and service delay. Moreover, we investigate the impacts on the port (and customers) to schedule trucks in such a way that containers can be repositioned regularly. To examine this policy, we study the effect of two types of traffic (uniform and non-uniform) on service delay and on the number of relocations.

Consider a $C \times P$ bay with $N$ containers and a given initial configuration. Suppose that there are $N$ external trucks each picking up one container. We study two cases regarding the arrival schedule of the external trucks:

1) Uniform traffic: $N$ external trucks arrive at the yard uniformly and continuously over time, i.e., exactly one truck arrives at each time-step; and

2) Non-uniform traffic: $N$ external trucks arrive at the yard in two batches (two bursts of traffic) and there are no arrivals between the arrival of the two batches.

We generate 500 random instances of a $6 \times 4$ bay with 18 containers, and solve each instance twice; once with uniform and once non-uniform arrival times of the external trucks. In the case of non-uniform traffic, the arrival schedule is such that the first 9 containers arrive at times 1 to 9, and the second batch arrives at times
20 to 28; with no trucks arriving no truck arrives between times 9 and 20. We then compare the average delay per container (over all instances) between the two cases.

Note that the number of relocations required to retrieve the containers is the same in both cases. To study the impact on service delay, we set the objective function as the weighted sum of relocations and retrieval delay with equal weights. For each instance, the moves for retrieving the first batch of containers ($c_1$ to $c_9$) are the same in both cases. In 38 out of 500 instances, no repositioning is performed during the off-peak hours. In other instances, the service delay of containers in the second batch is decreased when there is non-uniform traffic because the containers are repositioned, i.e., some of the relocation moves can be performed during the period without truck arrivals.

Figure 2-3 shows the distribution of the difference of average delay per container (average wait time) for the two cases (uniform traffic delay - non-uniform traffic delay). The red line in the box is the median and the blue dot is the mean of the distribution for the 500 instances. The average delay (over all 500 instances) per container is decreased by 17% when traffic is non-uniform.

In most container terminals, port operators use a scheduling system (also referred
to as an appointment system) for the external trucks. An appointment system allows truck drivers to book a pick-up time (or often a pick-up time-window) in advance. Such a system results in more balanced traffic in the terminal and thereby more timely service, which benefits both customers and port operators. Different scheduling policies can have different impacts on the service delay. In practice, an appointment system can recommend to each truck driver a pick-up time (based on bay configuration and the current arrival schedule of external trucks) so as to reduce waiting time.

The second important feature of the time-based model is the service-time window. Using time windows allows for more flexible constraints and retrieval policies. For instance, it is possible for the port operator to choose time windows such that certain trucks (for example those registered in the appointment system in advance) experience less delay, or to serve some trucks (retrieve some containers) out-of-order with specified maximum amounts of service delay for all (other) trucks. This policy is discussed in detail in Section 2.4. In addition to modelling flexibility, using time windows allows for more efficient computation by shrinking the search space of the IP (because we set many variables to zero by restricting the containers to be retrieved within a time window).

Note that one can set the time windows arbitrarily large without affecting optimality; i.e., any solution to a problem with a smaller time window is also a solution to the problem with larger time windows. However, for computational reasons, we strive to tighten the time windows as much as possible. In short, we are interested in the smallest time window that ensures problem feasibility and optimality. Proposition 2.2.2 gives a bound with such properties for the retrieval time window of each container ($\delta_n^*$), for a container relocation problem with the objective to minimize the total number of relocations. We use the following lemma to prove Proposition 2.2.2.

**Lemma 2.2.1.** Consider a CRP with the objective to minimize the total number of relocation moves. There exists an optimal sequence of moves, $M^*$, such that if at any time-step, $t$, there is a container that is ready to be retrieved, that time-step is not idle.
Proof. Suppose $M$ is an optimal sequence of moves, and let $M(t)$ be the move of time $t$. Also, suppose there exists a time step $\bar{t}$, and a container $(c_{\bar{n}}, d_{\bar{n}})$ such that: i) $M(\bar{t})$ is idle; and ii) $c_{\bar{n}}$ is in the bay at time $\bar{t}$, and $d_{\bar{n}} \leq \bar{t}$. Given $M$, we can construct another optimal solution, $M'$, as follows:

We swap the two moves, $M(\bar{t})$ and $M(\bar{t} + 1)$. As a result, $M(\bar{t} + 1)$ becomes idle. We repeat the swapping for $M(\bar{t} + 1)$ and $M(\bar{t} + 2)$, if conditions (i) and (ii) are satisfied for $M(\bar{t} + 1)$. We continue this process until we reach a time step that does not satisfy the two conditions. In the resulting sequence of moves, $M'$, the idle time step that was initially at $\bar{t}$ is postponed to a later time step (that does not satisfy conditions (i) and (ii)). Next we show that $M'$ is feasible and optimal.

First note that an idle time step does not make any changes in the configuration; therefore adding or removing or shifting an idle time does not violate any of the constraints that involve feasibility of configuration. For the same reason, constraints 2.9 that ensure at most one move is performed at each time step, will not be violated. Secondly, note that by postponing the idle time step, the retrieval time of a container in $M'$ can only be earlier than its retrieval time in $M$. Moreover, because we check condition (ii) for each swapping, it is ensured that a container is retrieved only after its departure time. Thus, all retrievals in $M'$ are within the allowable time windows. Lastly, the containers are retrieved in the prescribed order in $M'$, because we only swap an idle time step with its next move, and such a swapping does not affect the initial retrieval order of containers in $M$.

The resulting sequence of moves, $M'$, is also optimal; because the only difference between $M$ and $M'$ is that the order of some moves are different, which does not affect the total number of relocation moves.

Proposition 2.2.2. Consider a bay with $N$ containers, where $d_1 < d_2 < \cdots < d_N$. Suppose the objective is to minimize the total number of relocation moves, $R^*$ is the optimal value when $\delta_n$ is arbitrarily large, and $H \geq R^*$ is an upper bound given by any heuristic (easily computable). We can reduce $\delta_n$ to $\delta^*_n$, given by (2.17), without
increasing the number of relocations.

\[ \delta^*_n = H + (n - d_n)^+ . \]  (2.17)

**Proof.** In the time interval \([1, d_n + \delta_n]\), two types of moves (relocation and retrieval) or idle time-steps are possible. By definition, exactly \(n\) containers must be retrieved by the end of time \(d_n + \delta_n\). Moreover, \(H\) provides an upper bound on the number of relocation moves (\(H\) can be computed using existing fast heuristics).

To count the idle time-steps in the time interval \([1, d_n + \delta_n]\), we use Lemma 2.2.1 in the following way: Suppose container \((c_n, d_n)\) is retrieved at time \(\hat{d}_n \geq d_n\). The case that \(\hat{d}_n = d_n\) is trivial because \(\delta_n\) is zero, and any bound obtained for \(\delta_n\) would be valid. If \(\hat{d}_n > d_n\), then we know that in the time interval \([d_n, \hat{d}_n]\), container \(c_n\) is available to be retrieved; so there are no idle time-steps in this interval. Also, in the time interval \([1, d_n - 1]\) there are at least \(n - 1\) time-steps where a container is available to be retrieved, which means there are at most \(d_n - 1 - (n - 1)\) non-idle time-steps. Therefore, in the time interval \([1, d_n + \delta_n]\), there are at most \(d_n - n\) idle time-steps (note that \(\hat{d}_n = d_n + \delta_n\) because we have a solution in which \(c_n\) is retrieved at \(\hat{d}_n\)). By counting the maximum number of relocations, retrievals, and the idle time-steps, we get an upper bound on the retrieval time window as follows:

\[ d_n + \delta_n \leq n + H + \max(d_n - n, 0) \Rightarrow \delta_n \leq H + (n - d_n)^+ . \]  (2.18)

Note that \(\delta^*_n\) given by (2.17) is the largest required time window. Also, it might be tight for some containers, but not for all.

\[ \square \]

### 2.3 Mathematical Model and Formulation for the DCRP

In this Section, we extend the time-based model and formulation to the DCRP and illustrate how our model can find an optimal sequence of stacking and retrieval moves...
and take advantage of flexibility in the stacking process.

We define movement and configuration variables for the incoming containers in (2.19) and (2.20), and add Constraints (2.21) to capture the relationship between stacking variables and their corresponding configuration variables.

\[
\begin{align*}
  s_{ijnt} &= \begin{cases} 
    1 & \text{if container } n \text{ is stacked in } [i,j] \text{ at time } t, \\
    0 & \text{otherwise}; 
  \end{cases} \\
  z_{nt} &= \begin{cases} 
    1 & \text{if container } n \text{ has been stacked at time } t' \in \{1, \ldots, t-1\}, \\
    0 & \text{otherwise}; 
  \end{cases} \\
  z_{nt} &= \sum_{i=1}^{C} \sum_{j=1}^{P} \sum_{t'=1}^{t-1} s_{ijnt'}, \quad n = 1, \ldots, N, \quad t = 1, \ldots, T. \quad (2.21)
\end{align*}
\]

We also replace Constraint (2.6) with Constraint (2.22) to ensure that at each time-step, each container is either in the bay, outside the bay waiting to be stacked, or has been retrieved and delivered to an external truck.

\[
\sum_{i=1}^{C} \sum_{j=1}^{P} b_{ijnt} + v_{nt} = z_{nt}, \quad n = 1, \ldots, N, \quad t = 1, \ldots, T. \quad (2.22)
\]

Moreover, we define \( a_n \) as the arrival time of incoming container \( n \) (the earliest time that it is ready to be stacked), and \( \alpha_n \) as the maximum amount of delay that is allowed in stacking container \( n \), similar to \( d_n \) and \( \delta_n \) respectively for the arrival time of external trucks and retrieval time window. Note that the maximum number of required time-steps \( (T) \) is now given by 2.23:

\[
T = \max_{1 \leq n \leq N, 1 \leq m \leq N, d_m < \infty} \{a_n + \alpha_n, d_m + \delta_m\}. \quad (2.23)
\]

We also add Constraints (2.24) to (2.26). Constraints (2.24) ensure that each container is stacked only after its scheduled arrival time. Constraints (2.25) ensure that containers are stacked within their allowable time windows, and Constraints (2.26) re-
quire that containers are not stacked at some time beyond their allowed time window. Moreover, we add stacking moves for updating bay configuration and replace Constraints (2.10) with Constraints (2.27). Lastly, Constraints (2.9) are replaced with Constraints (2.28) to ensure that, at each time-step, at most one move (retrieval, staking, or relocation) is performed.

\[
\sum_{i=1}^{C} \sum_{j=1}^{P} \sum_{t=1}^{a_n-1} s_{ijnt} = 0, \quad n = 1, \ldots, N
\]  
(2.24)

\[
\sum_{i=1}^{C} \sum_{j=1}^{P} \sum_{t=a_n}^{a_n+\alpha_n} s_{ijnt} = 1, \quad n = 1, \ldots, N
\]  
(2.25)

\[
\sum_{i=1}^{C} \sum_{j=1}^{P} \sum_{t=a_n+\alpha_n}^{T} s_{ijnt} = 0, \quad n = 1, \ldots, N
\]  
(2.26)

\[
b_{ijnt} = b_{ijnt-1} - \sum_{k=1}^{C} \sum_{l=1}^{P} x_{ijklnt-1} + \sum_{k=1}^{C} \sum_{l=1}^{P} x_{klijnt-1} - y_{ijnt-1} + s_{ijnt-1}
\]  
(2.27)

\[
\sum_{i=1}^{C} \sum_{j=1}^{P} \sum_{n=1}^{N} x_{ijklnt} + \sum_{i=1}^{C} \sum_{j=1}^{P} \sum_{n=1}^{N} y_{ijnt} + \sum_{i=1}^{C} \sum_{j=1}^{P} \sum_{n=1}^{N} s_{ijnt} \leq 1, \quad t = 1, \ldots, T.
\]  
(2.28)

Note that we do not need to define any order for stacking containers or sequencing stacking and retrieval moves; instead, we model the stacking and retrieval processes by specifying departure and arrival times, and time windows. Retrieval moves are performed in the predefined order (due to Constraints (2.12)). It is possible to enforce stacking moves to follow a predefined ordering as well; however, in the stacking process usually a batch of containers (as opposed to a single container) is discharged from a vessel and all are available to be stacked with no difference in their order. This is an important modelling issue that cannot be captured by specifying stacking order. In our proposed time-based model, we assign the same stacking time to all containers available for stacking at the same time and add Constraints (2.29) to respect the stacking order of groups of containers. The actual order in which the containers within a group are stacked is enforced by minimizing the number of relocations and total delay for the stacking and retrieval processes:
For the DCRP, we add stacking delay and its corresponding weight to the objective function, the general form of which is given as follows:

\[
\sum_{t=1}^{T} z_{nt} \geq \sum_{t=1}^{T} z_{n't}, \quad \forall n, n' \quad s.t. \quad a_{n'} = \min\{a_i | a_i > a_n\}. \quad (2.29)
\]

In the following example, we illustrate a case applying the above model to find the best slots for stacking containers in a 3 x 4 bay such that the sum of total delay and number of relocations is minimized. A complete formulation of the DCRP is provided in Appendix D.

Example 2.3.1. The dynamic container relocation problem: Consider a set of 9 containers \((c_1, \ldots, c_9)\) that need to be stacked in a 3 x 4 bay, which is initially empty. The stacking schedule is given by \(S_s = (1, 1, 1, 1, 1, 2, 2, 2, 2)\), which implies that the first five containers have to be stacked before the last four containers, but the containers in each group do not have to be stacked in a particular order (this is enforced by Constraint 2.29). The retrieval process starts after the stacking process (i.e. there is no overlap between the two processes in this example). During the retrieval process, containers \(c_1\) to \(c_9\) will be picked up by external trucks whose arrival schedule is given by \(S_r = (15, 19, 18, 20, 16, 17, 21, 22, 23)\). The objective is sum of retrieval delay, stacking delay, and total number of relocations. The solution to this problem is the sequence of moves for stacking and later retrieving the containers, as well as the relocation moves. Table 2.4 summarizes the solution.

