Optimization of hinterland container transportation and terminal operations

Bernard Zweers

Contents

1 Introduction **1**

1.1 Multimodal transportation 3

1.2 Terminal operations 5

1.3 Overview of this dissertation 7

2 Optimizing barge utilization **11**

2.1 Introduction 11

2.2 Literature review 13

2.3 Problem formulation 14

2.4 Mathematical model 20

2.5 Solution methods 27

2.6 Numerical results 39

2.7 Conclusion 43

3 Planning hinterland transportation in congested deep-sea terminals **45**

3.1 Introduction 45

3.2 Literature review 46

3.3 Problem formulation 48

3.4 Mathematical model 54

3.5 Solution methods 58

3.6 Numerical results 69

3.7 Conclusion 80

4 Minimum cost paths with stochastic travel times and overbooking **83**

4.1 Introduction 83

4.2 Literature review 85

4.3 Problem formulation 87

4.4 Mathematical model 88

4.5 Solution method 91

4.6 Numerical results 99
4.7 Conclusion 103

5 Pre-processing moves in container yards 105
5.1 Introduction 105
5.2 Literature review 106
5.3 Problem description 110
5.4 Mathematical model 116

6 Optimizing pre-processing and relocation moves 125
6.1 Solution methods 125
6.2 Numerical results 142
6.3 Conclusion 148

7 Limited number of pre-processing moves 151
7.1 Solution methods 151
7.2 Extension to multiple bays 163
7.3 Numerical results 170
7.4 Conclusion 178

8 Approximation algorithms for cluster capacitated problems 181
8.1 Introduction 181
8.2 Literature review 184
8.3 Preliminaries 186
8.4 Maximum coverage problem with knapsack constraints 188
8.5 Maximum coverage problem with cluster constraints 198
8.6 Multiple knapsack problem with cluster constraints 203
8.7 Capacitated facility location problem with cluster constraints . 215
8.8 Conclusion 218

9 Conclusion 221

Bibliography 225

Summary 235

Samenvatting 239

1
Introduction

Although they are just simple metal boxes, the influence of shipping containers on the global economy cannot be overestimated. When we think of the important inventions of the twentieth century, usually communication methods such as radio, television, and computers come to mind. In the logistics and transportation domain, the invention of airplanes and the mass production of cars are the noticeable changes in that century. However, the rise of containerized transport after World War II is one of the main driving forces of the economy's current state of globalization (Headrick, 2009, Chapter 8).

Before World War II, the introduction of steamships and trains made transportation itself already rather cheap. However, loading and unloading a ship or train was extremely time-consuming and labor-intensive (Krugman, 2011). Goods were packed in barrels, bags, or wooden crates, but they often needed to be loaded and unloaded one by one. This slow process resulted in the fact that ships spent about two-thirds of their time in ports (Bernhofen et al., 2016). Moreover, many longshoremen worked under harsh working conditions.

The introduction of the container made shipping inexpensive because the process of loading and unloading a ship goes faster and fewer longshoremen are needed. The benefit of containers is that they have standardized dimensions. The size of a container is defined by *Twenty-foot Equivalent Units* (TEUs), which corresponds to the length of the container. The most commonly used containers have a length of twenty-foot and forty-foot, so 1 and 2 TEU, respectively. Some containers have a length of forty-five feet, which is considered as a 3 TEU container. The standardized dimension enables specialized equipment to handle the containers. A single crane operator can unload a forty-foot long container in less than two minutes (Levinson, 2016, Chapter 1). Consequently, a single person can unload many more goods than before the introduction of containers in the same amount of time. In the early years of the 1970s, labor productivity in terms of tons per hour in ports was about twenty times higher than at the beginning of the 1960s (Bernhofen et al., 2016). The drop in transportation costs caused that goods produced in other parts of the world with lower wages became cheaper than locally produced goods. Hence, nowadays,

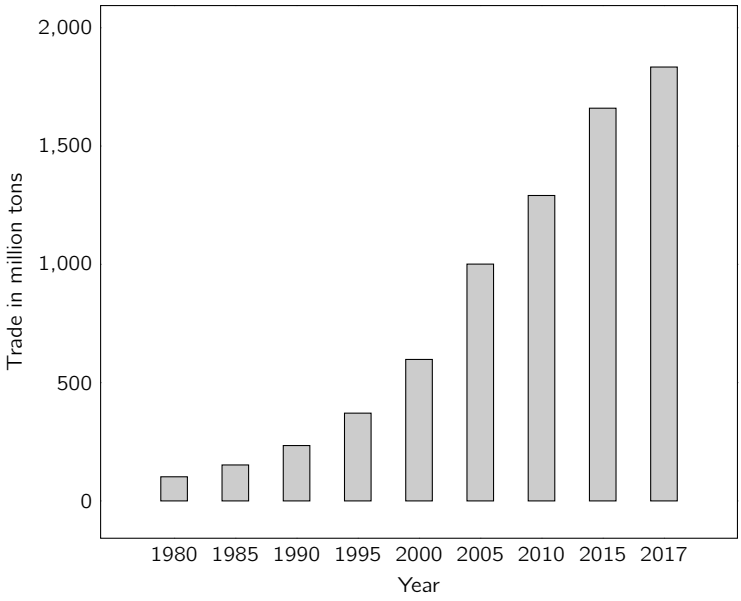


Figure 1.1 International seaborne trade that is carried by containers (Statista, 2020).

consumers can choose out of a much wider variety of products (Levinson, 2016, Chapter 1).

Nowadays, it is impossible to imagine trade without containers. About 75% of the global trade volume is carried by sea, and half of that part is shipped in containers (Lee and Song, 2017). In Figure 1.1, the historical amount of international seaborne trade shipped in containers is shown. We see a steady increase in the containerized trade over the years, resulting in the fact that in 2017 the number of goods shipped per container was 18 times higher than in 1980. The deep-sea vessels that are transporting these containers are also increasing in size, and the largest ships currently have a capacity of over 20,000 TEU. These large vessels usually only visit a few ports, and consequently, at these ports, many containers are transshipped. For instance, at the port of Rotterdam, the largest container port outside Asia (World Shipping Counsel, 2020), about 8.8 million containers were transshipped in 2019 (Port of Rotterdam Authority, 2020a). This number of yearly containers transshipped corresponds to almost 25,000 containers per day. However, the deep-sea ports are usually not the final destination of a container. The enormous number of containers requires efficient transportation to the final destination. The transportation from the deep-sea port to the final destination is referred to as *hinterland transportation* or *inland transportation*.

For efficient hinterland transportation, *inland container terminals* are crucial. An inland terminal has a barge and/or a train connection to one or multiple deep-sea ports. The barges and trains can be used to ship many containers simultaneously from the deep-sea port to the inland terminal. The trip from the inland to the destination is made with a truck because most customers do not have their own rail or waterway connection. Inland terminals enable the bundling of containers, which is more cost-efficient and releases some pressure of the operations in deep-sea terminals.

To optimize hinterland transportation decisions have to be made on different planning levels. In transportation problems, often three different planning levels are considered: *strategic*, *tactical*, and *operational*. Strategic problems typically require a large capital investment, and thus, the planning horizon is often multiple years. In tactical problems, the goal is to employ available resources in the most efficient way. Operational problems have a short-term horizon and a detailed description of the attributes of the planning (Crainic and Laporte, 1997). To illustrate these different levels, consider we would like to build an inland terminal with a barge service. Examples of strategic problems are the decision of the location of the inland terminal and the type of barges to buy. These barges often sail to deep-sea ports according to a fixed weekly schedule. Determining this schedule is a tactical problem. After we know which barge will visit which port, it has to be decided which containers will be loaded on that barge. This decision is an operational problem.

The motivation for this dissertation's research comes from a close collaboration with an inland terminal in the port of Amsterdam. In this dissertation, we focus on two types of operational problems encountered by inland terminals. The first problem is finding the cheapest and most reliable way to ship containers from the deep-sea port to the inland terminal. Second, the most efficient deployment of cranes at container terminals during idle time is studied. In this dissertation, efficient algorithms are developed to solve these problems.

1.1 Multimodal transportation

A term that is closely related to hinterland container transportation is *multimodal transportation*. It is a rather general term as it contains all types of transportation in which two or more modalities are used. For multimodal transportation, the use of a container is not a strict requirement, but as stated before, a container makes the transshipment from one node to another much more efficient. A specific variant of multimodal transportation in which the goods need to be in the same transportation unit during the entire trip is *intermodal transportation* (SteadieSeifi et al., 2014). The most commonly used transportation unit is the shipping container, but a standard letter that is

shipped using more than one modality is also an example of intermodal transport (Crainic and Kim, 2007). Another variant of multimodal transportation is *synchromodal transportation*. In synchromodal transportation, it is explicitly allowed that the planned modality of a container can change if it is already in transit (Dong et al., 2018). The main idea of synchromodal transport is to use real-time information to select the best mode under the current circumstances (SteadieSeifi et al., 2014). For instance, in synchromodal transportation, one can adapt to an accident on the railway by shipping more containers by truck and fewer by train. For overviews of these different types of transportation, we refer to Caris et al. (2014), SteadieSeifi et al. (2014), and Van Riessen et al. (2015). These three different definitions have significant overlap, and all are relevant to describe the problems arising in hinterland container transportation.

A container has three possible modes for inland transportation, namely *barges*, *trains*, and *trucks*. The fastest way to ship a container is with a truck. However, the disadvantage of trucks is that they can transport at most two TEU. On the other hand, trains and barges can ship around 100 to 250 TEU. Consequently, barge and train transportation is much cheaper because a crew consisting of a couple of people can ship the number of containers for which hundred truck drivers are needed. In this dissertation, we do not consider the option of train transportation, and only focus on barge and truck transportation. However, the mathematical models and algorithms developed in this dissertation can easily be extended to include train transportation.

Besides the fact that barge transportation is cheaper than truck transportation, there are two more advantages of barges. First of all, the CO_2-emission is lower for barges than for trucks (Den Boer et al., 2011). Second, deep-sea ports are often located in urban areas, and trucks contribute to road congestion. For instance, 40 percent of the vehicles visiting the port of Rotterdam is delayed (Behdani et al., 2016). Therefore, both the European Commission and the Port of Rotterdam stated in 2011 the ambition to achieve a modal shift from trucks to trains and barges (European Commission, 2011; Port of Rotterdam Authority, 2011). The aim of the European Commission is that by 2030, 30 percent of the long-distance road freight is shifted to trains and barges. The Port of Rotterdam has a similar ambition for 2030 (European Commission, 2011). Their goal is to reduce, compared with 2009, the share of road transportation from 45 to 35 percent and increase the percentage of containers transported by barge and train, respectively, from 40 to 45 percent and from 15 to 20 percent (Port of Rotterdam Authority, 2011).

However, this modal shift has not yet taken place. In the last decade, the share of inland waterway, train, and truck transport all have remained constant in the European Union (Eurostat, 2020). A major drawback of barge and train transportation is that the planning process is much more challenging than

for truck transportation. First of all, barges and trains have longer transit times than trucks, and as a result, the transportation plan needs to be made earlier. It often happens that not all the required information is available when the planning for barges or trains needs to be made. Moreover, only if the number of shipped containers is large enough, the economies of scale of barges and trains ensure that these modes of transportation are cheaper than truck transportation. Finally, the number of containers that can be loaded on a barge is not fixed but depends on the water level of a river. On the one hand, if the water level is high, the containers cannot be stacked high because the barge cannot sail underneath bridges. On the other hand, if the water level is low, the containers on a barge should not be too heavy because otherwise, the barge will hit the bottom of the river (Caris et al., 2014).

Another disadvantage of barges is that deep-sea terminals focus mainly on serving deep-sea vessels. The terminals give priority to deep-sea vessels' service because the costs of a delay for these vessels are much higher than the costs if a barge is delayed. Furthermore, sea shipping companies are big multinational firms that have better negotiating power than small barge operators and thus, have better service contracts (Caris et al., 2012). Consequently, if a deep-sea vessel is delayed, for instance, because of bad weather conditions or a late departure in the previously visited port, the service of the barges is also affected. To make things worse, if a barge is delayed at one terminal, it might also miss the next terminal's time window. The overall result is that the waiting times for barges are large in deep-sea ports, such as Antwerp (Caris et al., 2012) and Rotterdam (Port of Rotterdam Authority, 2020b). These waiting times make barge transportation less reliable and more expensive.

In this dissertation, we formulate three operational multimodal transportation problems encountered by inland terminals and shippers in practice as a mathematical optimization model. In each of these three problems, the goal is to balance the transportation plan's reliability with the costs. Furthermore, we propose efficient algorithms to solve these problems. Solving the right planning problems in real-time can help to overcome the earlier mentioned disadvantages of barge transportation. Therefore, these algorithms can contribute to the desired modal shift.

1.2 Terminal operations

At terminals, containers are unloaded from one mode of transportation and loaded to another. Many different operational challenges arise in this process. These operational problems can be divided into three categories: *ship planning problems*, *transportation problems*, and *storage and stacking problems* (Steenken et al., 2004). Ship planning problems are concerned with the

loading and unloading of a ship. The goal of these problems is to reduce the unproductive time that a ship is in a port. Examples of ship planning problems are the allocation of a ship to a berth and the assignments of cranes to parts of the ship to load and unload the containers (Kim and Günther, 2007). Transportation problems deal with finding the right container sequence to load a ship and ensuring that vehicles deliver containers to the crane. In stacking problems, the storage of containers at a container terminal is studied. The goal of both transportation and stacking problems is to enhance the productivity of the crane. In the remainder of this dissertation, we focus on stacking problems, so for an overview of the other operational problems in container terminals, we refer to Steenken et al. (2004), Stahlbock and Voß (2008), and Vis and De Koster (2003).

Storage and stacking problems would not exist if the different modes of transportation were perfectly synchronized. In case a truck would be ready when the container is unloaded from the ship, the container is not stored at the terminal. However, there is usually time between the arrival and departure of a container at a terminal. In general, a barge is sailing only once or twice a week from a deep-sea terminal to an inland terminal. Hence, there could be multiple days between the container's arrival at a deep-sea port and the moment it is picked up by a barge. Moreover, this problem arises also in inland terminals. The container might arrive by barge multiple days before it needs to be delivered by truck to the customer.

In the time between the arrival at and departure from a terminal, the container is stored in a storage yard. The space in container terminals is limited, and thus, containers are stacked on top of each other to save space. The standardized dimension of containers makes this fairly easy. However, the equipment used in container terminals can only access the top container of a stack. Therefore, if a container has other containers stacked on top of it at the moment it needs to leave the storage yard, then these containers need to be re-positioned. Hence, ideally, a container is on top of a stack at its retrieval moment. This position might be possible if we know the exact time a container will leave the terminal. Nevertheless, when a container arrives at a port, the departure time is often unknown. Consequently, it is unavoidable that extra re-positioning of a container is needed.

In the scientific literature, four different approaches exist that are concerned with the re-positioning of containers (Carlo et al., 2014). The first type of problem is the *storage assignment problem*. This problem deals with finding the best location in a yard for a container that is unloaded. In this type of problem, the goal is to minimize the number of future relocation moves. The objective of the second problem, the *container relocation problem*, is the same. The key difference is that in the container relocation problem, the container

for which we decide on the new location is already located in the storage yard. However, this container is stored on top of a container that needs to be retrieved from the yard. Hence, the blocking container needs to be relocated to another stack (Carlo et al., 2014).