As shown in Table 2.4, the optimal sequence of stacking is \(c_6, c_1, c_4, c_2, c_3\) for the containers of the first group and \(c_9, c_8, c_7, c_6\) for containers of the second group. With this stacking plan, the retrieval process has no relocation moves and the total retrieval
Table 2.4: Sequence of stacking and retrieval moves

<table>
<thead>
<tr>
<th>Stacking process</th>
<th>Retrieval process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container</td>
<td>Stacked in</td>
</tr>
<tr>
<td>t=1</td>
<td>c_5</td>
</tr>
<tr>
<td>t=2</td>
<td>c_1</td>
</tr>
<tr>
<td>t=3</td>
<td>c_4</td>
</tr>
<tr>
<td>t=4</td>
<td>c_2</td>
</tr>
<tr>
<td>t=6</td>
<td>c_3</td>
</tr>
<tr>
<td>t=7</td>
<td>c_9</td>
</tr>
<tr>
<td>t=8</td>
<td>c_8</td>
</tr>
<tr>
<td>t=9</td>
<td>c_7</td>
</tr>
<tr>
<td>t=10</td>
<td>c_6</td>
</tr>
</tbody>
</table>

delay is zero. If we had to specify a fixed stacking order as input to the model, the retrieval delay and relocations could be more because it is not trivial to specify the stacking order that minimizes the objective. For example, if we set the stacking order as c_1, c_2, c_3, c_4, c_5, c_9, c_8, c_7, c_6, total relocations and total retrieval delay is 3 and 23, respectively. Notice that as shown in this example, the time-based model for DCRP can also be used for determining the best slot to stack a container given its pick-up time in the future.

Throughout this section, our model and formulation assumed a first-come-first-served policy (similar to existing models in the literature and typical practices in container terminals). In the next section, we relax this assumption and modify the formulation to allow for out-of-order retrievals. We show that this retrieval policy helps reduce the number of relocations and retrieval delay, and therefore, can benefit both port operators and customers.

### 2.4 Out-of-Order Retrieval Policy

The formulation presented in Section 2.2 requires that containers be retrieved in a pre-defined order, a first-come-first-served policy, which is enforced by Constraints (2.12).

In this section, we examine the FCFS in the retrieval process and propose a more
general class of service policies which allows for flexibility in the order of retrievals.

Suppose that there are a number of trucks in the storage yard waiting in a queue for containers to be retrieved and delivered to them. If the trucks are served based on a FCFS policy, the containers have to be retrieved in the order dictated by the arrival order of the trucks. As an alternative to FCFS, we introduce the following flexible retrieval policy: Suppose a container that needs to be relocated during the retrieval process (because it is blocking a target container) is ready to depart, i.e., its corresponding external truck has arrived and is waiting somewhere in the queue. An alternative to relocating that container is to retrieve it out-of-order. We refer to this as a flexible retrieval planning policy. To avoid inequity and customer dissatisfaction, the level of flexibility should be defined and controlled to limit the number of out-of-order retrievals.

We define the level of flexibility, \( m \), as follows: for any container, \((c_n, d_n)\), at most \( m \) containers whose departure time is greater than \( d_n \), can be retrieved before \( c_n \).

For example if \( m = 1 \), before retrieving container \((c_n, d_n)\), at most one container whose departure time is greater than \( d_n \) may be retrieved. By definition, the FCFS policy is equivalent to \( m = 0 \).

We add the following constraints to allow for out-of-order retrievals and set the level of flexibility \( m \) in the model (all other constraints remain the same as before). For a bay with \( N \) containers and \( d_1 < d_2 < \cdots < d_N \), Constraints (2.31) ensure that at most \( n - 1 + m \) retrievals are performed before retrieving container \((d_n, c_n)\).

The following proposition shows that if Constraints (2.31) are satisfied, each external truck experiences at most \( m \) out-of-order retrievals.

\[
(t - m - n) \sum_{i=1}^{C} \sum_{j=1}^{P} y_{ijnt} + \sum_{i=1}^{C} \sum_{j=1}^{P} \sum_{n=1}^{N} \sum_{t'=1}^{t-1} y_{ijnt'} \leq t - 1, \\
t = 1, \ldots, T, \quad n = 1, \ldots, N \quad \text{and} \quad d_1 < d_2 < \cdots < d_N. 
\tag{2.31}
\]

**Proposition 2.4.1.** For a CRP with \( N \) containers, suppose that \( d_1 < d_2 < \cdots < d_N \). If before each container, \((c_n, d_n)\), at most \((n - 1) + m\) containers are retrieved, then
the number of out-of-order retrievals before each container is at most m.

Proof. Proof is by contradiction. Suppose there is at least one container with more than m out-of-order retrievals, and let n be one such container with m + 1 out-of-order retrievals. Since there are m + 1 out-of-order retrievals and we assumed that at most \((n - 1) + m\) containers are retrieved before container \((n, d_n)\), we know that the number of containers whose departure times are smaller than \(d_n\) and are retrieved before container \((n, d_n)\), is at most \((n - 1) + m - (m + 1) = n - 2\). Therefore, at least one container (from the set \{\((1, d_1), (2, d_2), \ldots , (n - 1, d_{n-1})\)\}) is retrieved after \((n, d_n)\). Without loss of generality, let \((n - 1, d_{n-1})\) be the container that is going to be retrieved after \((n, d_n)\). Note that at least \(n - 1 + m\) retrievals (including \(m + 1\) out-of-order) take place before retrieving container \((n - 1, d_{n-1})\), and this contradicts our assumption that before container \(n - 1\), we have at most \(((n - 1) - 1) + m = n - 2 + m\) retrievals.

The condition of having at most \((n - 1) + m\) retrievals before each container \(n\) is enforced by Constraint (2.31). These constraints ensure that if container \(n\) is being retrieved at time \(t\), then the number of retrievals performed up to time \(t\) does not exceed \((n - 1) + m\). Note that the first term in the left hand side is \((t - m - n) \times 1\) whenever container \(n\) is being retrieved at time \(t\) and therefore the constraint is reduced to \((t - m - n) + \sum_{i=1}^{C} \sum_{j=1}^{P} \sum_{n=1}^{N} \sum_{t'=1}^{t-1} y_{ijnt'} \leq t - 1\), enforcing the sum of retrievals up to time \(t\) to be at most \((n - 1) + m\). When container \(n\) is not being retrieved at time \(t\), the first term in the left hand side is zero and the constraint is reduced to \(\sum_{i=1}^{C} \sum_{j=1}^{P} \sum_{n=1}^{N} \sum_{t'=1}^{t-1} y_{ijnt'} \leq t - 1\) which is simply a loose bound (and a redundant constraint) on the number of retrieval moves up to time \(t\).

Example 2.4.2 below compares, for the two cases (with \(m = 0\) and \(m = 1\)), the number of relocations, the retrieval delay for each container, and the total delay.

Example 2.4.2. Out-of-order retrieval: Consider a \(3 \times 4\) bay with 9 containers with the initial configuration given in Figure 2-4. The arrival schedule of the external trucks that pick up the containers is given by \(S = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}\).
Figure 2-4: Initial configuration of the bay for Example 2.4.2

We solve this instance for two cases of $m = 0$ and $m = 1$ and compare the sequence of moves as shown in Figure 2-5a and 2-5b. For simplicity, the departure times of containers are not shown in the sequence of moves, and the containers are indicated by their indices.

(a) Sequence of moves for the bay of Example 2.4.2: inflexible case ($m = 0$, FCFS)

(b) Sequence of moves for the bay of Example 2.4.2: flexible case ($m = 1$)

Figure 2-5: An example of an out-of-order retrieval policy

The first two moves for both cases are relocating container $c_4$ to the first column and retrieving container $c_1$ from slot $[3, 2]$. Next, there are two decisions that are
made differently in the case of $m = 0$ and $m = 1$:

1) After retrieving $c_1$, container $c_2$, which is covered by $c_3$, is next to be retrieved if a FCFS policy is used. But in a flexible service policy, $c_3$ can be retrieved and loaded onto its truck (because 2 moves have been performed, it is time 3). In the case of $m = 0$, $c_3$ is relocated to the third column. In the case of $m = 1$, $c_3$ is retrieved out-of-order (before container $c_2$).

2) After retrieving $c_4$, container $c_5$ is next to be retrieved in a FCFS policy. But in a flexible policy, $c_6$ can be retrieved, because 5 moves (in the case of $m = 1$) or 6 moves (in the case of $m = 0$) have already been performed and it is time 6 or 7. In the case of $m = 0$, container $c_6$ is relocated to the third column. In the case of $m = 1$, $c_6$ is retrieved out-of-order (before container $c_5$).

The results of these two different decisions are that the number of relocations, the total delay, and individual delays (denoted by $w_i$ in Table 2.5) are reduced in the case of $m = 1$ compared to $m = 0$. Note that the number of out-of-order retrievals before each container is at most 1 (although the total number of such retrievals is 2 in this example). In fact, in this example, the trucks that pick up containers $c_1$, $c_3$, $c_4$, $c_6$, $c_7$, $c_8$, and $c_9$ do not experience any out-of-order retrievals. Another important point to note is that even for the trucks that experience 1 out-of-order retrieval (that is, trucks that pick up containers $c_2$ and $c_5$), their wait time decreases by 1 and 2 units, respectively.

Table 2.5 summarizes the results for two levels of retrieval process flexibility. Individual delays ($w_1$ to $w_9$) denote the wait time for trucks picking up containers $c_1$ to $c_9$. The total number of relocations and total delay drop by a significant amount (67% and 32%, respectively) when we allow for out-of-order retrieval. Moreover, each truck has a shorter wait time when such a policy is in effect.

In general, we can always obtain an improved or the same quality solution by allowing out-of-order retrievals. As shown in proposition 2.4.3, given any feasible sequence of moves, we can construct another sequence of moves that has the same or
Table 2.5: Number of relocations and delay for two levels of flexibility in Example 2.4.2

fewer relocations and the same or less retrieval delay, if any out-of-order retrieval is possible.

**Proposition 2.4.3.** Suppose \( M \) is a feasible sequence of moves for CRP with \( m \geq 0 \) out-of-order retrievals. Given \( M \), we can construct a new sequence of moves, \( M' \), that is feasible for CRP with \( m + 1 \) out-of-order retrievals and,

(a) The total number of relocations in \( M' \) is at most the number of relocations in \( M \);

and

(b) The delay of all containers for \( M' \) is at most the delay in \( M \).

**Proof.** The procedure for constructing \( M' \), similar to that illustrated in Example (2.4.2), is to retrieve a container that needs to be relocated but is ready to depart.

Suppose the total number of relocations in \( M \) is \( R \). For all \( 1 < r < R \), we define \( t(r) \) as the time of the \( r^{th} \) relocation, and \( n(r) \) as the index of the container that is relocated at \( t(r) \). Also, we denote the number of out-of-order retrievals that take place before retrieving container \( c_n \), by \( \sigma(c_n) \).

To construct \( M' \), we replace each relocation \( 1 \leq r \leq R \) with a retrieval, whenever the following two conditions are satisfied: i) \( d_{n(r)} \leq t(r) \); and ii) \( \sigma(c_i) < m + 1 \), for any container \( (c_i, d_i) \) such that \( d_i < d_{n(r)} \) and \( c_i \) is still in the bay. In other words, if a relocated container is ready to depart (condition i), we retrieve it rather than relocate it, provided that the number of out-of-order retrievals for each container does not exceed \( m + 1 \) (condition ii). Suppose relocation \( \hat{r} \) satisfies condition i and ii; thus we retrieve \( c_{n(\hat{r})} \) at \( t(\hat{r}) \). For the remaining moves in the sequence (after \( t(\hat{r}) \)), we form \( M' \) by replacing any moves that involve \( n(\hat{r}) \) with an idle time-step.
Now we show that \( M' \) is a feasible solution with \( m + 1 \) out-of-order retrievals. In terms of retrieval times, the only difference between \( M \) and \( M' \) is that some containers are retrieved earlier; for such containers, condition (i) ensures that they are retrieved only after their departure time. For the resulting configuration to remain feasible at each time-step, adjustments might be necessary. For example, suppose that there is a move in \( M \) at \( t > t(\hat{r}) \) to relocate a container to slot \([i, j]\), which happens to be on top of \( n(\hat{r}) \). In \( M' \), we perform the same move except that we relocate the container to slot \([i, j - 1]\), because container \( n(\hat{r}) \) in \( M' \) is not in the bay after time \( t(\hat{r}) \). Similarly, some of the retrieval moves in \( M' \) will be from one slot lower. Finally, condition (ii) ensures that the number of out-of-order retrievals for each container does not exceed \( m + 1 \).

Next, we show that statement (a) and (b) hold for \( M' \): If there exists at least one \( \hat{r} \) that satisfies (i) and (ii), then the number of relocations is decreased by at least one. Moreover, because all containers in \( M' \) are retrieved earlier or at the same time as in \( M \), the delay of each container is at most its delay in \( M \).

Finally, using Lemma (2.2.1), some of the idle times in \( M' \) can possibly be eliminated; thus, the delay of some containers in \( M' \) can be reduced below the delay in \( M \).

In the next section, we study computationally the effect of out-of-order retrievals on the number of relocation moves and retrieval delay.

### 2.4.1 Computational Experiments for a Flexible Retrieval Policy

To evaluate the out-of-order retrieval policy, we perform numerical experiments using the integer program introduced in Section 2.2. The main goal of the computational experiments is to study the following questions: What is the quantitative impact of an out-of-order retrieval policy on the number of relocation moves and on the average retrieval delay?; how does the improvement change as we vary bay size?; and what is the impact of an out-of-order retrieval policy on service equity?
Effect of bay size on the number of relocations and delay in an out-of-order retrieval policy: To study the impact of bay size on the number of relocations and delay, we solve random instances for five different bay sizes with 4, 5, 6, 7, and 8 columns. For each bay size, there are 4 tiers in each column. The initial configuration of each random instance is such that the first 3 tiers are full and there is no container in the top tier; so there are initially 12, 15, 18, 21 and 24 containers in the bay, respectively. To evaluate the impact of different levels of flexibility, for each of the 5 bay sizes, we solve the random instances $m$ in constraint (2.31) set to 0, 1, and 2. Changing the bay size and the level of flexibility, there are 15 different test settings. We solve 1000 random instances for each test setting and take the average over the instances to compute the number of relocations and average delay (per truck).

Figure 2-6 shows the results of our numerical experiments. The bay size (the number of columns in the bay) is shown on the horizontal axis. The percent decrease in the total number of relocations is shown in Figure 2-6a, and the percent decrease in the average delay per truck is shown in Figure 2-6b. In both plots, the percent decrease is computed relative to the base ($m = 0$, the FCFS policy) for two levels of flexibility.

![Graphs showing effect of bay size on relocations and delay](image)

Figure 2-6: Effect of an out-of-order retrieval policy on the total number of relocations and on average customer wait time

As can be seen in Figure 2-6a, 1 out-of-order retrieval reduces the number of
relocation moves by 32% in a bay of size 4 × 4 and by 23% in a bay of size 4 × 8. Note that the decrease in the number of relocations depends on the bay size. The percent decrease becomes smaller as the bay gets larger, but with a decreasing rate. To understand why, note that an out-of-order retrieval of container \((c_n, d_n)\) has two benefits: first, we avoid relocating \(c_n\), and thus reduce the total number of relocations by 1. Second, we avoid future relocations that might be incurred by relocating \(c_n\) (if we relocate \(c_n\) to a “bad” column that has a container with higher priority, we need to relocate \(c_n\) again later). When the bay gets larger, it becomes more likely that we find a “good” column (a column that has no container with departure time smaller than \(d_n\)) for relocating \(c_n\). Therefore, the beneficial effect of out-of-order retrieval diminishes as the bay gets larger.

The top line in Figure 2-6a illustrates that allowing for 2 out-of-order retrievals has a larger effect on the number of relocations. The relative decrease is 37% and 31% for a bay of size 4 × 4 and 4 × 8, respectively.

Figure 2-6b depicts the impact of an out-of-order retrieval policy on the average delay (per truck). A similar trend can be seen for the relative decrease in the average delay. Allowing for 1 out-of-order retrieval results in a 32% decrease for a 4 × 4 bay and a 22% decrease for a 4 × 8 bay.

Effect of an out-of-order retrieval policy on service equity: When a FCFS policy is replaced by an out-of-order retrieval policy, truck drivers might perceive service to be unfair when another truck is served out-of-order before them.

However, the inequity perception can go either way: a truck can receive its container out-of-order or can experience an out-of-order retrieval before it is served. To formalize this, we compute the number of out-of-order retrievals performed before serving a particular truck as follows:

For each random instance of a test setting, we randomly select a container and refer to the truck that receives that container as \(r\). We then check that in each of the random instances, how many out-of-order retrievals occur before truck \(r\) receives its container.
Figure 2-7: Long term histogram of out-of-order retrieval moves performed before truck $r$ is served: negative numbers indicate that truck $r$ is served out-of-order; positive numbers indicate that other truck(s) are served out-of-order; zero indicates that neither of these cases happen.

Figure 2-7 shows the distribution of the number of out-of-order retrievals for truck $r$, for two bay sizes (a small bay with 4 columns and a large bay with 8 columns), and two levels of flexibility ($m = 1$ and $m = 2$). The integer numbers on the vertical axis show the number of out-of-order retrieval moves that truck $r$ experiences in the long term and the horizontal axis shows the frequency of each case happening. In a flexible retrieval process, three cases can occur when truck $r$ is served:

- Truck $r$ is not served out-of-order, and does not experience any out-of-order retrieval. This case corresponds to number zero on the vertical line.

- One or more trucks that are supposed to be served after $r$, receive their container out-of-order before $r$ (a truck that is served out-of-order and also experiences some out-of-order retrievals is also classified in this group). This case corresponds to the
positive numbers on the vertical axis. The positive numbers indicate the number of
trucks that are served out-of-order; note that the maximum of these positive numbers
depends on \( m \). For example in Figures 2-7a and 2-7c, only +1 is on the vertical axis
because \( m = 1 \). In Figures 2-7b and 2-7d, there are two positive values (+1 and +2)
because \( m = 2 \).

- Truck \( r \) is served out-of-order. For example it is the \( n^{th} \) truck in the line, but
is served as the \( m^{th} \) truck where \( m < n \). This case corresponds to negative numbers
on the vertical line. The value of the negative numbers indicates how much earlier
truck \( r \) is served. For example the case that truck \( r \) is the \( n^{th} \) truck and is served as
the \( m^{th} \) truck, corresponds to \( m - n \). Similarly, the case that truck \( r \) is served as the
third truck is indicated by value \(-2\).

As shown in Figure 2-7, more than 40% of the time truck \( r \) does not experience
out-of-order service for another truck and is also not served out-of-order.

Moreover, the frequency that truck \( r \) is served out-of-order (negative numbers) is
almost the same as the frequency that it experiences out-of-order retrievals (positive
numbers), when \( m = 1 \). These experiments imply that in the long term, the service
that each truck receives is not adversely affected by of the out-of-order retrieval policy.
Further, as shown in Figure 2-6b, the average retrieval delay (per truck) is decreased
as a result of out-of-order retrievals.

### 2.5 The \( A^* \) Algorithm

For the sake of completeness, we first present the \( A^* \) algorithm applied to the CRP,
previously introduce by [30]. The algorithm is described in Figure 2-8 [3].

The \( A^* \) algorithm is a branch-and-bound method that builds a tree for the se-
quence of decisions. Let us first define the input and output of the algorithm and
some notation.

The inputs to the \( A^* \) algorithm are as follows:

- \( B \) is the bay with given initial configuration to be solved optimally. We denote
the optimal solution (the minimum number of relocations for retrieving containers

58
from $B$) by $z_{opt}(B)$.

- $R(D)$ and $S(D)$ are functions that provide upper and lower bounds, respectively, on $z_{opt}(D)$ for any configuration $D$.

- $\mathcal{N}$ is the maximum number of nodes in the decision tree.

The output of the $A^*$ algorithm is as follows:

- $z_{A^*}$ is the best incumbent found by $A^*$.
- $\gamma$ is the gap between $z_{A^*}$ and $z_{opt}(B)$. Notice that if the algorithm did not reach $\mathcal{N}$ nodes, then the algorithm found the optimal solution (i.e. $\gamma = 0$ and $z_{A^*} = z_{opt}(B)$).

We denote the level of the tree by $l$. A node is at level $l$ if $l$ relocations have been done on the initial bay to get to this node (at $l = 0$ the tree has one node, which is the initial bay, $B$). Moreover, we denote any configuration that is at level $l$ by $B^l$. Also, we define the cumulative upper and lower bounds as $U(B^l) = R(B^l) + l$ and $L(B^l) = S(B^l) + l$ (line 10 of Algorithm 1). Notice that by definition, $S(B^l) \leq z_{opt}(B^l) \leq R(B^l)$. Moreover, $L(B^l) \leq z_{opt}(B) \leq U(B^l)$, for all $B^l$ in the $A^*$ tree.

At each level of the tree and for each node, the task is to retrieve the target container which we denote by $n$. If $n$ is not blocked, it can be retrieved and no relocation is needed, in which case no new node is created. If $n$ is blocked by one or more containers, the topmost blocking containers needs to be relocated to another column. In such a case, a new level will be added to the tree and a new node is created in the new level for each possible move. Once all containers are retrieved (at least on one of the paths), the optimal number of relocations is the number of levels and the optimal sequence of moves is the sequence of decisions on that path.

The size of the tree grows exponentially with the number of levels in the tree and the number of nodes at each level. To overcome this, the $A^*$ algorithm uses a decision rule to prune the tree with the insurance of keeping the optimal path in the tree. At each level of the tree and for each node, the algorithm uses a lower and upper bound and the following rules to decide about pruning or keeping the node (with the guarantee of not removing the optimal path):

(i) If $L(B^l) = U(B^l)$, then $z_{opt}(B^l) = U(B^l)$ and we can stop branching on this
path. From this point, we can simply follow the feasible solution given by the upper bound.

(ii) If $L(B') \geq z_{A^*}$ then this node can be pruned because the optimal solution for this node is going to be greater than its lower bound, hence greater than $z_{A^*}$. The best incumbent is updated if a $B'$ such that $U(B') < z_{A^*}$ is found (lines 12, 13 of Algorithm 1).

The size of the $A^*$ tree depends on the initial bay size and whether the bay is easy to solve or not. Since the CRP is NP hard (as shown by Caserta et al. [7]), for some instances the number of nodes is too large to find the optimal solution. For such instances, we can set a maximum number of nodes allowed in the $A^*$ tree and return our best incumbent if the number of nodes reaches $N$ (notice that best incumbent is useful as a solution only if the upper bound used by the algorithm provides a feasible solution). Although the optimality is not insured in this case, we can provide a guaranteed maximal gap with the optimal solution. The gap is given by $\gamma = z_{A^*} - L_{min}$ (line 24 of Algorithm 1).

The size of the $A^*$ tree also varies according to the local lower and upper bounds used for pruning the nodes. The closer the bounds are to the optimal solution at each node, the faster the tree is pruned hence the faster it solves. In this thesis, we use the heuristic (introduced by Caserta et al. [7]) to compute the upper bound, and we use two types of lower bounds: counting and look-ahead. A brief description of the upper and lower bounds and their properties are given below. For the proofs and detailed study of these bounds, see [3].

2.5.1 Lower Bound

Counting lower bound: This bound was introduced by Kim and Hong in [14] and it is based on the following simple observation. If a container is blocking, then it must be relocated at least once. Thus we can obtain a lower bound on the number of relocations by counting the number of blocking containers in $B$. We denote this lower bound by $S_0(B)$ and the corresponding cumulative lower bound by $L_0$. Note that if
Algorithm 1 $A^*$ Algorithm

1: procedure $[z_{A^*}, \gamma] = A^*(B, R, S, N)$
2: Pre-processing:
3: while target container $n$ is on top do
4:     retrieve target container $n$ of $B$ and $n \leftarrow n + 1$
5: end while
6: Initialize:
7: $z_{A^*} \leftarrow \infty$, $l \leftarrow 0$, $B^l \leftarrow B$, $m \leftarrow 0$ (number of nodes)
8: Loop:
9: while all nodes at level $l$ are not pruned do
10: for all nodes $B^l$ at level $l$ do
11:     $U(B^l) = R(B^l) + l$ and $L(B^l) = S(B^l) + l$
12: Updating the incumbent:
13: if $U(B^l) < z_{A^*}$ then
14:     $z_{A^*} \leftarrow U(B^l)$
15: end if
16: Pruning:
17: if $U(B^l) > L(B^l)$ and $L(B^l) < z_{A^*}$ then
18:     for Every 'Child' of $B^l$ do
19:         if $m \geq N$ then
20:             Stop
21:         else
22:             add 'Child' to the tree at level $l + 1$ and $m = m + 1$
23:         end if
24:     end for
25: end if
26: end for
27: $l \leftarrow l + 1$
28: end while
29: Gap:
30: $L_{\min} = \min_{D \text{ leaf not pruned}} (L(D))$
31: $\gamma \leftarrow z_{A^*} - L_{\min}$
32: end procedure

Figure 2-8: The $A^*$ algorithm

A container blocks more than one container (like container 6 that blocks 1 and 4 in Figure 2-9), we count this container only once. For the bay in Figure 2-9, $S_0(B) = 2$ (counting containers 5 and 6).

The counting lower bound has the following properties:
(i) For any configuration \( B \), we have \( S_0(B) \leq z_{opt}(B) \).

(ii) For any configuration \( B \), any level \( l \geq 0 \), any configuration \( B^l \) in level \( l \) of \( A^* \) and any child \( B^{l+1} \) of \( B^l \), we have \( L_0(B^{l+1}) \geq L_0(B^l) \).

**Look-ahead lower bound:** The counting lower bound counts the unavoidable relocations. The underlying idea of the look-ahead lower bound is to count the additional relocations that may be necessary as a result of one of the unavoidable relocations. If one of the unavoidable relocations is a *bad* move (i.e., if the relocated container blocks one or more container in its new column), a future relocation for the relocated will be necessary. For example, in the bay shown in Figure 2-9, any relocation for container 6 would be a "bad" move hence it has to be relocated at least twice. Similarly, after relocating container 6 and retrieving container 1, any relocation for container 5 would be a "bad" move; so it has to be relocated at least twice. Therefore, the look-ahead lower bound is \( S_0(B) + 2 = 2 + 2 = 4 \).

Zhu et al ([30]) used this idea to define the following family of lower bounds that provides tighter bounds by taking into account future relocations. For the sake of completeness, we redefine the bounds formally ([3]):

Let \( n_1 \) be the smallest container in the bay (without loss of generality we assume that \( n_1 = 1 \)). Let \( k \) be a container \( (k \in \{n_1, \ldots, N\}) \) and let \( R_k(B) \) be the set of containers blocking \( k \) and not blocking any container \( k' \) such that \( k' < k \).

Let \( B_{n_1} = B \) and let \( B_{n_1+1} \) be the bay where container \( n_1 \) and containers in \( R_{n_1}(B) \) have all been discarded from \( B_{n_1} \). Also, let \( MOM(B) \) be the maximum of minimums of all columns of any bay \( B \) (for example for the bay in Figure, 2-9 \( MOM(B) = \max\{1, 2, 3\} = 3 \)). By recursion, we can define a sequence of bays \( B_k \) for \( k \in \{n_1 + 1, \ldots, N\} \).

For \( n \in \mathbb{N} \), we now define the \( n^{th} \) look-ahead lower bound (denoted by \( S_n(B) \)):

\[
S_n(B) = S_0(B) + \sum_{k=n_1}^{\min(n+n_1-1,N)} \sum_{r \in R_k(B)} \chi(r > MOM(B_k)),
\] (2.32)
where $\chi$ is the indicator function. The corresponding cumulative lower bound is denoted by $L_n(B)$.

The look-ahead lower bound has the following properties (the proofs are given in [3]):

(i) For any configuration $B$ and $n \in 1, \ldots, N$, we have $S_n(B) \leq z_{opt}(B)$,

(ii) For any configuration $B$ and $n \in 1, \ldots, N$, we have $L_n(B) \geq L_{n-1}(B)^4$,

(iii) For any $n \in 1, \ldots, N$, configuration $B$, level $l \geq 0$, configuration $B^l$ in level $l$ of $A^*$ and any child $B^{l+1}$ of $B^l$, we have $L_n(B^{l+1}) \geq L_n(B^l)$.

The numerical experiments on medium-sized instances in [3] demonstrate that the average number of nodes needed (for obtaining the optimal solution) when using $S_N$ is $1/3$ of the number of nodes when using $S_0$. Also, using $S_1$ and $S_2$ instead of $S_0$, decreases the average number of nodes by $1/3$ and $1/2$, respectively. The authors of [3] also show that $S_N$ on the optimal path converges to the optimal solution very quickly.

### 2.5.2 Upper Bound

Any feasible solution can be used as an upper bound. So we can use the existing heuristics to construct feasible solutions that serve as upper bounds. Ideally we would want a heuristic that is close to the optimal solution and also easy to compute. (Note that we need to compute the upper bound at every node). We use the heuristic proposed by Caserta et al. ([7]) (which we refer to as heuristic $H$), because it is easy to compute and on average performs better than other heuristics. For the sake of completeness, we describe the $H$ heuristic here. Moreover, we present several properties of this heuristic. The proofs for these properties can be found in [3].

---

4Notice that $S_N(B) = S_{N-C}(B)$ because when we reach a bay with $N - C$ containers, there is always at least one empty column. Also note that if $k$ is the smallest integer such that $B_k$ has an empty column, the lower bound does not change, i.e., $S_N(B) = S_{k-1}(B)$. Thus in practice, the lower bounds are computed up to the point that an empty column is created in the bay as a result of retrievals.