The third and fourth research directions, the *pre-marshalling problem* and the *re-marshalling problem* are, as their names already suggest, similar. In these problems, it is exploited that a few hours before containers will leave the terminal, the sequence in which they need to be retrieved is known. Hence, the containers can be reshuffled such that each container is on top of a stack at its retrieval moment. The difference between the two problems is that in the re-marshalling problem, the containers are positioned in an empty storage area, whereas in the pre-marshalling the containers are reshuffled in their current storage area (Carlo et al., 2014).

In this dissertation, we introduce a new type of problem, which is a combination of the pre-marshalling and the container relocation problem. The movement of containers in the pre-marshalling problem has the advantage that it can be performed at times the workload at the terminal is lower. However, to reshuffle the containers so that all containers are in the correct position requires more moves than needed in the container relocation problem. Nevertheless, the moves in the container relocation problem are performed when there is already a truck or ship waiting. Therefore, if a container is not on top of the stack at the moment it is picked up, the truck or ship has to wait longer. If the turnaround time of vessels and trucks is decreased, they can ship more containers, which will eventually decrease the transportation costs. The benefits of the pre-marshalling problem and the container relocation problem are combined using the concept of *pre-processing moves*. Similar to the pre-marshalling moves, these pre-processing moves are performed before retrieving containers from the storage yard. However, we do not require that all containers are in the correct position after the pre-processing moves. As a result, fewer moves are needed than in the pre-marshalling problem. On top of that, more moves than in the container relocation are executed in relatively idle periods.

1.3 Overview of this dissertation

This dissertation contains three chapters in which operational problems in multimodal transportation are discussed. Moreover, another three chapters are devoted to the concept of pre-processing moves in container terminals. Finally, one chapter deals with approximation algorithms for problems that, on a high-level, reflect the problems of the previous chapters.

In Chapter 2, we present a problem encountered by the inland terminal we collaborated with. In this problem, a set of containers is located at different

deep-sea terminals and needs to be transported to an inland terminal. The objective is to assign these containers to the available barges and trucks such that the costs are minimized. The costs do not only consist of the actual transportation costs, but also the storage costs at both the deep-sea and inland terminal are taken into account. A barge that visits more terminals is less reliable, and thus, we penalize every terminal visit. We formulate an Integer Linear Program (ILP) to solve this problem. However, as the problem is NP-hard, the running time for realistic instances is too large. Therefore, we also develop a heuristic that solves the ILP in two-phases. The running time of the heuristic is only a couple of seconds and gives solutions extremely close to optimal. We compare the solutions of the ILP and the heuristic with an algorithm that mimics the planner who currently does the planning by hand at the inland terminal. The results show that the ILP-based heuristic yields significant better solutions than the planner algorithm. The content of this chapter has been published in Zweers et al. (2019).

Chapter 3 focuses on the stochasticity of the barge service at deep-sea terminals. At congested deep-sea ports, such as Rotterdam and Antwerp, the exact number of containers that can be loaded and unloaded at a deep-sea terminal is uncertain at the time the barge planning is made. Hence, more containers might be loaded on a barge than can be unloaded at a terminal. In this chapter, we model this problem as a two-stage stochastic problem with recourse. We use the *Sample Average Approximation* (SAA) method to solve this problem. The SAA method converges to the optimal solution, but the method is not scalable. Therefore, we also give a heuristic based on the optimal solution for a simplified problem. This heuristic computes in a few seconds solutions for larger instances that are only 1% worse than the SAA solution, which needs a couple of hours of computation time. Moreover, the heuristic that we present gives better results than more general heuristics for stochastic assignment problems. This chapter is based on Zweers et al. (2020b).

The focus of Chapter 4 is more on general multimodal transportation and less specific on hinterland container transportation. In this chapter, we consider a shipper concerned with the shipment of a single container through a network. The problem's goal is to find a minimal cost route such that the container arrives before a deadline at its destination. However, in practice, the time it takes to arrive at the destination is stochastic. We include two types of stochasticity: the randomness of the travel times and the possibility of the overbooking of a transportation leg. Due to this stochasticity, each route has an on-time arrival probability, and Pareto-optimal solutions with this probability and the costs of a route are constructed. An optimal algorithm is developed for this problem, but also a heuristic to solve larger networks in short computation time. The heuristic we develop replaces all stochastic variables by a deterministic risk-

measure. Afterward, the problem can be solved as an ILP. The running time of the heuristic is only a fraction of the time needed by the optimal algorithm. Furthermore, the optimality gap of the heuristic is about 2%. This research has been conducted as part of the Lane Analysis and Route Advisor project, which was funded by the Dutch Institute for Advanced Logistics (DINALOG), and is based on Zweers and Van der Mei (2020).

Chapter 5 is an introduction into the next part of this dissertation in which we discuss stacking operations in container yards. In this chapter, we introduce the concept of *pre-processing moves*, and use that to define two new problems: the *Stochastic Container Relocation Problem with Pre-Processing* (SCRPPP) and the *Stochastic Container Relocation Problem with Constrained Pre-Processing* (SCRPCPP). The objective of the SCRPPP is to minimize the weighted sum of the number of pre-processing moves and the expected number of relocation moves. The weight assigned to the pre-processing moves is an input parameter that can take any value between zero and one, whereas the weight of the relocation moves is normalized to one. This problem generalizes the pre-marshalling problem and the container relocation problem. If the weight factor for pre-processing moves is close to zero, the problem is equivalent to the pre-marshalling problem. In contrast, if the weight factor is equal to one, it corresponds to the container relocation problem. In the SCRPCPP, a constraint is set on the number of pre-processing moves, and the goal is to minimize the expected number of relocation moves. In practice, the SCRPCPP can be applied when the time to perform pre-processing moves is limited. In this chapter, we do not give solution methods for those problems, but we show that they are NP-hard, and we derive bounds regarding the maximum number of containers in a yard. Parts of this chapter have appeared in Zweers et al. (2020a) and Zweers et al. (2020c).

In Chapter 6, we give solution methods for the SCRPPP. First, we present a branch-and-bound (B&B) algorithm to solve the SCRPPP to optimality. For this B&B algorithm, we derive a lower bound for the objective function of the SCRPPP. Unfortunately, the SCRPPP is NP-hard, and the running time of the B&B algorithm is too long for larger instances. Hence, we also develop a local search heuristic to solve the SCRPPP. For this local search heuristic, it is crucial to have an estimation for the number of expected relocation moves for a bay. Therefore, in this chapter, we develop a method that estimates based on a few rules the expected number of relocation moves. This chapter is based on Zweers et al. (2020a).

The solution methods for the SCRPCPP are introduced in Chapter 7. We can solve the SCPRCPP to optimality using a similar B&B algorithm as for the SCRPPP, but the upper and lower bound are different for these two problems. Therefore, these bounds will be derived in Chapter 7. Moreover, in this chapter,

we use the B&B algorithm in a heuristic. In this heuristic, the expected number of relocation moves for a bay is not calculated exactly but estimated using the rule-based estimation method introduced in the previous chapter. Although this heuristic is much faster than the optimal algorithm, its computation time is still too long for the larger instance. Hence, another heuristic is developed for the SCRPCPP to produce good solutions in a few seconds for realistic size instances. Where the SCRPPP is only solved for a single row of containers, the SCRPCPP can be naturally extended to multiple rows. Using the algorithms for a single row, we use an ILP to solve the extension to multiple rows of containers. The contributions of this chapter have been presented in Zweers et al. (2020c).