5In Section 2.6, we present the numerical results of comparing different heuristics and a new heuristic proposed in this thesis.
Heuristic H: Suppose \( n \) is the target container located in column \( c \), and \( r \) is the topmost blocking container in \( c \). We use the following rule to determine \( c^* \), the column where \( r \) should be relocated to.

\[
c^* = \begin{cases} 
\arg\min \{ \min(c_i) : \min(c_i) > r \} & \text{if } \exists c_i \text{ such that } \min(c_i) > r; \\
\arg\max \{ \min(c_i) \} & \text{otherwise.}
\end{cases}
\]

Where \( \min(c_i) \) is the minimum of column \( c_i \). We set \( \min(c_i) = N + 1 \) if \( c_i \) is empty.

If there is at least one column where \( \min(c_i) \) is greater than \( r \) (\( r \) can only do "good" moves), then we relocate \( r \) to such a column where \( \min(c_i) \) is minimized, since columns with larger minimums can be useful for larger blocking containers. If there is no column satisfying \( \min(c_i) > r \) (\( r \) can only do "bad" moves), then we choose the column where \( \min(c_i) \) is maximized in order to delay the next unavoidable relocation of \( r \) as much as possible. We will refer to this heuristic as heuristic \( H \) and denote its number of relocations by \( z_H(B) \).

Heuristics \( H \) has the following properties:

(i) For any configuration \( B \), we have \( z_{opt}(B) \leq z_H(B) \),

(ii) For any bay with \( C \) columns and at most \( C \) containers, and with any configuration \( B \), we have \( S_0(B) = z_{opt}(B) = z_H(B) \),

(iii) For any bay with \( C \) columns and at most \( C + 1 \) containers, and with any configuration \( B \), we have \( S_1(B) = z_{opt}(B) = z_H(B) \),

(iv) For any bay with \( C \) columns, for any configuration \( B \) with at most:

- \( C+2 \) containers, we have \( z_H(B) \leq z_{opt} + 2 \),
- \( C+k \) containers where \( k \geq 3 \), we have \( z_H(B) \leq z_{opt}(B) + \frac{k(k+1)}{2} \).

Property (iii) implies that if in the \( A^* \) algorithm we use heuristic \( H \) and \( S_1 \) to compute upper and lower bounds respectively, the algorithm will terminate and return
the optimal solution after at most $N - C - 1$ retrievals. A similar result with different guarantees on the optimality gap is implied from property (iv).

Notice that although most instances can be solved with the $A^*$ algorithm within a reasonable time, there are some "hard-to-solve" instances which require building a tree with a large number of nodes. Nevertheless, the $A^*$ method is tunable. We can set a maximum number of nodes ($\mathcal{N}$) to explore and stop the algorithm when the tree reaches this number of nodes. If it reaches $\mathcal{N}$ nodes, we can compute an upper bound on the gap between the best solution found by the tree and the optimal solution. The authors of [31] study the impact of $\mathcal{N}$ on the average guaranteed gap with optimality, and the percentage of instances solved optimally, for medium-sized instances. They show that the average guaranteed gap at the root node is very small (less than 0.4), and in more than 70% of instances, the optimal solution is found at the root node. Their experiments demonstrate the trade-off between the quality of the solution (represented by the average guaranteed gap) and its tractability (represented by $\mathcal{N}$). From the practical point of view, one can set $\mathcal{N}$ according to the desirable average gap.

We conclude this section with an example from [3] that illustrates the $A^*$ decision tree for a small bay.

![Figure 2-9: Initial configuration of the bay for Example 2.5.1](image-url)

**Example 2.5.1. $A^*$ decision tree** Figure 2-10 presents an example of $A^*$ where the initial bay is the one in Figure 2-9. It uses heuristic $H$ to obtain an upper bound and $S_0$ to obtain a lower bound. Let us examine the 4 rightmost configurations (at level 2) in Figure 2-10. The third node at this level uses rule (i) to stop branching from this node since lower and upper bounds are both equal to 4 at this node. Notice that $z_A = 4$, so we can also prune the first, second and fourth nodes using rule (ii).
Therefore, the tree is complete at level 2. We follow the path until the third node at level 2 and then follow the feasible solution given by the upper bound from this point.

2.6 Heuristic Approach

In this section, we introduce a new class of heuristics, referred to as Tree Heuristic (TH-L), for the CRP with complete information. Moreover, we compare 3 existing heuristics ([14], [7], [21]) and our new heuristic with the A* algorithm. Our experimental results are that the tree heuristic outperforms the existing heuristics and on average has a very small gap with the optimal solution.

2.6.1 Tree Heuristic

We introduce a new class of heuristics that uses the idea of “branching” to improve any existing heuristic; we refer to this class as Tree Heuristic (TH-L). The basic
idea of TH-L is to take the L best candidate columns for each relocation move (i.e., L columns with the best score computed by a heuristic) and branch on them to construct a decision tree. Once the entire tree is constructed, the path with the minimum number of relocations will determine the best sequence of moves obtained by TH-L. The L best columns can be chosen using any of the existing heuristics that compute a score for each column. Using this principle and considering several good candidates for each relocation, it is less likely to make a mistake. Here, we implement the TH-L with the H heuristic and present the algorithm in Figure 2-11. In our experiment, we set \( L = 2 \). Note that TH results in fewer or the same number of relocations compared to H since the path of H is included in the tree of TH-L.

Algorithm 2 Tree Heuristic

1: procedure \([Z_{TH}] = TreeHeuristic(B, L)\)
2: \(Z_{TH} \leftarrow 0\)
3: while B is not empty do
4: Retrieval:
5: 
6: if target container n is on top of its column then
7: 
8: Retrieval n form B
9: Relocation:
10: 
11: \( r \leftarrow \) topmost blocking container
12: \( C_1 \leftarrow \) sort arg increasingly \{\min(c_i) \mid \min(c_i) > r\}
13: \( C_2 \leftarrow \) sort arg decreasingly \{\min(c_i) \mid \min(c_i) < r\}
14: \( C_3 \leftarrow \langle C_1, C_2 \rangle \)
15: \( S \leftarrow C_3[1 : L]\)
16: \( Z_{TH} = Z_{TH} + \min_{s \in S}\{TreeHeuristic (B where r moves to column s,L)\}\)
17: end if
18: end while
19: end procedure

Figure 2-11: The tree heuristic
2.6.2 Numerical Experiments

Several heuristics have been developed for the CRP. Heuristics are particularly useful in real-time decision making in container terminals because they are fast and easy to implement. Here we study the performance of the tree heuristic [3], heuristic H, heuristic KH [14] (which uses an estimate of future relocations), and heuristic LA-5 [21] (which takes into account the possibility of repositioning moves).

For the numerical experiment, we randomly generate 100,000 instances with 4 tiers, 7 columns, and 21 containers (with 3 containers in each column) \(^6\). We then solve each of the instances five times (using 3 existing heuristics, our TH-L heuristic, and using the \(A^*\) algorithm to obtain the optimal solution).

In order to compare the performance of different heuristics, we measure the following two metrics over the 100,000 instances:

- The distribution of \(z_{\text{heuristic}} - z_{\text{opt}}\);
- The expected performance ratio of heuristic defined as

\[
PR(heuristic) = \frac{z_{\text{heuristic}} - z_{\text{opt}}}{z_{\text{opt}}},
\]

where \(z_{\text{heuristic}}\) is the number of relocations in the heuristic solution, and \(z_{\text{opt}}\) is the optimal solution obtained by the \(A^*\) algorithm. PR shows the relative optimality gap for a given heuristic.

Results for the four heuristics are summarized in Table 2.6. First notice that H is optimal in most instances (87%). This is one of the main reasons that heuristic H was chosen as an upper bound in the \(A^*\) algorithm. Secondly, TH-2 outperforms the three other heuristics, H, KH and LA-5, in distribution and in terms of the average performance ratio.

Finally, notice that the figures in the distribution column for LA-5 do not add up to 100% because there are few cases where LA-5 is better than the \(A^*\), as a result

\(^6\)The configuration of each bay has uniform probability, meaning that each container is equally likely to be anywhere in the configuration, with the restriction that there are 3 containers per column.
of taking into account repositioning moves; for 0.5% of the 100,000 instances, LA-5 finds a solution with fewer relocations than $A^\ast$. Recall that we implement the $A^\ast$ algorithm for the restricted CRP.

<table>
<thead>
<tr>
<th>Gap with Optimal</th>
<th>H</th>
<th>KH</th>
<th>LA-5</th>
<th>TH-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>87.0%</td>
<td>31.8%</td>
<td>83.5%</td>
<td>95.7%</td>
</tr>
<tr>
<td>1</td>
<td>11.4%</td>
<td>21.4%</td>
<td>13.7%</td>
<td>3.98%</td>
</tr>
<tr>
<td>2</td>
<td>1.4%</td>
<td>21.3%</td>
<td>2.0%</td>
<td>0.3%</td>
</tr>
<tr>
<td>≥ 3</td>
<td>0.2%</td>
<td>11.4%</td>
<td>0.3%</td>
<td>0.02%</td>
</tr>
<tr>
<td>E[PR]</td>
<td>1.44%</td>
<td>16.0%</td>
<td>1.81%</td>
<td>0.44%</td>
</tr>
</tbody>
</table>

Table 2.6: Benchmarks of heuristics on 100,000 instances

As the last experiment in this section, we study how parameter L affects the performance of the TH-L heuristic. We consider the same 100,000 instances and solve them with L varying from 1 to 6. Notice that TH with L=6 considers all possible columns for a relocation (since each bay has 7 columns); hence it gives the same solution as the $A^\ast$ and $PR(TH-6) = 0$. Also, $L = 1$ gives the same solution as heuristic H.

<table>
<thead>
<tr>
<th>L</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR (TH-L)</td>
<td>1.44%</td>
<td>0.44%</td>
<td>0.27%</td>
<td>0.20%</td>
<td>0.16%</td>
<td>0%</td>
</tr>
</tbody>
</table>

Table 2.7: Effect of parameter L on the performance of heuristic TH-L

The main observation is that the marginal gain of branching is maximum when we increase L from 1 to 2. By considering two promising columns instead of one, TH-2 finds better solutions for most instances where H was not optimal. Note that increasing L (considering more candidate columns) will further improve the solution; however the gain from more branching is small considering the exponentially increasing cost of computation, which is common in these types of combinatorial optimization problems.
2.7 Concluding Remarks

In this chapter, we studied the CRP with complete information. First, we illustrated how incorporating container arrival times and departure times in the model allows for optimizing the truck wait times together with the number of relocation moves, and for planning repositioning moves. We developed a time-based mathematical model and presented an IP formulation of this model, and used it to study the effect of repositioning moves during off-peak hours on the average truck wait times. Our experiments showed that repositioning moves during off-peak hours can significantly decrease the congestion in the port and the average wait times of external trucks. This suggests that port operators should manage the appointment systems such that there is a non-uniform traffic of external trucks (with peak and off-peak hours).

We further extended our formulation to the dynamic setting where containers are stacked and retrieved continuously. Assuming that there is more flexibility in stacking order (compared to the retrieval order of containers), we illustrated that one can exploit such flexibility to further decrease the number of relocations by jointly optimizing the stacking slot and stacking order.

We also proposed a flexible retrieval policy that allows for out-of-order retrievals, and used our IP formulation to study the effect of this policy on the number of relocations and on truck wait times. We demonstrated that such a policy can significantly decrease the number of relocations and truck wait times; and hence can be an efficient policy especially during the peak hours in the port.

Moreover, we revisited a fast and efficient method, the $A^*$ algorithm, that provides optimal solutions for the CRP, and we discussed several properties of this method as applied to the CRP. Lastly, we proposed a heuristic for the CRP and assessed its performance and computational tractability through experiments and benchmarked it against the $A^*$ algorithm.
Chapter 3

CRP with Incomplete Information

In this chapter, we describe the setting of the container relocation problem with incomplete information, i.e. the case where no or partial information about the departure time of the containers is available in advance. In Section 3.2 we present a 2-stage approximate stochastic optimization algorithm, called ASA*, for the CRP with incomplete information. We also give theoretical bounds on the approximation error and present results of numerical experiments on mid-sized bays.

In Section 3.3, we introduce three myopic heuristics and show through numerical experiments that they are fast and perform well on average compared to the ASA* algorithm. Lastly, in Section 3.4, we use ASA* and one of the myopic heuristics to assess the value of information, that is the effect of different levels of information on the average number of relocations.

3.1 Problem Setting

The CRP with incomplete information involves retrieving $N$ containers from a bay with $C$ columns and $P$ tiers, where partial information about the departure order of the containers is initially available. Usually, very little information is available about the containers that are going to be retrieved far in the future. Thus, it is reasonable to assume that at any given time step $t$, we only know the departure order of a certain number of containers in the bay (the containers that are going to be retrieved
within a short time horizon after \( t \). We refer to such containers in the bay as *known* containers. Similarly, we refer to the remaining containers in the bay as *unknown* containers. Note that all unknown containers have larger index (i.e., later departure time) than known containers.

As time passes, some of the known containers are retrieved; and, more information becomes available, causing some of the unknown containers to become known. In general, the information might be updated multiple times during the retrieval process. For example, every time that the arrival time of a truck is provided by the truck driver, the information gets updated and some unknown containers become known. In the most general case, the information can be updated every time step. Alternatively, we can consolidate several small pieces of information into one or a few pieces, and assume that the information is revealed at \( \Gamma \) different times during the retrieval process (i.e., \( \Gamma \) is the number of times that information is updated). In such a case, we have a multi-stage problem with \( \Gamma \) sets of decisions to make.

Here we focus on a 2-stage setting; we assume that a subset of containers is initially known and that the departure order of all the remaining ones becomes known at time \( t^* \). We denote the set of known containers by \( K \) and assume that containers \( \{1, 2, \ldots, |K|\} \) are those known at time zero. Similarly, the set of unknown containers is denoted by \( U \), and containers \( \{|K|, |K|+1, \ldots, N\} \) become known at time step \( t^* > 0 \) (recall that a time step is the minimum amount of time required for completing one retrieval or one relocation move). We refer to this setting as a 2-stage CRP since there are two types of decisions that need to be made: first-stage decisions (retrievals and relocations before time step \( t^* \)) and second-stage decisions (retrievals and relocations after time step \( t^* \)). We assume that before \( t^* \), we have probabilistic information about the containers in \( U \), meaning that we know the probability of realization of each possible departure order. Such information can be obtained from historical data or from an appointment system that provides some estimate of departure times of the containers. This can be in the form of a time window for retrieving each container. We denote the set of possible departure orders of the containers in \( U \) (possible scenarios) by \( Q \). From now on, we assume that all scenarios are equally likely, i.e., probability of
each scenario is \( \frac{1}{|Q|} \), and the number of scenarios is \((N - |K|)!\). Note, however, that we could use the algorithm to solve the CRP with any other probability distribution on the departure order of unknown containers.