In previous chapters, numerical experiments have been used to compare heuristic methods' performance with the optimal solution. In Chapter 8, we study polynomial-time algorithms that have a theoretical proven approximation guarantee, so-called *approximation algorithms*. The problems studied in Chapters 2, 3, and 7 have in common that they can be seen as packing problems with two levels of capacity constraints. In Chapter 8, we study a generalization of these types of problems. We introduce a new variant of the Maximum Coverage Problem with Knapsack constraints (MCPK) in which the knapsacks are partitioned into clusters. These clusters impose additional capacity constraints to the problem, which results in the Maximum Coverage Problem with Cluster constraints (MCPC). We obtain a $\frac{1}{3}\left(1-\frac{1}{e}\right)$-approximation algorithm based on the Linear Programming (LP) relaxation for this problem. In this algorithm, we reformulate the natural ILP, enabling us to reduce the cluster capacitated problem to a problem with only capacities on a knapsack level. Moreover, we also use *pipage rounding* to round the solution of the LP-relaxation to a partially integral solution. A specific case of the MCPC is the Multiple Knapsack Problem with Cluster constraints (MKPC). We show that the algorithm for the MCPC gives a $\frac{1}{3}$-approximation algorithm for the MKPC. Moreover, if the MKPC instance satisfies a certain property, an *iterative rounding* approach gives a $\frac{1}{2}$-approximation algorithm. Finally, we show that the technique for reducing the cluster capacity to a knapsack level also applies to the Capacitated Facility Location Problem with Cluster constraints (CFLPC). For this problem, we use an existing $(4.562+\varepsilon)$-approximation algorithm for the Capacitated Facility Location Problem and show that the approximation guarantee remains the same if we add cluster constraints. This chapter is based on Schäfer and Zweers (2020).

Finally, in Chapter 9, we conclude this dissertation and present an outlook on further research directions.

2
Optimizing barge utilization

2.1 Introduction

In this chapter, we consider an operational barge planning problem from an inland terminal's perspective. This problem contains detailed attributes to model the operational planning process at an inland container terminal as realistic as possible. A set of containers is located at multiple deep-sea terminals, and they need to be transported to the inland terminal. The objective is to make a cost-efficient and reliable transportation plan. Moreover, this planning should be made in a few seconds to make real-time adjustments possible.

For each container, the inland terminal has to decide on the mode and day of transportation. For the mode of transportation, there are two options: barges and trucks. The inland terminal has contracts with barge operating companies offering a barge service and trucking companies that have trucks available for the transportation of containers. These companies deliver the container to the inland terminal, from which a truck is used to transport the containers to their final destination. However, this last-mile transport is not included in our problem. For an operational planning problem for the delivery of containers by trucks, we refer to Caris and Janssens (2013).

The possible transportation day is limited by the container's arrival day at the deep-sea terminal and the moment it has to be delivered at the customer. Another critical factor in the decision for the day of transportation is the storage costs at the inland and deep-sea terminal. The storage costs differ per day and are not the same for the deep-sea terminal and the inland terminal. Since the storage costs are substantial, optimizing the time at a terminal is an essential aspect of hinterland container logistics (Iannone, 2012).

In our problem, we do not consider the route of barges. First of all, the appointments a barge operator needs to make are restrictive and not negotiable, and thus, they indirectly imply the route the barge needs to sail. Moreover, as

---

This chapter is based on B.G. Zweers, S. Bhulai, and R.D. van der Mei. Optimizing barge utilization in hinterland container transportation. *Naval Research Logistics*, 66:253–271, 2019.

pointed out by Fazi et al. (2015), the deep-sea terminals are usually densely clustered. Therefore, a good route is not hard to construct. Nevertheless, mooring at a terminal is a time-consuming process; thus, if a barge moors at a terminal, it is beneficial to load many containers. On top of that, at many terminals, there is a big chance of incurring a delay. At the port of Rotterdam, delays of a few hours are not uncommon for barges. Hence, we require that terminals are visited as rarely as possible to reduce the delay of a barge.

Besides making a barge planning that does not visit many terminals, the barges' capacity should also be used as much as possible. If there are more containers on a barge, fixed costs such as the skipper's wage can be divided over more containers. Consequently, the cost per container will decrease. Therefore, one part of our barge planning objective is to minimize the empty container spots of the barges. The combination of maximizing the number of containers shipped by barge, minimizing the storage costs, and minimizing the number of terminals visited by barge is a complicated and non-trivial problem. Sometimes, it might be better to have more storage costs and to visit fewer terminals or to ship more containers by barge. Making a transportation plan that deals with all these aspects is now done by hand at the inland terminal. However, we will propose an Integer Linear Program (ILP) model to solve this problem.

The contribution of this chapter is threefold. First, we discuss a problem faced by an inland terminal, which has not been studied in the literature before. This problem's goal is to minimize the costs of container hinterland transportation and the number of terminals visited by barge. On the one hand, this problem is richer than problems considering only the containers' assignment to existing services. On the other, it is easier than problems in which the routes are part of the decision. The latter fact makes that the problem can be solved much faster while still being relevant for practice. Second, we present an ILP model that can be used to solve this problem. We test our model on real-life data from an inland terminal based in the Netherlands, and we achieve a cost reduction of about 20% compared to current practice. For large problem instances, the computation time of the ILP might be too long to be used in practice. Hence, our third contribution is that we propose a method that solves the ILP in two steps. This method immensely reduces the computation time of difficult instances while still producing solutions that are extremely close to the optimal solution.

The organization of this chapter is as follows. We start by giving a review of the existing literature for operational problems in container hinterland transportation in Section 2.2. Afterward, we give a detailed description of the problem in Section 2.3. This problem is formulated as a mathematical optimization problem in Section 2.4, and in Section 2.5, we present three methods to solve the problem. In Section 2.6, the three methods are used to solve medium-

sized and large-sized instances, and the results are compared. Finally, we will draw some conclusions and indicate further research directions in Section 2.7.

2.2 Literature review

In the multimodal literature, most attention has been paid to strategic problems, such as network design, but little focus has been on operational problems (Mes and Iacob, 2016). Nevertheless, there are a few recent papers that explicitly deal with operational decision making in multimodal transportation. Based on the decision made, the problems in the existing literature are divided into three categories: (i) problems in which containers are assigned to existing barge services, (ii) problems in which the routes of barges are determined for a given demand, and (iii) problems in which both the assignment of containers to barges and the route of barges are decided upon.

In the first category, the assignment of containers to services can be done offline and online. In an offline planning problem, the assignment of containers to services is done once the information of all containers is known, whereas, in an online problem, the planner needs to decide the service of a container immediately once the booking is made. In Baykasoglu and Subulan (2016), a multi-objective offline planning problem of the loads in an intermodal network is presented. In their model, the transportation costs, the service level, and CO_2-emissions are optimized for both the import and export flow of containers. Another offline problem is discussed in Tierney et al. (2014), in which the inter-terminal transportation in a deep-sea terminal is analyzed. The goal of this problem is not to minimize costs, but to minimize the delay from containers being transshipped within the port. Pérez Rivera and Mes (2016) study an offline problem in which containers have to be assigned to modes of transportation in a synchromodal network in order to minimize the costs. They formulate this problem as a Markov decision process in order to also take into account uncertain future costs. In Heggen et al. (2019), an integrated approach for the assignment of containers to a rail service and the delivery of containers per truck to their final destination is studied. This integrated approach is solved using a large neighborhood search heuristic, and results in cost savings and better utilization of the capacity of the long-haul transportation mode.

In Van Riessen et al. (2016), optimal offline solutions are obtained for an intermodal planning problem in which the objective is to minimize the transportation costs and penalties of being late. These offline solutions are used to infer a decision tree that is then used to make online decisions. Mes and Iacob (2016) also consider an online planning problem. They propose a k-shortest path approach in which the planner receives for each order the k best paths in the network in terms of costs, delays, and CO_2-emissions. Finally, the work of

Wang et al. (2016) incorporates revenue management for barge transportation. For every incoming order, a decision needs to be made whether to accept the order or not and which service to use for the container.

The paper of Li et al. (2016) falls under the second category of the literature. In this paper, a distributed constraint optimization problem is formulated to decide upon the route of vessels in a port. In this problem, it is known for each vessel which terminals it should visit and how many containers to load at these terminals, and the route of the vessel needs to be determined. In Karlaftis et al. (2009), a case study is presented in which containers have to be transported between ports on islands in the Aegean Archipelago and mainland Greece. If a ship is visiting a port, all containers have to be transported, so this problem is formulated as a capacitated vehicle routing problem.

Finally, we discuss three studies that fall into the third category. In Fazi et al. (2015), a decision tool is developed for planning hinterland transportation. Similar to our problem, Fazi et al. (2015) maximize barge utility and penalize a barge visit at a terminal. Nevertheless, they account for sailing time, and the transportation costs are per hour. Therefore, they need to calculate the actual route the barge is sailing. Moreover, they are not considering any storage costs at the deep-sea or inland terminal. The model of Behdani et al. (2016) decides which containers to assign to a service and when that service should leave a terminal. Their goal is to minimize both the transportation costs and the waiting time of containers at terminals. Finally, in the work of Sharyapova (2014), a scheduled service network design is introduced with synchronization and transshipment constraints. The goal here is to minimize the total operational costs by selecting which services to use, determining the timing of the vehicles, and deciding which containers to transport with which service. Since the time of a service is incorporated in the decision making, we have chosen to group this work in the third category and not in the first.

2.3 Problem formulation

Our problem focuses on the transportation of containers from multiple deep-sea terminals to a single inland terminal. Each container arrives with a deep-sea vessel at a deep-sea terminal. If a container is already at the deep-sea terminal, the day of arrival is naturally known. In case a container arrives in the future, only the *estimated time of arrival* (ETA) is known. We will refer in both cases to ETA as the day of arrival. Moreover, for each container, the so-called *call date* is given. The call date represents the day at which the container has to be delivered to the customer. Customers are located in the direct neighborhood of the inland terminal, and thus if the container arrives at the inland terminal exactly at the call date, it can still be shipped on time to the

customer. Therefore, the call date can also be seen as the day the container has to arrive at the inland terminal.

As unloading a large deep-sea vessel may take hours, we assume that the container is available for hinterland transportation a day after the ETA. In our problem, transportation of the containers by barge takes one day, so the day before the call date is the last day that a container can be shipped by barge. As transportation by truck is much faster, the container can still be shipped by truck on the call date. Our approach can easily be adjusted to a situation in which transportation or unloading takes a different time length.

The ETA and the call date of the container impose hard constraints on the possible days of shipments. On top of that, the ideal day of shipment is also influenced by *demurrage costs* and *storage costs*. Demurrage costs are costs paid to the carrier of the container if the container stays too long at the deep-sea terminal. Usually, a container has a certain demurrage free period for which no demurrage costs have to be paid. After that demurrage free period, demurrage costs are paid per day that a container is located at a deep-sea terminal. Storage costs are the costs associated with the number of days the container is located at the inland terminal. Generally, the storage costs per day are much lower than the demurrage costs per day because space at an inland terminal is less scarce than at a deep-sea terminal. In other words, before the demurrage free period has ended, it is cheaper to store the container at the deep-sea terminal than at the inland terminal and vice versa after the demurrage free period has ended. For each day, it is straightforward to calculate the demurrage and storage costs that are incurred by transporting a container on that day. Given these costs, finding the day for which the minimum demurrage and storage costs have to be paid is easy.

The barge operator makes a schedule that specifies which barge is present at the deep-sea port on which day. Therefore, we consider this barge schedule as input for our model. We assume that each barge has only one day on which it can load containers at the deep-sea port. Consequently, if we assign a container to a barge, then the day of transportation is also known. Each barge has a maximum capacity that cannot be exceeded. The size of a container is measured in a Twenty-foot Equivalent Unit (TEU), and so is the maximum capacity of the barge. For each barge, there are fixed costs to use that barge. Moreover, transporting a unit of TEU on a barge has certain costs.

Besides a barge, it is also possible to ship a container by truck. We assume that a truck can only transport a single container, irrespective of the size of a container. As a result, the costs of shipping a container by truck are much higher than the shipping costs by barge. However, the advantage of trucking a container is that from a practical perspective it is reasonable to assume that

every day there is a truck available. Furthermore, we also assume that the number of trucks available for transportation is sufficient to ship all containers by truck if needed. Consequently, the day of truck shipment is the day with the minimum storage and demurrage costs. As a result, the transportation day of a truck does not need to be determined. All in all, for each container, it has to be decided on which barge it is shipped or if it is shipped by truck.

We consider a planning problem with a finite horizon. In determining the length of the planning horizon, a trade-off has to be made between reliable information and planning flexibility. If a short planning horizon is chosen, the information of the containers is rather reliable and unlikely to be subject to changes. On the other hand, only a few barges might be available for the transportation of the containers. Hence, the assignment possibilities for each container are restrictive. In case a long planning horizon is chosen, the flexibility of assigning the containers to barges increases, but the available information becomes unreliable.

A consequence of the finite planning horizon is that some containers have a call date after the end of the planning horizon. We will call these containers *low-priority* containers. Containers that need to be shipped during the planning period will be called *high-priority* containers. In Section 2.4, a more formal definition of low and high-priority containers will be given. When time progresses, each low-priority container will eventually become a high-priority container. These low-priority containers do not necessarily need to be transported in the planning period. Two standard approaches for dealing with low-priority containers are either ignoring them for the current planning period or forcing them to be transported anyway. These approaches work fine if the number of arriving containers at the deep-sea port and the available barge capacity is almost constant for each day. However, in practice, these numbers are far from constant over the days.

To illustrate why these two methods might fail, consider a situation in which a large batch of containers is available for transportation on a specific day, but the call date of these containers is a day after the end of the planning horizon. If there is a barge in the planning period with some unused capacity and the low-priority containers are ignored, we face the risk that there is insufficient barge capacity after the current planning period and we need to transport (part of) the batch per truck, whereas it was possible to ship at least part of the batch per barge. On the other hand, if there is not enough capacity on the barges in the current planning period and one has decided that low-priority containers have to be transported, some containers will be transported by truck in the current planning period. However, it might be possible that there is a large amount of barge capacity available on the day after the end of our planning horizon. So in hindsight, it would have been possible to ship our batch of containers by barge.

Ideally, we would like to ship the low-priority containers only if there is capacity left on the barges in the current planning period and wait with transportation of the low-priority containers if no barge capacity is left. Therefore, it is not possible to take minimizing the transportation costs as an objective because not transporting low-priority containers is always cheaper than transporting the low-priority containers. To ensure that low-priority containers also have the possibility of being transported, we have chosen to maximize the containers transported by barge as the objective, instead of minimizing the transportation costs. Since transportation by barge is much cheaper than by truck, a transportation plan in which the number of containers shipped per barge is maximal is also likely to be a plan in which the transportation costs are low.

The containers are located at multiple deep-sea terminals, and in order to visit as few terminals as possible by barge, we impose penalty costs for each visit. The penalty costs result in a situation in which a terminal is only visited if there are enough containers that can benefit from the fact that they do not have to be transported by truck. Achieving a situation in which there is a large set of containers available for transportation might require that some containers have to be shipped on another day than their ideal shipment day.

Instead of penalizing a terminal visit by barge, it is also possible to impose a constraint on the number of terminals visited per barge. The reason why we have not chosen this option is twofold. First, if visiting one more terminal than the 'maximum' number of terminal visits reduces the transportation costs by a large amount, we would like to visit that terminal anyway. Second, if there are two schedules with the same total transportation costs and one is visiting fewer terminals than the other, the schedule with fewer terminal visits is preferred. In summary, a trade-off has to be made between the transportation costs, the demurrage costs, the storage costs, and the number of terminals that are visited by barge.