We use a 2-stage stochastic optimization technique to solve this problem, where in the first-stage we minimize \( E[z] \) as follows:

\[
\min_{\sigma_1, \ldots, \sigma_{t^* - 1}} E[z] = \sum_{q \in Q} \frac{1}{|Q|} z(B(q)),
\]

where \( B(q) \) is the resulting bay when scenario \( q \) is realized, \( z(B(q)) \) is the total number of relocations for \( B(q) \), and \( \sigma_1, \ldots, \sigma_{t^* - 1} \) are the first-stage decisions. The algorithm is explained in the next section.

### 3.2 The ASA* Algorithm

Recall that the \( A^* \) algorithm, as explained in Section 2.5, relies on the assumption that the departure order of containers is known in advance. Here we explain how the \( A^* \) algorithm can be adapted in a Dynamic Programming (DP) framework for solving the CRP with incomplete information. The 2-stage problem can be solved with the \( A^* \) algorithm in 4 steps as follows:

1. We build the tree with \( t^* - 1 \) moves for time-steps 1, 2, \ldots, \( t^* - 1 \), in a similar way as illustrated in Figure 2-10; we denote this tree by \( T_{[1, t^* - 1]} \).
2. For each node at level \( t^* - 1 \), we compute the expected number of remaining relocations. Theoretically, we enumerate all possible scenarios and solve the CRP with complete information corresponding to each scenario, using the \( A^* \) algorithm.
3. We find \( p^* \) (the optimal path or sequence of moves) that minimizes the expected total number of relocations over all path (up to and including time \( t^* - 1 \)).
4. Once \( p^* \) is selected up to time-step \( t^* - 1 \), and we observe the information at \( t^* \), we use the \( A^* \) algorithm to solve a specific instance through the end.

Notice that to find the optimal path for time interval \([1, t^* - 1]\), we need to solve up to \((c-1)(t^*-1)(N-|K|)!\) instances in Step (2) with the \( A^* \) algorithm. Although \( A^* \)
is fast, the number of scenarios is prohibitively large and enumerating all scenarios is not feasible due to limited resources of memory and long computation time. We next explain how we use sampling and pruning to overcome these issues. We also quantify the error incurred as a result of sampling and pruning. We refer to the resulting algorithm as ASA* (Approximate Stochastic A*).

**Limiting the number of possible scenarios on each path.** We overcome this issue by sampling from the set of possible scenarios on each path and computing the number of relocations for the sampled departure orders rather than for all possible orders. For each path \( p \), let \( \bar{z} \) be the expected relocations from the samples, that is the sampling average. Also let \( E[z] \) be the true mean. To insure a large enough sample to get a good approximation, we use the following version of Hoeffding’s inequality:

\[
P(|E[z] - \bar{z}| > \delta) \leq 2 \exp \left( \frac{-2\delta^2}{(r_{\text{max}} - r_{\text{min}})^2} \right),
\]

where \( \delta \) is a pre-specified level of desired precision. The random variable of interest (which is the number of relocations), is bounded by \( r_{\text{max}} \) and \( r_{\text{min}} \). Note that \( r_{\text{min}} = 0 \); for \( r_{\text{max}} \), we do not have a tight bound but we can use \( N(P - 1) \) as an upper bound (since each of the \( N \) containers is blocked by at most \( P - 1 \) containers). Let us denote the desired probability for bounding the error (i.e., RHS of (3.2)) by \( \epsilon \). For a given \( \delta \) and \( \epsilon \), the required number of samples can be computed as follows.

\[
S \geq \frac{r_{\text{max}}^2 \ln \frac{\epsilon}{2}}{-2\delta^2}.
\]

By sampling from the possible scenarios on each path, we can significantly reduce the number of scenarios (and thus the computation time). For example, for a bay with 7 columns, 4 tiers, 21 containers, and \( |\mathbf{K}| = 6 \), the total number of possible scenarios on any of the paths at time \( t^* \) is about \( 10^{11} \). Using inequality (3.3), the total number of scenarios would be around 30000 for \( \delta = 0.5, \epsilon = 0.05 \), and \( r_{\text{max}} = 63 \). Note that by sampling, we incur an error and ASA* may choose a suboptimal path, \( \hat{p} \) where \( E[z_{\hat{p}}] > E[z_{p^*}] \) (recall that \( p^* \) is the optimal path that would be chosen.
without sampling). In the next proposition, we show that such an error, denoted by \( e_1 \triangleq \mathbb{E}[z_p] - \mathbb{E}[z_{p^*}] \), is bounded.

**Proposition 3.2.1.** Suppose for each path, we estimate the number of relocations using \( S \) independent samples, where \( S \) is given in (3.3). Also, suppose \( ASA^* \) chooses path \( \tilde{p} \). Let \( e_1 \triangleq \mathbb{E}[z_p] - \mathbb{E}[z_{p^*}] \). Then we have \( \mathbb{E}[e_1] \leq 2\delta \sqrt{\frac{\pi}{-\ln(\epsilon/2)}} \).

**Proof.** We can write \( \mathbb{E}[e_1] \) as follows:

\[
\mathbb{E}[e_1] = (\mathbb{E}[\mathbb{E}[z_p] - \mathbb{E}[p]]) - (\mathbb{E}[\mathbb{E}[z_{p^*}] - \mathbb{E}[z_{p^*}]] + \mathbb{E}[\mathbb{E}[z_{p^*}] - \mathbb{E}[z_{p^*}]]
\]

(3.4)

\[
\leq \mathbb{E}[[\mathbb{E}[z_p] - \mathbb{E}[p]] + \mathbb{E}[[\mathbb{E}[z_{p^*}] - \mathbb{E}[z_{p^*}]]].
\]

The above inequality holds because \( ASA^* \) chose \( \tilde{p} \), and thus we have \( \tilde{z}_p < \tilde{z}_{p^*} \). Moreover, we can compute \( \mathbb{E}[[\mathbb{E}[z_p] - \mathbb{E}[p]] \) and \( \mathbb{E}[[\mathbb{E}[z_{p^*}] - \mathbb{E}[z_{p^*}]] \) using their CDF. We denote \( [\mathbb{E}[z_p] - \mathbb{E}[p]] \) by \( \Delta \). Note that \( 0 \leq \Delta \leq r_{\text{max}} \) and inequality (3.2) gives a bound on its CDF, when estimating \( \tilde{z}_p \) with \( S \) samples. For \( \mathbb{E}[\Delta] \), we have:

\[
\mathbb{E}[\Delta] = \int_0^{r_{\text{max}}} 1 - F_\Delta(x)dx
\]

(3.5)

\[
\leq \int_0^{r_{\text{max}}} 2 \exp\left(\frac{-2Sx^2}{r_{\text{max}}^2}\right)dx
\]

(3.6)

\[
= 2 \int_0^{r_{\text{max}}} \left(\frac{\epsilon}{2}\right)z^2 dx \leq 2 \int_0^{\infty} \left(\frac{\epsilon}{2}\right)z^2 dx = \delta \sqrt{\frac{\pi}{-\ln(\epsilon/2)}}.
\]

(3.7)

We can compute \( \mathbb{E}[[\mathbb{E}[z_{p^*}] - \mathbb{E}[z_{p^*}]] \) in a similar way. The first equality in (3.7) is obtained by substituting \( S \) with the expression in (3.3). Using the bound on \( \mathbb{E}[\Delta] \) and inequality (3.5), we can compute \( \mathbb{E}[e_1] \) as follows:

\[
\mathbb{E}[e_1] \leq 2\delta \sqrt{\frac{\pi}{-\ln(\epsilon/2)}}.
\]

(3.8)

**Pruning the paths of** \( T_{[1,t^*-1]} \). To address this issue, we use the upper bound and lower bounds to prune the nodes of \( T_{[1,t^*-1]} \), similar to the \( A^* \) algorithm. However,
since some of the containers are unknown before $t^*$, we have to compute the expectations ($\mathbb{E}[L]$ and $\mathbb{E}[U]$). Again, we use the idea of sampling and we estimate these values by computing $\bar{L}$ and $\bar{U}$ using $S$ samples, where $S$ is obtained from Inequality (3.3). Because of the sampling error, we may prune an optimal path by mistake. In the next two propositions, we show that such an error is bounded, whether we prune some paths once at $t^*-1$ (resulting in error $e_2$) or prune some paths at several time-steps before $t^*-1$ (resulting in error $e_3$).

**Proposition 3.2.2.** Suppose that for each path $p$ at time-step $t^*-1$, we estimate the expected lower and upper bounds ($\bar{L}_p$ and $\bar{U}_p$) from $S$ samples where $S$ is given in (3.3). Also, suppose ASA* chooses to prune one or more paths (each denoted by $\tilde{p}$). We have $\mathbb{E}[e_2] \leq 2\delta \sqrt{\frac{\pi}{-\ln(\varepsilon^2)}}$.

**Proof.** Let $\tilde{p}$ be the path with minimum estimated upper bound. Notice that pruning $\tilde{p}$ would be a mistake if the true mean of $L_{\tilde{p}}$ is less than the true mean of $U_{\tilde{p}}$, i.e., $\mathbb{E}[L_{\tilde{p}}] < \mathbb{E}[U_{\tilde{p}}]$; in such a case, $e_2 \leq \mathbb{E}[U_{\tilde{p}}] - \mathbb{E}[L_{\tilde{p}}]$. Otherwise $e_2$ is zero; thus we have $e_2 \leq (\mathbb{E}[U_{\tilde{p}}] - \mathbb{E}[L_{\tilde{p}}])^+$. Let us denote $\bar{L}_{\tilde{p}} - \mathbb{E}[L_{\tilde{p}}]$ and $\mathbb{E}[U_{\tilde{p}}] - \bar{U}_{\tilde{p}}$ by $x_L$ and $x_U$, respectively. Also, let $d$ be $\bar{L}_{\tilde{p}} - \bar{U}_{\tilde{p}}$. The true and estimated values of the upper bound and lower bound, and the loss are illustrated in Figure 3-1; $e_2$ is shown by the thick line segment. We can express $e_2$ as follows:

$$e_2 \leq ((\mathbb{E}[U_{\tilde{p}}] - \bar{U}_{\tilde{p}}) + (\bar{L}_{\tilde{p}} - \bar{L}_{\tilde{p}}) + (\bar{L}_{\tilde{p}} - \mathbb{E}[L_{\tilde{p}}]))^+ = (x_U - d + x_L)^+. \quad (3.9)$$

The expected loss, $\mathbb{E}[e_2]$, can be bounded as shown in (3.12).

$$\mathbb{E}[e_2] = \mathbb{E}[(x_L + x_U - d)^+] \leq \mathbb{E}[(x_L + x_U)^+] \leq \mathbb{E}[(x_L + x_U)] \leq \mathbb{E}[(|x_L| + |x_U|)] = \mathbb{E}[|x_L|] + \mathbb{E}[|x_U|] \leq 2\delta \frac{\pi}{\sqrt{-\ln(\varepsilon^2)}}. \quad (3.10)$$

The first inequality in (3.10) holds because $d$ is always positive. Inequality (3.11) results from the triangular inequality, and the last inequality is obtained by replacing...
\( \mathbb{E}[|x_L|] \) and \( \mathbb{E}[|x_U|] \) by the expression in (3.7). \( \square \)

**Proposition 3.2.3.** Suppose we prune some paths at \( m \) time-steps \( t_1, t_2, \ldots, t_m < t^* - 1 \). At each time of pruning and for each path \( p \), we estimate the expected lower and upper bounds (\( \bar{L}_p \) and \( \bar{U}_p \)) from \( S \) samples where \( S \) is given in (3.3). Also, suppose that at each time of pruning, \( ASA^* \) chooses to prune one or more paths (each denoted by \( \hat{p} \)). The expected total loss, \( \mathbb{E}[e_3] \), is bounded by \( m \left( \frac{\epsilon}{2} \frac{\delta^{d_{min}^2}}{2} + \frac{d_{min}}{\delta} \sqrt{-\ln\left(\frac{\epsilon}{2}\right)} \frac{\pi}{2} \left( \frac{\epsilon}{2} \right)^{\frac{3}{2}} + \frac{U_p^{max}}{U_p} \right) \) where \( U_p^{max} = \max\{U_{\hat{p}_1}, \ldots, U_{\hat{p}_m}\} \), and \( d_{min} = \min\{d_1, \ldots, d_m\} \).