Running example

Throughout this paper, the same problem instance will be used as an example to illustrate the problem. First, in Figure 2.1, we explain the different characteristics of containers and afterward in Figures 2.2 and 2.3, we illustrate how containers can be assigned to barges and trucks. In Figure 2.1, an example of possible classes of containers at a deep-sea terminal is given. In this example, there are ten containers, which all have a size of two TEU, and the planning horizon is three days. A container can only be transported on a day if the rectangle representing the container is in the column representing that day. For instance, container 2 can be transported on all three days, but container 10 only on day 3. As container 2 is available for transportation on day 1 and needs to be transported at the latest on day 3, it means that its ETA is day 0 and its

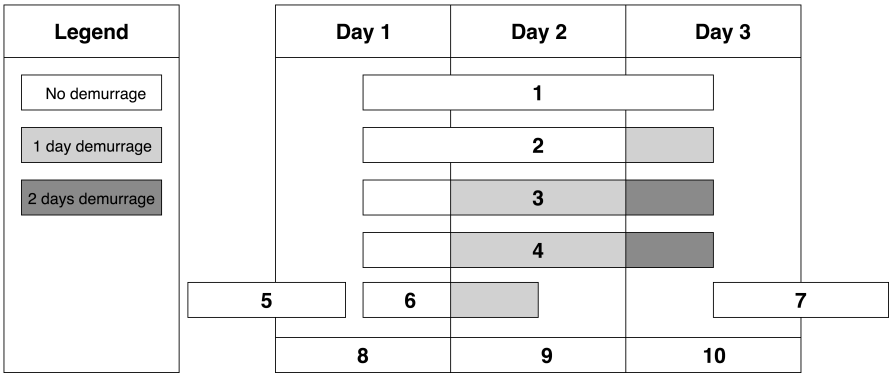


Figure 2.1 Example of different container classes at a deep-sea terminal.

call date is day 4.

The fact that container 2 is light grey on day 3 means that demurrage costs have to be paid if it is transported on that day. Containers 3 and 4 are colored dark grey on day 3 because demurrage costs for two days have to be paid when they are shipped on day 3. Containers 1, 2, 3, and 4 are rather flexible because we can ship them on any of the three days, whereas containers 8, 9, and 10 can only be transported on one specific day. Besides, container 5 can also only be transported on day 1 because it was not assigned to a truck or barge before day 1. Container 7 is a low-priority container because its call date is after the end of the planning period.

In the running example, we will consider a situation with two terminals: terminal R and S. The ten containers, that are given in Figure 2.1, are located at both terminal R and S. Moreover, barges are available on day 1 and day 3, and both of these barges have a maximum capacity of 15 TEU. As there are only containers of 2 TEU, each barge can ship at most seven containers. The fixed costs of using the barges are set low enough to ensure that both barges will always be used. Container 9 from Figure 2.1 is ignored in the remainder of the running example because it can never be transported by a barge. In Figures 2.2 and 2.3, two different examples are given of how the containers can be allocated to barges and trucks.

At the top of Figure 2.2, the available containers are shown for each terminal for each day. On day 1 in Figure 2.2, barge 1 is visiting terminal R, as indicated by the arrow pointing from barge 1 to the box with all available containers at terminal R on day 1. The available containers that are transported by a barge are indicated with a circle within the square of the container. So in Figure 2.2, all containers at terminal R that are available for transportation are transported by barge 1. At terminal S on day 1 in Figure 2.2, there are containers with

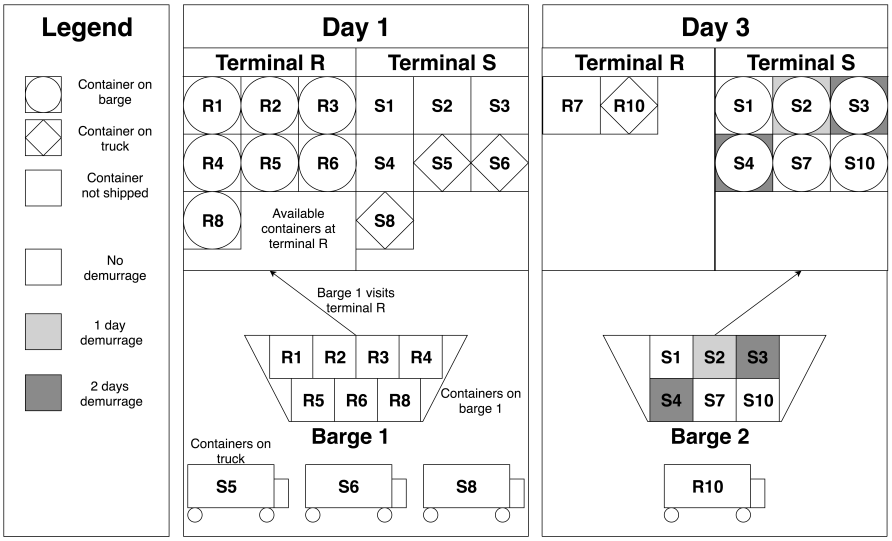


Figure 2.2 Example of container assignment with one terminal visit per barge.

a diamond inside their box. These containers, namely S5, S6, and S8, will be transported by a truck. We need to ship these containers with a truck because on day 3, when the next barge is available, they cannot be transported anymore. If there is neither a circle nor a diamond inside the square of a container, it means that a container is not transported on that day. For example, containers S1, S2, S3, and S4 are not transported on day 1 in Figure 2.2. On day 3, in Figure 2.2, container S2 has a light grey box and containers S3 and S4 have a dark grey box. Similar to Figure 2.1, the light grey box represents a container for which one day of demurrage costs have to be paid, and the dark grey box represents a container for which two days of demurrage costs have to be paid.

In contrast to visiting only one terminal, it is also possible that a barge visits two terminals, as is illustrated in Figure 2.3. If we compare the situations of Figures 2.2 and 2.3, we see that in Figure 2.2 four containers are shipped per truck and in Figure 2.3 only two. Besides, in Figure 2.3 seven containers are shipped on the barge of day 3 and in Figure 2.2 only six. Moreover, in the situation illustrated in Figure 2.2, there are in total five days with demurrage costs and in Figure 2.3 only four. All in all, the barges in Figure 2.2 are visiting fewer terminals than in Figure 2.3, but in Figure 2.2 more demurrage days occur, fewer containers are shipped per barge and more per truck. It depends on how severe a visit of a terminal is penalized if the situation of Figure 2.2 or 2.3 is preferred.

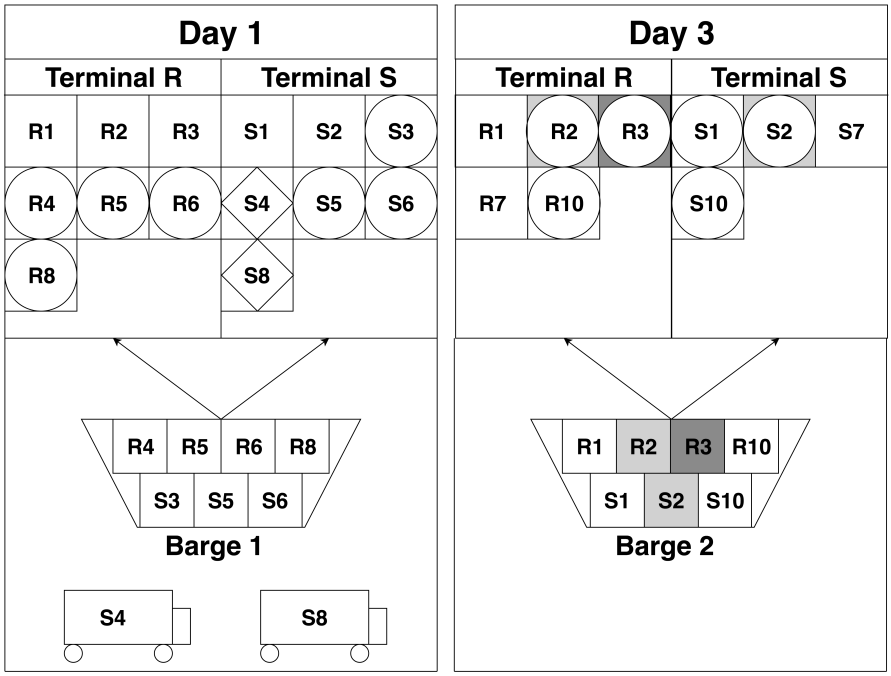


Figure 2.3 Example of container assignment with two terminal visits per barge.