**Proof.** Let \( \hat{p}_{max} \) be the path that maximizes \( \mathbb{E}[U_{\hat{p}}] \) among \( \hat{p}_1, \ldots, \hat{p}_m \). We bound \( e_3 \) as follows:

\[
e_3 \leq P(\text{mistake at } t_1 \cup \cdots \cup \text{mistake at } t_m) (\mathbb{E}[U_{\hat{p}_{max}}] - \mathbb{E}[z_{p*}]) \tag{3.13}\]
\[
\leq m P(\text{mistake at } t_1) (\mathbb{E}[U_{\hat{p}_{max}}] - 0) \tag{3.14}\]
\[
= m P(\text{mistake at } t_1) (\mathbb{E}[U_{\hat{p}_{max}}] - U_{\hat{p}_{max}} + U_{\hat{p}_{max}}) \tag{3.15}\]
\[
\leq m P(\text{mistake at } t_1) (|x_U| + U_{\hat{p}}^{max}). \tag{3.16}\]
Notice that inequality 3.14 is obtained by using the union bound. Moreover, we replace the probability of mistakes at each step by the maximum probability. Without loss of generality, we assume that maximum probability is at $t_1$. Also note that we do not know which path is $\hat{p}_{\text{max}}$ because we have not observed $E[U_{p_1}], \ldots, E[U_{p_m}]$. Nevertheless, regardless of the path, we can replace $E[U_{\hat{p}_{\text{max}}}] - \overline{U}_{\hat{p}_{\text{max}}}$ with $|x_U|$ (by definition) to obtain (3.16). Also, we can bound $\overline{U}_{\hat{p}_{\text{max}}}$ by $\overline{U}_{\hat{p}}^{\text{max}}$. Now we compute $P($mistake at $t_1$). Let us denote $|x_L|$ and $|x_U|$ by $x$ and $y$, respectively. Also, let $w = x + y$ and $f_W(w)$ be the PDF of $w$. Notice that the probability of making a mistake depends on value of $d$; the maximum probability corresponds to the stage with smallest value of $d$ and can be computed as follows:

$$P(\text{mistake at } t_1) = P(x + y - d_{\text{min}} > 0)$$

$$= \int_{d_{\text{min}}}^{\infty} f_W(w)dw$$

$$= \int_{d_{\text{min}}}^{\infty} \int_0^w f_X(w-y)f_Y(y)dydw.$$  (3.17)

$f_X(x)$ and $f_Y(y)$ can be obtained from (3.2), and the above integral can be bounded as follows:

$$\int_{d_{\text{min}}}^{\infty} \int_0^w f_X(w-y)f_Y(y) = \int_{d_{\text{min}}}^{\infty} \int_0^w 4\left(-\frac{\ln(\frac{\epsilon}{2})}{\delta^2}\right)^2 y(w-y)\left(\frac{\epsilon}{2}\right)^2 \frac{(w-y)^2}{2\delta^2}$$

$$= \left(\frac{\epsilon}{2}\right)^2 \frac{d_{\text{min}}^{\text{max}}}{\delta^2} + \frac{d_{\text{min}}^{\text{max}}}{\delta} \sqrt{\frac{-\ln(\frac{\epsilon}{2})\pi}{2}} \left(\frac{\epsilon}{2}\right)^2 \frac{d_{\text{min}}^{\text{max}}}{\delta^2}\text{Erf}\left(\frac{d_{\text{min}}\sqrt{-\ln(\frac{\epsilon}{2})}}{\sqrt{2}\delta}\right)$$

$$\leq \left(\frac{\epsilon}{2}\right)^2 \frac{d_{\text{min}}^{\text{max}}}{\delta^2} + \frac{d_{\text{min}}^{\text{max}}}{\delta} \sqrt{\frac{-\ln(\frac{\epsilon}{2})\pi}{2}} \left(\frac{\epsilon}{2}\right)^2 \frac{d_{\text{min}}^{\text{max}}}{\delta^2}. $$  (3.18)

From 3.16, 3.18, and expression 3.7 for $E[|x_U|]$, it follows that:

$$E[e_3] \leq m \left(\left(\frac{\epsilon}{2}\right)^2 \frac{d_{\text{min}}^{\text{max}}}{\delta^2} + \frac{d_{\text{min}}^{\text{max}}}{\delta} \sqrt{\frac{-\ln(\frac{\epsilon}{2})\pi}{2}} \left(\frac{\epsilon}{2}\right)^2 \frac{d_{\text{min}}^{\text{max}}}{\delta^2}\right) \left(\sqrt{\frac{\pi}{2}} - \overline{U}_{\hat{p}}^{\text{max}}\right).$$  (3.19)

where $\overline{U}_{\hat{p}}^{\text{max}} = \max\{\overline{U}_{p_1}, \ldots, \overline{U}_{p_m}\}$, and $d_{\text{min}} = \min\{d_1, \ldots, d_m\}$.  

□
In Propositions 3.2.1-3.2.3, we bound the loss that can be incurred by pruning some paths at time $t^* - 1$ ($e_1$ and $e_2$) or at $t < t^* - 1$ ($e_3$). Notice that $e_1$ and $e_2$ are independent of the bay size; thus if we prune paths only at $t^* - 1$, the loss remains unchanged and the relative loss (as a percentage of total relocations) decreases as the bay gets larger. Therefore, for large bays and “hard-to-solve” configurations, one can pick larger $\delta$ and $\epsilon$ that result in a smaller number of samples, and the relative error would still be small.

Table 3.1 shows $e_1$, $e_2$ and $e_3$. The losses are very small even for combinations of $\delta$ and $\epsilon$ that result in a reasonably small number of samples. For example, for $\delta \leq 1$ and $\epsilon < 0.1$, $\mathbb{E}[e_1]$ and $\mathbb{E}[e_2]$ are no more than 2. Notice that $e_1$, $e_2$ are independent of the bay size; so the relative error ($\frac{e_1 + e_2}{\mathbb{E}[z]})$ decreases as the bay gets larger ($\mathbb{E}[z]$ is the average number of relocations when full information is available). For the $e_3$, this measure is almost constant for different bay sizes and can be controlled by changing $\epsilon$, $\delta$, and $m$ (the number of times that we do pruning at $t < t^* - 1$).

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\epsilon$</th>
<th>$\mathbb{E}[e_1]$ and $\mathbb{E}[e_2]$</th>
<th>$C$</th>
<th>$\mathbb{E}[e_3]$</th>
<th>% error($\frac{\mathbb{E}[e_3]}{\mathbb{E}[z]}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.01</td>
<td>0.15</td>
<td>10</td>
<td>0.91</td>
<td>0.07</td>
</tr>
<tr>
<td>0.1</td>
<td>0.05</td>
<td>0.18</td>
<td>15</td>
<td>1.36</td>
<td>0.073</td>
</tr>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>0.2</td>
<td>20</td>
<td>1.81</td>
<td>0.074</td>
</tr>
<tr>
<td>0.5</td>
<td>0.01</td>
<td>0.77</td>
<td>25</td>
<td>2.26</td>
<td>0.075</td>
</tr>
<tr>
<td>0.5</td>
<td>0.05</td>
<td>0.92</td>
<td>30</td>
<td>2.72</td>
<td>0.076</td>
</tr>
<tr>
<td>0.5</td>
<td>0.1</td>
<td>1.02</td>
<td>35</td>
<td>3.17</td>
<td>0.076</td>
</tr>
<tr>
<td>1</td>
<td>0.01</td>
<td>1.54</td>
<td>40</td>
<td>3.62</td>
<td>0.076</td>
</tr>
<tr>
<td>1</td>
<td>0.05</td>
<td>1.85</td>
<td>45</td>
<td>4.07</td>
<td>0.076</td>
</tr>
<tr>
<td>1</td>
<td>0.1</td>
<td>2.05</td>
<td>50</td>
<td>4.52</td>
<td>0.076</td>
</tr>
</tbody>
</table>

Table 3.1: Left table: Expected loss due to sampling and pruning at $t^* - 1$; Right table: Expected loss due to pruning at $t < t^* - 1$ for $\epsilon = 0.05$, $\delta = 0.5$, $d_{\min} = 1$, $\frac{U_{\max}}{U_p} = 2N$, and $m = 5$

Next we present numerical experimental results that show the $ASA^*$ algorithm is fast and efficient and most of the instances are solvable within a reasonable time for medium-sized instances.
Number of nodes and computation time for ASA*. Stochastic optimization methods that are based on enumerating scenarios are usually computationally expensive. The ASA* algorithm, however, is fast and tractable due to the use of sampling and pruning that allow for suppressing many nodes in the tree. More importantly, the ASA* is tunable in the sense that one can set \( \epsilon \) and \( \delta \) to change the number of nodes and thereby solve an instance within a desired amount of time, and yet ensure that the loss from using large \( \epsilon \) or small \( \delta \) is bounded (as shown in propositions 3.2.1-3.2.3).

Figure 3-2a shows the cumulative distribution of number of nodes (after pruning) for the tree up to \( t^* \), for a bay with 7 columns, 4 tiers, and 3 containers per column. It can be seen that for half of the instances, the tree up to \( t^* \) has 100 or fewer nodes. Also, about 90% of instances have fewer or less nodes. Note that the nodes of the trees that are constructed after time \( t^* \) for solving instances corresponding to different scenarios do not have too much of an effect on computation time because the A* algorithm is very fast.

Figure 3-2b shows the cumulative density of average computation time. It can be seen that half of the instances are solved in less than two minutes and 90% of instances are solved in 15 minutes or less.

Although the ASA* algorithm is fast for medium-sized bays, it might not be tractable for large bays. We next introduce 3 heuristics and evaluate their performance through numerical experiments.

### 3.3 Heuristic Approach

To the best of our knowledge, the CRP with incomplete information has not been much studied. Here we present 3 heuristics (all extensions of existing heuristics for the CRP with complete information). It is worth mentioning that unlike the ASA*, extending these heuristics to the multi-stage setting (where information is revealed gradually at multiple time-steps) is straightforward and computationally tractable.
Figure 3-2: Tractability of the ASA* algorithm: algorithm implemented with $\delta = 0.5$ and $\epsilon = 0.05$
**Myopic-\textit{MinMax} Heuristic.** This is an extension of the heuristic H that we explained in Section 2.5.2. The myopic-\textit{MinMax} heuristic works as follows:

Suppose \( n \) is the target container located in column \( c \). If \( n \) is not blocked by any container, it is retrieved without any relocation. Otherwise, let \( r \) be the topmost blocking container in column \( c \), and \( c^* \) be the column where \( r \) should be relocated. The myopic-\textit{MinMax} heuristic determines \( c^* \) using the same rules as the H heuristic explained in Section 2.5.2, except that we assign the index of \( N + 1 \) to all unknown containers (containers \( \{ |K|, |K|+1, \ldots, N \} \)) and we set \( \min(c_i) = N+2 \) if \( c_i \) is empty. In case of a tie, container \( r \) is placed on top of the highest column (i.e., the column which has more containers than the others). Further ties are broken arbitrarily.

**Myopic-\textit{RI} Heuristic.** This is an extension of the Reshuffling Index (RI) heuristic that is introduced by Murty et al. [19]. Let \( r \) be the container that needs to be relocated. For each column in the bay, \( RI \) is zero if the column is empty. Otherwise, \( RI \) is equal to the number of containers that are already in the column and the departure time of these containers is earlier than the departure time of \( r \). Container \( r \) is then relocated to the column with the lowest \( RI \). In case of a tie, container \( r \) is placed on top of the highest column (i.e., the column which has more containers than the others), and the assignment is done randomly for a further tie.

Note that we need to know the departure time of all containers to compute the \( RI \). In order to extend this heuristic to the CRP with incomplete information, we assign index of \( N + 1 \) to all unknown containers. If \( r \) is itself a known container, the unknown containers in the candidate column do not need to be counted (because they will be retrieved after \( r \)); thus \( RI \) equals the number of known containers that are in the candidate column and will be retrieved earlier than \( r \). If \( r \) is an unknown container and there exists some unknown containers in the candidate column, we compute \( E[RI] \), that is, the number of known containers below \( r \) plus the expected number of unknown containers that will be retrieved later than \( r \).\footnote{This heuristic is similar to the \textit{ERI} heuristic proposed by Ku [16] for the case that a departure time window of containers is known.} To compute \( E[RI] \), we assign equal probabilities to all different departure orders among unknown
containers (including \( r \)) in the candidate column. For example, if there only exist 2 unknown containers below \( r \), there will be 3 possible departure orders: \( r \) may be the first, second, or the last to be retrieved; hence \( \mathbb{E}[RI] = 0 \times \frac{1}{3} + 1 \times \frac{1}{3} + 2 \times \frac{1}{3} = 1 \).

In general, if after placing \( r \) on top of a column, there will be \( n_1 \) known containers (with earlier departure time) and \( n_2 \) unknown containers below it, we have:

\[
\mathbb{E}[RI] = n_1 + \sum_{k=1}^{n_2} \frac{n_2(n_2+1)}{2} \cdot \frac{1}{n_2 + 1} = n_1 + \frac{n_2}{2}.
\]  

(3.20)

We use \( \mathbb{E}[RI] \) and follow the same rules as in the \( RI \) heuristic to assign a column to \( r \).

**Myopic-ATIB Heuristic.** This is an extension of the average time index-based (ATIB) heuristic proposed by Hakan Akyüz and Lee [12]. The ATIB heuristic tries to relocate container \( r \) to a column with the latest average departure time. To achieve this, for each column, the average departure time of existing containers is calculated and compared with the departure time of \( r \). Then, a weight of zero is assigned to any column that its average departure time is larger than the departure time of \( r \). A weight of 1 is assigned otherwise. Then \( r \) is placed on top of the column with the lowest weight. In case of a tie, \( r \) is relocated according to the lowest reshuffling index (explained above). In case of a further tie, the assignment is done using the same tie breaking steps described in the \( RI \) heuristic. In order to extend this heuristic to the CRP with incomplete information, we assign index of \( N + 1 \) to all unknown containers and follow the same steps as in the \( ATIB \) heuristic.

**3.3.1 Benchmark of the Myopic Heuristics**

In this section we present the results of numerical experiments on different bay sizes and with the 3 heuristics explained in Section 3.3.

Figures 3-3 to 3-5 show the ratio of the number of relocations between the heuristic and that for the case of full information. The horizontal axis is the number of columns in the bay. The vertical axis shows \( \frac{z_{heuristic}}{z_H} \), where \( z_H \) is the number of relocations.
Figure 3-3: Performance of the myopic heuristics when information level is \([0.25N]\)

computed by heuristic H (explained in Section 2.5.2), and \(z_{\text{heuristic}}\) is the number of
relocations computed by one of the myopic heuristics. For each bay size (with \(C\)
columns), we compute the average of relocations over 10000 random instances with
\(C\) columns and 4 tiers, with the topmost tier initially empty.

In Figures 3-3 to 3-5, the amount of information initially available is \([0.25N]\),
\([0.5N]\), and \([0.75N]\), respectively \((N\) is the number of containers in the bay and \([x]\)
is the smallest integer larger than \(x\)). Also, for all the incomplete information cases,
we fix \(t^*\) at \([0.25N] + 1\) to ensure that at every time-step before \(t^*\), at least one
container is known. In each figure, the line with square markers corresponds to the
lowest height heuristic which basically places a container on top of the column that
has the fewest number of containers (ties are broken arbitrarily). The lowest height
Several observations can be made from Figures 3-3 to 3-5. First and foremost, the myopic-$RI$ heuristics outperforms the other heuristics for any bay size and any information level; its gap with the case of full information is less than 5% when the information level is 25%, and the gap drops to 1% when 75% of containers are initially known. Second, for any information level, the myopic-$RI$ and myopic-$ATIB$ heuristics outperform the myopic-$MinMax$ heuristic and the performance gap between them shrinks as more information is initially available. Last, the myopic-$MinMax$ heuristic is worse than the lowest height heuristic when the information level is 25%.
1.25

1.2

1.15

Myopic-MinMax heuristic

Myopic-RI heuristic

Myopic-ATIB heuristic

Lowest height

1.05

-------------- --

10 20 30 40 50 60 70

Number of columns of the bay

Figure 3-5: Performance of the myopic heuristics when information level is $[0.75N]$

3.4 Value of Information

We study the value of information initially available ($|K|$), using the $ASA^*$ algorithm and the myopic-$MinMax$ heuristic. The same qualitative results hold for the myopic-$RI$ and myopic-$ATIB$ heuristics. For the sake of brevity, we show only the plots for the myopic-$MinMax$ heuristic.