2.4 Mathematical model

In Section 2.4.1, the problem described in the previous section is formulated as a mathematical optimization model. In this model, the penalty that needs to be paid for a terminal visit is an important input parameter. In Section 2.4.2, we present three properties of the optimal solution depending on the value of this penalty.

2.4.1 Optimization model

First of all, some notation is introduced. The set of containers available for transportation is given by $\mathcal{C}$ and consists of n containers. These containers are located at R different deep-sea terminals. A container i arrives at day a_i at the deep-sea terminal m_i. Moreover, the call date of container i is b_i, and its size is t_i TEU. We denote the maximum size of all n containers as T, i.e., $T = \max_{i=1,\dots,n} t_i$.

The last day of the planning horizon is given by τ, and b barges are available in the entire planning period. The set of these barges is given by $\mathcal{B}$. A barge v is at the deep-sea port at day d_v and has a capacity of u_v TEU. Using barge v

c^T	Shipping costs per truck
c_v^B	Shipping costs for one unit of TEU on barge v
c_{iv}^D	Demurrage costs for container i if it is transported by vehicle v
c_{iv}^S	Storage costs for container i if it is transported by vehicle v
π_{rv}	Penalty costs if barge v is visiting terminal r
ρ_v	Fixed costs for barge v
u_v	Maximum capacity of barge v in TEU
d_v	Day that barge v is at the deep-sea port
t_i	Size in TEU of container i
a_i	Estimated arrival date of container i at the deep-sea terminal
b_i	Call date of container i
m_i	Deep-sea terminal where container i is located
τ	Last day of the planning horizon
T	Largest size in TEU of a container, i.e., $T = \max_{i=1,\ldots,n} t_i$
$\mathcal{B}$	Set of all available barges
$\mathcal{C}$	Set of all containers
$\mathcal{H}$	Set of all high-priority containers
$\mathcal{L}$	Set of all low-priority containers

Table 2.1 Notation for the input parameters used in Chapter 2.

imposes fixed costs of ρ_v. Moreover, the visit of terminal r by barge v results in a penalty of π_{rv}. Each container could be assigned to b barges or a truck, and thus, there are $b+1$ vehicles: $v = 0$ is a truck, and $v = 1, \ldots, b$ are the barges. Let the barges be numbered in increasing order of their day at the deep-sea port. In other words, barge 1 is the first barge to be in the deep-sea port and barge b the last. The demurrage and storage costs for container i on vehicle v are given by, respectively, c_{iv}^D and c_{iv}^S. Finally, the transportation costs for a container on a truck are given by c^T, and c_v^B denotes the shipping costs of one unit of TEU on barge v. The notation of input data is summarized in Table 2.1.

In Section 2.3, high-priority containers were defined as containers that had to be transported in the planning period. Now, a more formal definition will be given. Two criteria determine the priority of a container. First, if the call date of a container is before the end of the planning period, it is a high-priority container. Second, if the total costs of shipping the container on the last barge of the planning period are higher than the total costs of shipping a container by truck, the container's priority is also high.

The second criterion is based on the fact that the demurrage costs are higher than the storage costs at the inland terminal. So after the end of the demurrage free period, the total storage and demurrage costs will increase each

day. Hence, if it is cheaper to transport a container per truck in the current planning period than on the last barge, it is cheaper to transport the container per truck in the current planning period than on a barge in the next planning period.

Using the notation just introduced, the two criteria will be formalized in Definition 2.1 below. If a container is not a high-priority container, it is automatically a low-priority container, i.e., $\mathcal{L} = \mathcal{C} \setminus \mathcal{H}$.

Definition 2.1 (High-priority container)**.** *Container $i \in \mathcal{C}$ is in the set of high-priority containers $\mathcal{H}$, if at least one of the two inequalities holds:*

- $b_i \leq \tau$
- $c_b^B + c_{ib}^D + c_{ib}^S \geq c^T + c_{i0}^D + c_{i0}^S$.

Let us assume we have an assignment σ that assigns all containers to a barge, a truck, or no transportation mode. Let us denote the set of containers that are assigned to vehicle $v = 0, 1, \ldots, b$ in this assignment σ by $\mathcal{C}_\sigma(v)$. Furthermore, let $\mathcal{B}_\sigma$ denote the set of barges that is used, and the set of terminals visited by barge v in assignment σ is given $\mathcal{R}_\sigma(v)$. An assignment σ is feasible if it satisfies the following conditions:

- $\mathcal{H} \subseteq \cup_{v=0}^{b} \mathcal{C}_\sigma(v) \subseteq \mathcal{C}$
- $\sum_{i \in \mathcal{C}_\sigma(v)} t_i \leq u_v$ for $v \in \mathcal{B}_\sigma$
- $a_i < d_v < b_i$ for $i \in \mathcal{C}_\sigma(v)$ and $v \in \mathcal{B}_\sigma$.

The first condition requires that each high-priority container is transported. The second condition ensures that the number of TEU shipped on a barge does not exceed the barge's capacity. Finally, it is required that a container is only transported on a barge that is at the deep-sea port one day after the container's ETA and a day before its call-date. The objective of the problem is to find a feasible assignment σ that minimizes the expression

$$\begin{aligned} &\sum_{v \in \mathcal{B}_\sigma} \left(\rho_v + \sum_{r \in \mathcal{R}_\sigma(v)} \pi_{rv} + c_v^B \left(u_v - \sum_{i \in \mathcal{C}_\sigma(v)} t_i \right) \right) + c^T |C_\sigma(0)| \\ &+ \sum_{v=0}^{b} \sum_{i \in \mathcal{C}_\sigma(v)} c_{iv}^D + c_{iv}^S. \end{aligned} \tag{2.1}$$

The objective consists of three parts, namely the costs associated with barge transportation, the truck transportation costs, and the demurrage and storage

costs. The barge transportation costs itself also consists of three sums. In the first sum, the fixed costs for barge usage are minimized, and the penalties for visiting a terminal are taken into account in the second sum. The expression $u_v - \sum_{i \in \mathcal{C}_\sigma(v)} t_i$ corresponds to the unused capacity of barge v. By imposing the costs of c_v^B for every unit of TEU not being transported on barge v, the utilization of the capacity of barge v is maximized.

2.4.2 Bounds on properties of the optimal solution

The penalty π_{rv} for visiting terminal r by barge v is an artificial penalty. Hence, it could be set to any value. The value of π_{rv} indirectly imposes certain bounds on the optimal solution. In Lemmas 2.1, 2.2, and 2.3 below, we show the relation between the value of π_{rv} and respectively, the minimum number of containers needed to visit terminal r by barge v, the maximum number of terminals visited by barge v and the minimum number of TEU on barge v.

Lemma 2.1. *If $\pi_{rv} \geq \lambda(c^T + T c_v^B)$ and barge v is visiting terminal r, then at least λ containers from terminal r are loaded on barge v.*

Proof. For the sake of contradiction, let $\mathcal{K}$ be a set consisting of $k < \lambda$ containers from terminal r, which are loaded on barge v in the optimal solution. The total costs of shipping the containers in set $\mathcal{K}$ are equal to:

$$\pi_{rv} + \sum_{i \in \mathcal{K}} \left(c_{iv}^D + c_{iv}^S - t_i c_v^B \right). \tag{2.2}$$

Using the fact that the size of a container is at most T TEU ($t_i \leq T$), we can derive the following lower bound on the costs of shipping the k containers from set $\mathcal{K}$ with barge v:

$$\begin{aligned} \pi_{rv} + \sum_{i \in \mathcal{K}} \left(c_{iv}^D + c_{iv}^S - t_i c_v^B \right) &\geq \pi_{rv} + \sum_{i \in \mathcal{K}} \left(c_{iv}^D + c_{iv}^S - T c_v^B \right) \\ &= \pi_{rv} - kT c_v^B + \sum_{i \in \mathcal{K}} \left(c_{iv}^D + c_{iv}^S \right). \end{aligned} \tag{2.3}$$

The fact that each container could be transported by a truck is used below for an upper bound for (2.2).