We take 1000 random instances of a bay with 7 columns, 4 tiers, 21 containers, and 3 containers per column. We solve each instance with 6 levels of information: $|K| = [0.25N], [0.375N], [0.5N], [0.625N], [0.75N]$, and $[0.9N]$, using the $ASA^*$ algorithm and the myopic-$MinMax$ heuristic ($[x]$ is the smallest integer larger than $x$). For all cases, we fix $t^*$ at $[0.25N] + 1$ to ensure that at every time-step before $t^*$, at least one container is known. For each of the 12 cases, we compute the average number of relocations over the 1000 instances. We then compare the relative gap of
each case with the average relocation for the CRP with full information: \( \frac{z_{ASA^*} - z_{opt}}{z_{opt}} \)
and \( \frac{z_{MMH} - z_{opt}}{z_{opt}} \), where \( z_{MMH} \) is the average number of relocations obtained by the myopic-\( MinMax \) heuristic.

Figure 3-6a shows the relative gap in the number of relocations for different levels of information. With the \( ASA^* \) algorithm, the gap is about 8% when 25% of the containers (6 containers) are known at time zero. The gap reduces to 3% when half of the containers are initially known and is almost zero when 90% of containers are known at time zero. The same behavior can be observed for the heuristic \(^2\). In both cases, the marginal value of information becomes smaller when more information is available. This is more significant for the heuristic: for example, when the level of information increases from 25% to 50%, there is a significant drop in the gap; then the gap decreases more slowly and approaches zero at 100% information. Note that with 100% information, the myopic-\( MinMax \) heuristic is the same as \( H \) heuristic.

To gain insight into the value of information for the myopic-\( MinMax \) heuristic, recall that the it behaves similar to the \( H \) heuristic as long as \( \min(c_j) \) for all columns are known. Since all unknown containers have larger indices than known containers, knowing at least one container in each column is sufficient to get the same \( \min(c_j) \) as in heuristic \( H \). Thus, after some point, having more information does not have much of an effect on the number of relocations when we use the myopic-\( MinMax \) heuristic.

Figure 3-6b shows how the gap between the myopic-\( MinMax \) heuristic and the \( ASA^* \) shrinks as the level of information increases. When 25% of containers are initially known, using the myopic-\( MinMax \) heuristic results in 12% more relocations on average (compared to \( ASA^* \)). This gap drops to less than 2% when all containers are initially known. Note that when more than 50% of containers are initially known,

---

\(^2\)The gap between the heuristic and the full information is smaller for myopic-\( RI \) and myopic-\( ATIB \) as demonstrated in Figures 3-3 to 3-5. To avoid clutter in the plot, we show only the gap for the myopic-\( MinMax \).
(a) Value of information: percentage difference in number of relocations with the case of full information.

(b) Comparison of the myopic-\textit{MinMax} heuristic and the \textit{ASA*} algorithm: percentage difference in number of relocations for different levels of information.

Figure 3-6: Experiment results for medium-sized instances (7×4 bay)
the solution provided by the myopic-$MinMax$ heuristic is reasonably close to the solution of $ASA^*$ (the gap is less than 5%). Therefore, the myopic-$MinMax$ heuristic can be used in practice as it is easy to implement and efficient.

It is interesting to see the effect of different levels of information on the number of relocations in larger bays. As demonstrated in Figures 3-3 to 3-5, the ratio of $E[z_{heuristic}]$ and $E[z_H]$ does not constantly increase with bay size. For each heuristic and for each information level, this ratio converges to a constant as the bay gets larger. Moreover, the ratio drops quickly as the level of information increases. For example, for the myopic-$MinMax$ heuristic, the ratio converges to 40% when 25% of the containers are initially known. This number drops to 3% when the level of information is 75%.

What is shown in Figures 3-3 to 3-5 provides a useful insight for port operators. Considering the gain at each level of information, port operators can design appointment systems that capture this gain by offering faster or discounted service to customers who provide their arrival times in advance.

3.5 Concluding Remarks

In this chapter, we studied the CRP with incomplete information, i.e., the setting where the departure order of containers is not known in advance and only some probabilistic distributions of the departure order may be available. We formulated the problem in a 2-stage setting and adapted the $A^*$ method to develop a stochastic optimization algorithm for the 2-stage problem. The resulting algorithm, called $ASA^*$ requires constructing a decision tree whose size depends on the bay size, the number of containers in the bay, and the number of scenarios that need to be considered for unknown containers. To overcome the exponentially and fast growing size of the tree, we only considered a random sample of possible scenarios for the unknown containers. Moreover, we used a pruning rule (also based on sampling) to further

---

3 Notice that the performance of myopic heuristics can be considered as an upper bound on the performance of the $ASA^*$ algorithm.
limit the number of nodes in the tree. We bounded the approximation errors due to sampling and pruning and showed that such errors are small compared to the value of the objective function (number of relocations). We also showed that sampling and pruning makes the algorithm reasonably fast and thereby an efficient tool to be used in practice for medium-sized instances. More importantly, the ASA* algorithm can be used as a benchmark for evaluating heuristics for the CRP with incomplete information.

Given the computational intractability of the ASA* algorithm for large instances, we also introduced three myopic heuristics that are easy to implement, and demonstrated through numerical experiments that they perform well on average. In particular, we compared those heuristics with the H heuristic for the CRP with complete information for large bays and for different levels of information. Our experiments showed that the gap between the myopic heuristics and the H heuristic converges to a constant as the bay gets larger. Hence, given the decent performance and the short computation time of the myopic heuristics, they can be used in real-time operations in ports.

Lastly, we studied the value of information by demonstrating the impacts of different levels of information (i.e., the number of containers initially known in the two-stage setting of the problem) on the number of relocations. In particular for medium-sized instances, we showed that the number of relocations with the ASA* algorithm (when the information level is more than 25%), is within 5% of the optimal number of relocations with complete information; and the gap drops to less than 1% when the information level is 75% more.
Chapter 4

Simulation Study of Port Operations

In this chapter we introduce a simulator that we developed for simulating different aspects of port operations in an integrated framework. This simulator is programmed in MATLAB and replicates the operations involving import containers on the sea-side and land-side. It enables users to study the effects of different policies and heuristics on the efficiency of the operations, in particular on the number of relocations and truck wait times.

In Section 4.1, we describe different modules of the simulator and our main modelling assumptions. In Section 4.2 we describe the user interface and its features, and in Section 4.3, we present the results of some experiments that we performed using our simulator.

4.1 Simulator

The simulator is designed to replicate operations on the sea-side and land-side and it models many practical details of operations. To this end, the modelling horizon is discretized into small enough time-steps and one or more tasks (including discharge, stacking, retrieval, and relocation) are carried out at each time-step. A time-step

---

1Our assumptions are based on the inputs provided by the operators of a maritime port in the middle east. Many of our simulation inputs (such as yard capacity, number of equipment, and completion time of different tasks) are also based on the information provided to us.

2We set the length of a time-step to one minute which is small enough compared to the completion time of different tasks (for example, a relocation move takes 3-5 minutes and discharging a container
may also be idle if there is no task to complete.

The simulator can be used for static or dynamic operations, and with complete or incomplete information. It consists of an initialization module, a module for updating block configuration after completion of a move, a module for information update (for the case of incomplete information), and four modules for the four main operations (discharge, stacking, relocation, and retrieval). These modules are explained below.

**Initialization.** This module initializes the yard layout (blocks of containers and their configuration) and arrival and departure times of containers. It also generates data structures for each container and equipment (including internal trucks, RTG cranes, berth cranes, and ships).

Configuration of each block is described by two $C \times P \times B$ matrices where $C$ is the number of columns in one bay, $P$ is the number of tiers, and $B$ is the number of bays in the block. In both matrices, each argument corresponds to one slot in the block. Arguments of the first matrix are the IDs of containers located in the block, and arguments of the second matrix are rankings of the containers based on their departure time. In the case of incomplete information, an unknown container has a rank larger than the rank of all known containers similar to what we explained in Section 3.3 for the myopic heuristics, except that here unknown containers can have different ranks depending on the departure zone they belong to. We explain departure zones in the information update module.

The arrival of external trucks and ships is modelled as follows. Arrival times of the external trucks are generated according to a Poisson process with a fixed rate that is a user input. The generated arrival times are then shifted such that on each day, external trucks arrive over a period of 12 hours starting from the beginning of the day. Notice that each generated arrival time corresponds to the departure time of a container (including those that are already in the yard and those that will arrive and be stacked later). Ships also arrive according to a Poisson process with a fixed rate specified by the user. In addition to arrival rates, the number of ships arriving takes 2-3 minutes).
daily, and the number of containers on each ship are also specified by the user.

**Block configuration update.** After completion of a move (stacking, retrieval, or relocation) in a block, two main functions are executed to update the configuration of that block:

(i) Containers ID update: The ID matrix is updated as follows:

- The argument corresponding to the slot of the retrieved container or the current slot of the relocated container is set to zero; and
- The argument corresponding to the slot of the stacked container (the new slot of the relocated container) is set to the ID of the stacked container (relocated container).

(ii) Containers ranking update: Ranking of the containers are recalculated and updated given the updated ID matrix.

**Information update.** This module handles the gradual reveal of information by updating the departure zone of unknown containers according to the assumptions for information update as explained below. We assign each container in the storage yard and each incoming container to one of the following three time zones. At each time-step \( t \), we update the departure zone of containers according to \( t \) and their departure times.

- **Zone 0:** These are the known containers, i.e., their exact departure times are known and are used when applying heuristic rules. In our simulations, at each time-step \( t \), we assign a container to zone 0 if its departure time is between \( t \) and \( t + t^* \). We do simulations with different values of \( t^* \) (ranging from 30 minutes to 3 hours) to study the value of information.

- **Zone 1:** These are the containers that are going to be retrieved by the end of the day. In our simulations, at each time-step \( t \), we assign a container to zone 1 if its departure time is between \( t + t^* \) and \( 1440 \times \left\lceil \frac{t}{1440} \right\rceil \).
• Zone 2: These are the containers that will not be retrieved by the end of the day and will be retrieved at some unknown later time. In our simulations, at each time-step \( t \), we assign a container to zone 2 if its departure time is greater than \( 1440 \times \left\lfloor \frac{t}{1440} \right\rfloor \).

Discharge. This module handles the sea-side operations, that is discharging containers from a ship and transferring them to the storage yard. Upon arrival of a ship, three main functions are executed in this module:

(i) Allocation of berth cranes: Depending on the number of containers in the ship and number of berth cranes available, one or more berth cranes are assigned to the ship.

(ii) Allocation of internal trucks: Depending on the number of internal trucks available and the number of busy berth cranes, one or more trucks are assigned to each berth crane.

(iii) Container discharge and transfer: Containers are discharged from the ship according to a given discharge order. Each container is transferred to the storage yard using an available internal truck.

(iv) Block allocation: A block in the storage yard is assigned to each container as soon as it is discharged and an arbitrary slot is saved for the container in that block.

Stacking. This module handles the stacking process in the storage yard. Upon arrival of an internal truck at the block (that was determined in the block allocation function of the discharge module), two main functions are executed:

(i) Stacking slot allocation: A row and a column in the block is assigned to the container.

(ii) Container stacking: The stacking move is performed, i.e., the block configuration is updated, the RTG crane becomes available for the next move, and the internal
truck goes back to the sea-side (and becomes available for transferring another container)

**Relocation.** This module handles the relocation process in the storage yard. Two main functions are executed in this module:

(i) Relocation slot allocation: A row and column is assigned to the blocking container.

(ii) Container relocation: The relocation move is performed; i.e., the block configuration is updated and the RTG crane becomes available for the next move.

**Retrieval.** This module handles the retrieval process in the storage yard. Two main functions are executed in this module:

(i) Unblocking the target container: The blocking containers on top of the target container are relocated using the relocation module.

(ii) Container retrieval: Once all blocking containers are relocated, the retrieval move is performed; i.e., the block configuration is updated and the RTG crane becomes available for the next move.

At each time-step, container departure zones are updated in the information update module. Then for each berth crane and each RTG crane, depending on its availability, one of the four main tasks is initiated if there is any task that needs to be completed. Several heuristic rules are used to make the series of decisions for these tasks. Some of these decisions (like finding a slot for a containers that is being relocated) and their corresponding heuristics were discussed in previous chapters. Others are explained below.

Table 4.1 summarizes the heuristics that are used by our simulator for making decisions in different modules. The decisions include assigning a block to a discharged container, assigning a row and column to a container that needs to be stacked, and
assigning a row and column to a container that needs to be relocated. Several heuristics were explained in Sections 2.6 and 3.3 for a relocation move. We use the same heuristics to assign a slot to a container that needs to be stacked in two steps: First, we assign a block to a container that is discharged. In order to do so, we find a slot by applying one of the heuristics to all rows and columns in all non-full blocks of the storage yard. We then choose the block that has the best slot for the incoming container (according to the heuristic) and save one arbitrary slot in that block for the container (i.e., we decrement the number of free slots in the block by one). In case of a tie, we choose the block that has the shortest queue of internal trucks, i.e., the block that has the minimum number of containers waiting to be stacked in it. Further ties are broken arbitrarily (block with smallest index is chosen). Once a block is assigned to a container, it takes the internal truck a few minutes (depending on the distance between the berth crane and the assigned block) to arrive at the block. Because during this time the configuration of that block can change (by stacking, relocating, or retrieving another container), we re-apply the heuristic to all columns in that block when the internal truck arrives. This is the second step in assigning a slot to an incoming container. If there is an external truck awaiting a retrieval and an internal truck awaiting a stacking at the same block, we give priority to the stacking task, unless the external truck has waited more than a maximum allowed wait time (defined in the simulator).

Other decisions that need to be made include berth crane allocation and internal truck allocation. As explained in the discharge module, we allocate them based on the number of available machines and number of containers in a ship.

<table>
<thead>
<tr>
<th>Heuristics</th>
<th>Complete information</th>
<th>Incomplete information</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>myopic-MinMax</td>
<td></td>
</tr>
<tr>
<td>RI</td>
<td>myopic-RI</td>
<td></td>
</tr>
<tr>
<td>ATIB</td>
<td>myopic-ATIB</td>
<td></td>
</tr>
<tr>
<td>lowest height</td>
<td>lowest height</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1: Heuristics used in the simulator
4.2 User Interface

The operational parameters that are specified by the user are as follows:

- Storage yard capacity: Number of blocks, number of rows per block, number of columns per row, number of tiers in each column, and initial capacity utilization rate of the yard ($\gamma$);

- Port equipment: Number of berth cranes, number of RTG cranes in each block, and number of internal trucks;

- Traffic parameters: arrival rate of external trucks ($\lambda$), arrival rate of ships ($\mu$), and number of ships arriving daily;

- Information setting (complete information or incomplete information) and the horizon length for the case of incomplete information;

- Intra-bay relocation moves (selecting this option specifies that relocations are allowed only in the same bay where the container is located. Relocation to other bays in the same block may be performed if this option is deselected);

- Simulation length specified in time-steps (a time-step is equal to one minute);

- Relocation and stacking heuristics to be used in the simulation.

The interface that allows the user to enter these inputs is demonstrated in Figure 4-1.

Changing different inputs such as traffic parameters, stacking and relocation heuristics, and horizon length allows for studying different aspects of the operation. In the next section, we present the results of two studies performed using the simulator.
4.3 Experiments

The results presented in this section are from simulating three days of operations with the parameters described in Table 4.2. The arrival rate of the ships is set such that on average 6 ships arrive during the first 6 hours of the day. The arrival rate of external trucks is a variable that is changed across the simulations and is set such that on average \( n_e \) trucks show up during the first 12 hours of the day.\(^3\)

We compare the results obtained by the myopic-\( \text{MinMax} \) and the myopic-\( \text{RI} \) heuristics (the myopic-\( \text{ATIB} \) heuristic has a similar performance to myopic-\( \text{RI} \)). For the case of complete information, we compare the H heuristic with the reshuffling index heuristic. In each simulation, the same heuristic is used for stacking and relocation moves.

In our simulations, we change the value of \( n_e \) (the average number of external trucks arriving daily) ranging from 2000 to 3000. Note that with 6 ships arriving daily, each having 350 containers, there is a balance between the number of containers retrieved and the number of containers stacked when \( n_e = 2100 \). For \( n_e > 2100 \), the yard capacity utilization will go below the initial value of 90%.

We run simulations with different values of \( t^* \), with each \( t^* \) indicating the infor-

\(^3\)We set \( r_e = \frac{2n_e}{1440} \) and for each day we shift the arrival times of the trucks arriving in the second half of the day such that they arrive over the next day.
<table>
<thead>
<tr>
<th>Yard capacity and equipment</th>
<th>Traffic and capacity utilization*</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td># Blocks</td>
<td>30</td>
</tr>
<tr>
<td># Rows per block</td>
<td>50</td>
</tr>
<tr>
<td># Columns per row</td>
<td>7</td>
</tr>
<tr>
<td># Tiers</td>
<td>4</td>
</tr>
<tr>
<td># Berth cranes</td>
<td>12</td>
</tr>
<tr>
<td># RTG</td>
<td>30</td>
</tr>
<tr>
<td># Internal trucks</td>
<td>58</td>
</tr>
</tbody>
</table>

* For capacity utilization rate of u%, up to \( \frac{u}{100} \times C \times (P - 1) \) slots in each row of a block are occupied.

Table 4.2: Value of simulation parameters

Information level. Recall that at any time, departure times of the containers that will depart in the next \( t^* \) time-steps are known. In our simulator, each time-step is one minute. For the case of "no" information, we assume that \( t^* = 15 \). The rationale for this value is that once an external truck checks in at the entrance gate, it typically takes 15-20 minutes for it to arrive at the yard to pick up its container. Therefore, it is reasonable to assume that the departure times of containers are known 15 minutes in advance even if there is no appointment system.

We first look at the average number of relocations (per retrieval). In Figure 4-2, the vertical axis shows the percent difference between the average number of relocations obtained by myopic-\textit{MinMax} and myopic-\textit{MinMax}. The numbers on the horizontal axis correspond to different levels of traffic in the storage yard, and the four lines in the plot correspond to different levels of information.

Consistent with the results of the numerical experiments in Section 3.3, the myopic-\textit{RI} heuristic outperforms the myopic-\textit{MinMax} in terms of the average number of relocations. Two other observations can be made. First, the gap between the two heuristics diminishes as the arrival rate of external trucks becomes larger, i.e, the yard capacity utilization rate becomes smaller. This makes sense intuitively, because when the yard is emptier and there are more empty columns, the two heuristics are very likely to choose the same column (the empty one). Second, the gap between the
two heuristics, for each traffic level, shrinks as the information level increases. The reason is that the average number of relocations for the myopic-\textit{MinMax} heuristic constantly decreases as the information level increases. For the myopic-\textit{RI}, however, the average number of relocations is almost constant for all information levels (except when full information is available)\footnote{Intuitively, this happens because for the myopic-\textit{MinMax} heuristic, all columns with no known containers are the same (they all get index of \(N + 1\)). As soon as one container in any such column becomes known, the minimum index for that column changes and the heuristic is very likely to make a different choice. For the myopic-\textit{RI}, however, a few more known containers are not very likely to change the choice made by the heuristic unless most containers in one column become known such that the score of that column changes sufficiently, thereby becoming a worse or better candidate column than others.}.

![Figure 4-2: Percentage difference in number of relocations between myopic-\textit{MinMax} and myopic-\textit{RI} heuristics](image)

Moreover, notice that as shown in Table 4.3, performance of the myopic-\textit{MinMax} heuristic depends on the information level, whereas the myopic-\textit{RI} heuristic results in almost the same average number of relocations regardless of the information level, unless full information is available. Therefore, in practice, the myopic-\textit{RI} heuristic can be used as it has a satisfactory performance even with very little or no information.
We also compare the average customer wait time and the average stacking delay (average amount of time it takes to stack a container in the yard after it is discharged from the ship). Table 4.3 summarizes the average wait time and the average stacking delay for the case that \( n_e = 2000 \) (which means that the yard capacity utilization rate remains almost constant at 90%). First notice that unlike the number of relocations, the average wait time and the average stacking delay do not depend on the information level (unless full information is available). Moreover, the myopic-RI heuristic results in less average wait time and less average stacking delay compared to the myopic-MinMax heuristics. The difference is more significant for the average stacking delay which is about 12 minutes with the myopic-MinMax and about 5 minutes with the myopic-RI heuristic.

<table>
<thead>
<tr>
<th></th>
<th>( t^* = 15 )</th>
<th>( t^* = 60 )</th>
<th>( t^* = 120 )</th>
<th>( t^* = 80 )</th>
<th>Full information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Average wait time</strong></td>
<td>myopic-MinMax</td>
<td>18.03</td>
<td>16.17</td>
<td>16.29</td>
<td>16.32</td>
</tr>
<tr>
<td></td>
<td>myopic-RI</td>
<td>16.33</td>
<td>16.38</td>
<td>16.38</td>
<td>16.35</td>
</tr>
<tr>
<td><strong>Average stacking delay</strong></td>
<td>myopic-MinMax</td>
<td>12.66</td>
<td>12.42</td>
<td>12.15</td>
<td>12.4</td>
</tr>
<tr>
<td></td>
<td>myopic-RI</td>
<td>4.79</td>
<td>4.78</td>
<td>4.78</td>
<td>4.78</td>
</tr>
<tr>
<td><strong>Average number of relocations</strong></td>
<td>myopic-MinMax</td>
<td>0.7</td>
<td>0.66</td>
<td>0.63</td>
<td>0.63</td>
</tr>
<tr>
<td></td>
<td>myopic-RI</td>
<td>0.51</td>
<td>0.52</td>
<td>0.52</td>
<td>0.52</td>
</tr>
</tbody>
</table>

Table 4.3: Average customer wait time, average stacking delay, and average number of relocations for different levels of information with the myopic-MinMax and myopic-RI heuristics; \( n_e = 2000 \)

Lastly, for the myopic-RI and \( t^* = 15 \), we show the average customer wait time and average stacking delay for different traffic levels in Table 4.4. Contrary to the average wait time that increases with the traffic level, the stacking delay remains constant regardless of the arrival rate of external trucks. \(^5\)

\(^{5}\)Note that the average wait times for \( n_e > 2000 \) are unreasonably large because the number of stacked and number of retrieved containers are not balanced. As a result, the yard capacity utilization rate exceeds 90%, a case that rarely happens in practice. We provide these results to demonstrate how the average wait time and average stacking delay change in extreme cases of traffic and congestion in the yard.
<table>
<thead>
<tr>
<th>( n_e )</th>
<th>2000</th>
<th>2250</th>
<th>2500</th>
<th>2750</th>
<th>3000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average wait time</td>
<td>16.33</td>
<td>41.99</td>
<td>68.59</td>
<td>85.88</td>
<td>102</td>
</tr>
<tr>
<td>Average stacking delay</td>
<td>4.79</td>
<td>4.84</td>
<td>4.89</td>
<td>4.83</td>
<td>5.01</td>
</tr>
</tbody>
</table>

Table 4.4: Average customer wait time and average stacking delay for different levels of traffic with the myopic-\( MinMax \) and the myopic-\( RI \) heuristics; \( t^* = 15 \)

4.4 Concluding Remarks

Operations in the storage yard are substantially interrelated with the operations on the sea-side and both have many practical details that cannot be efficiently captured in an optimization framework. In this chapter, we introduced a simulator that allows for integrated simulation of long periods of realistic-scale operations in the land-side and the sea-side. The simulator consists of separate modules for different parts of the operations (including stacking, retrieval, and relocation in the storage yard, and discharging, berth allocation, and internal truck allocation on the sea-side), thereby allowing users to study various policies and different heuristics for each type of operation.

As an example, we compared the performance of the myopic-\( MinMax \) and the myopic-\( RI \) heuristics in terms of the average number of relocations, the average customer wait time, and the average stacking delay over a period of 3 days with different levels of traffic and different levels on information. Our simulation results show that the performance of myopic-\( RI \) in terms of all three metrics does not depend on the information level. Moreover, the myopic-\( RI \) heuristic outperforms the myopic-\( MinMax \) heuristic. Although the myopic-\( RI \) heuristic requires slightly more computation effort than the myopic-\( MinMax \), it is still fast enough to be used in practice.
Chapter 5

Conclusions and Directions for Future Research

In the storage yard of container terminals, the import containers are retrieved from the bays to be delivered to the external trucks (retrieval process), and, the export containers are stacked in the bays after being discharged from the vessels (stacking process).

Managing the operations in the storage yard of container terminals is of great importance and practical interest, as any inefficiency in the storage yard can be a major bottleneck in the entire operations of the terminal. Among different aspects of the yard operations, the container relocation problem has been widely studied in the literature. As the problem title suggests, all studies have focused on the problem of minimizing the number of relocation moves which is a cost-based objective in the sense that relocation moves are costly for the port operators. The other important aspect which is the service-based objective of minimizing truck (customer) wait times, however, has been left out of most existing models and heuristics developed for the CRP.

In this thesis, we introduced a time-based mathematical model for the CRP and present an integer programming formulation and an extension of our formulation for the DCRP, i.e., the setting that containers are stacked and retrieved continuously. Exploiting the service times (departure and arrival times of containers), our model can
be used as a planning tool and provides port operators with the following flexibilities:

- Jointly minimizing the number of relocations and truck wait times and giving different priorities to them;
- Giving different waiting time guarantees to different customers;
- Planning repositioning moves in off-peak hours; and
- Planning the stacking and retrieval moves in the DCRP and optimizing the stacking order of containers.

Moreover, we presented a flexible retrieval policy and used our IP formulation to study this policy from the point of view of port operators and customers. We showed that a flexible retrieval policy can result in fewer relocations and less truck wait times and hence is aligned with the benefits of both port operators and customers. We also showed through experiments that the out-of-order retrievals in such a policy do not create inequity in the service.

A critical assumption of the CRP is that the retrieval order of all containers in the bay are considered to be known in advance. However, in practice, such information is not available far in advance and the exact departure order of containers becomes known only as their retrieval times approach. Relaxing the assumption of knowing departure times in advance results in the CRP with incomplete information which has not gained much attention.

In this thesis, we studied the CRP with incomplete information in a 2-stage setting, where the retrieval order of a subset of containers is known initially and the retrieval order of the remaining containers is revealed later but all at once. We developed a 2-stage approximate stochastic optimization algorithm that optimizes the expected number of relocations given a probabilistic distribution on the departure order of containers. We gave theoretical bounds on the approximation errors of our algorithm and show through experiments that it is fast and efficient. We also used our algorithm to study the value of information in medium-sized instances. Our experiments show that when the retrieval order of 50%-75% of containers is known at the beginning, the
loss from the missing information is negligible and the average number of relocations is very close to that of the CRP with full information. It is worth mentioning that the relocation problem also arises in other storage systems such as steel plate stacking and warehousing systems (see Kim et al. [13] and Zäpfel and Wasner [27] for the former, and Chen et al. [8] for the latter). These systems also face the challenge of retrieving with incomplete information. We believe our methodology can be applicable to the operations of these systems as well.

The CRP is notorious for its computational intractability and most research studies have designed heuristics in order to solve the problem, particularly for large instances. In this thesis, we also proposed different heuristics for the CRP with complete and incomplete information and evaluate their performance through numerical experiments and benchmark them against optimization methods. For the CRP with complete information, we developed a heuristic (that has a decision tree framework) and by construction can improve upon any existing heuristic. For the CRP with incomplete information, we proposed three myopic heuristics and demonstrated that they perform very well on average and the resulting number of relocations is very close to the number of relocations obtained by the heuristic for CRP with complete information.

We also developed a simulator that replicates the operations in the storage yard of the container terminal and on the sea-side. This simulator provides flexibility in terms of capturing the practical details of the operations that cannot be modelled in an optimization framework. Moreover, given its computational tractability, the simulator can be used as a tool for studying different aspects of operations in realistic-scale and over long periods of time (for example, simulating a week of operations, with parameters used in practice, takes less than ten minutes to complete).

Although the CRP has been studied in the literature of container terminals and also in the context of other applications, there are still many untackled challenges.

Existing models and heuristics for the CRP with complete information lack in capturing customer-centric elements, in particular truck wait times. Our time-based mathematical model and IP formulation for the CRP and DCRP take into account
the cost-based and service-based objectives. However, because the run times of the integer program are prohibitively long, the design of different heuristics to provide quality solutions is warranted. Moreover, future research on the DCRP might include further studying the importance of the stacking process (stacking order and stacking slots) on the number of relocations and wait times during the retrieval process. Also, future studies could design heuristics for continuous stacking and retrieval processes.

One of the main issues that has been minimally addressed in the literature of the CRP is uncertainty. The majority of mathematical models and heuristics for the CRP are based on the unrealistic assumption that arrival and departure times of containers are known in advance. We studied the CRP with incomplete information in this thesis and proposed a stochastic optimization model that is based on one-time information reveal. Additional research could extend this model to a multi-stage setting where information is revealed gradually over time. Moreover, typically in practice, information from the appointment systems or historical data provides a departure time window for containers. Future studies of CRP with incomplete information might translate such time windows into departure orders such that they can be used in a scenario-based optimization model (like the one presented in this thesis). Alternatively, models or heuristics that are based on departure time windows could be developed.
Bibliography