$$\begin{aligned} \pi_{rv} + \sum_{i \in \mathcal{K}} \left(c_{iv}^D + c_{iv}^S - t_i c_v^B \right) &\leq \sum_{i \in \mathcal{K}} \left(c_{i0}^D + c_{i0}^S + c^T \right) \\ &\leq \sum_{i \in \mathcal{K}} \left(c_{iv}^D + c_{iv}^S + c^T \right) \\ &= k c^T + \sum_{i \in \mathcal{K}} \left(c_{iv}^D + c_{iv}^S \right). \end{aligned} \tag{2.4}$$

The first inequality uses the fact that the total costs for trucking the containers in the set $\mathcal{K}$ are an upper bound for the optimal costs of transporting the containers in the set $\mathcal{K}$. The second inequality follows from the property that container i can be transported by a truck on any day, so we know that the sum of the demurrage and storage costs when transporting the container by truck are not larger than when the container is shipped on barge v: $c_{i0}^D + c_{i0}^S \leq c_{iv}^D + c_{iv}^S$. Combining the lower bound from (2.3) and the upper bound from (2.4) leads to the following inequality:

$$\pi_{rv} - kTc_v^B + \sum_{i \in \mathcal{K}} \left(c_{iv}^D + c_{iv}^S\right) \leq kc^T + \sum_{i \in \mathcal{K}} \left(c_{iv}^D + c_{iv}^S\right),$$

which implies that

$$\pi_{rv} - kTc_v^B \leq kc^T.$$

Since $k < \lambda$, this leads to a contradiction of the assumption of this lemma:

$$\pi_{rv} \leq k(c^T + Tc_v^B) < \lambda(c^T + Tc_v^B).$$

□

Lemma 2.2. *If $\pi_{rv} \geq \frac{u_v\left(c_v^B + c^T\right) - \rho_v}{\lambda}$ for all terminals r and barge v, then barge v will not visit more than λ terminals.*

Proof. Let the set of terminals visited by barge v and the set of containers transported on barge v be denoted as $\mathcal{R}$ and $\mathcal{K}$, respectively. For the sake of contradiction, assume that barge v is visiting $m = |\mathcal{R}| > \lambda$ terminals. The costs of shipping the containers from $\mathcal{K}$ on barge v are equal to

$$\sum_{r \in \mathcal{R}} \pi_{rv} + \sum_{i \in \mathcal{K}} \left(c_{iv}^D + c_{iv}^S - t_i c_v^B\right) + \rho_v.$$

Firstly, a lower bound on these total costs will be derived:

$$\begin{aligned}
&\sum_{r \in \mathcal{R}} \pi_{rv} + \sum_{i \in \mathcal{K}} \left(c_{iv}^D + c_{iv}^S - t_i c_v^B\right) + \rho_v \\
\geq\ &\sum_{r \in \mathcal{R}} \pi_{rv} - u_v c_v^B + \sum_{i \in \mathcal{K}} \left(c_{iv}^D + c_{iv}^S\right) + \rho_v \\
\geq\ &\sum_{r \in \mathcal{R}} \pi_{rv} - u_v c_v^B + \sum_{i \in \mathcal{K}} \left(c_{i0}^D + c_{i0}^S\right) + \rho_v \\
\geq\ &\frac{m}{\lambda}\left(u_v\left(c_v^B + c^T\right) - \rho_v\right) - u_v c_v^B + \sum_{i \in \mathcal{K}} \left(c_{i0}^D + c_{i0}^S\right) + \rho_v.
\end{aligned} \tag{2.5}$$

The first inequality follows from the fact that the total number of TEU assigned to a barge can never exceed the capacity. The second inequality holds because the storage and demurrage costs for shipping a container by truck are always lower than shipping a container by barge. The final inequality is a direct consequence of the assumption of this lemma. Similar to Lemma 2.1, an upper bound can be derived using the costs of transporting the containers per truck:

$$\begin{aligned}\sum_{r\in\mathcal{R}}\pi_{rv}+\sum_{i\in\mathcal{K}}\left(c^D_{iv}+c^S_{iv}-t_i c^B_v\right)+\rho_v &\leq \sum_{i\in\mathcal{K}}\left(c^D_{i0}+c^S_{i0}+c^T\right)\\ &= |\mathcal{K}|c^T+\sum_{i\in\mathcal{K}}\left(c^D_{i0}+c^S_{i0}\right)\\ &\leq u_v c^T+\sum_{i\in\mathcal{K}}\left(c^D_{i0}+c^S_{i0}\right).\end{aligned} \tag{2.6}$$

The final inequality follows from the fact that the smallest size of a container is 1 TEU, so the cardinality of the set $\mathcal{K}$ is at most u_v. Combining the lower bound from (2.5) and the upper bound from (2.6), we get the relation

$$\frac{m}{\lambda}\left(u_v\left(c^B_v+c^T\right)-\rho_v\right)-u_v c^B_v+\sum_{i\in\mathcal{K}}\left(c^D_{i0}+c^S_{i0}\right)+\rho_v\leq u_v c^T+\sum_{i\in\mathcal{K}}\left(c^D_{i0}+c^S_{i0}\right),$$

which implies that

$$\frac{m}{\lambda}\left(u_v\left(c^B_v+c^T\right)-\rho_v\right)-u_v c^B_v+\rho_v\leq u_v c^T.$$

This inequality can only hold if $\frac{m}{\lambda}\leq 1$, which is a contradiction to our assumption that $m>\lambda$. □

Lemma 2.3. *If $\pi_{rv}\geq \alpha u_v(c^T+c^B_v)-\rho_v$ for every terminal r for barge v and $0\leq\alpha\leq 1$, then barge v is filled with at least αu_v if barge v is used for transportation.*

Proof. Let the set of terminals visited by barge v and the set of containers transported on barge v be denoted as $\mathcal{R}$ and $\mathcal{K}$, respectively. For the sake of contradiction, assume that barge v is filled with βu_v TEU with $\beta<\alpha$. The costs of shipping the containers from $\mathcal{K}$ on barge v is equal to

$$\sum_{r\in\mathcal{R}}\pi_{rv}+\sum_{i\in\mathcal{K}}\left(c^D_{iv}+c^S_{iv}-t_i c^B_v\right)+\rho_v.$$

Similar to the proofs of Lemmas 2.1 and 2.2, a lower bound and upper bound

for these costs will be derived. The lower bound is equal to

$$\begin{aligned} &\sum_{r\in\mathcal{R}} \pi_{rv} + \sum_{i\in\mathcal{K}} \left(c_{iv}^{D} + c_{iv}^{S} - t_i c_v^{B}\right) + \rho_v \\ &\geq \pi_{rv} + \sum_{i\in\mathcal{K}} \left(c_{iv}^{D} + c_{iv}^{S} - t_i c_v^{B}\right) + \rho_v \\ &\geq \pi_{rv} - \beta u_v c_v^{B} + \sum_{i\in\mathcal{K}} \left(c_{i0}^{D} + c_{i0}^{S}\right) + \rho_v. \end{aligned} \tag{2.7}$$

The final inequality uses the fact that we know that barge v is filled with $0 \leq \beta u_v$ TEU. The upper bound uses again the fact that all containers could be shipped per truck and is as follows:

$$\begin{aligned} \sum_{r\in\mathcal{R}} \pi_{rv} + \sum_{i\in\mathcal{K}} \left(c_{iv}^{D} + c_{iv}^{S} - t_i c_v^{B}\right) + \rho_v &\leq \sum_{i\in\mathcal{K}} \left(c_{i0}^{D} + c_{i0}^{S} + c^{T}\right) \\ &\leq \beta u_v c^{T} + \sum_{i\in\mathcal{K}} \left(c_{i0}^{D} + c_{i0}^{S}\right). \end{aligned} \tag{2.8}$$

Combining the lower and upper bound from (2.7) and (2.8), we get

$$\pi_{rv} - \beta u_v c_v^{B} + \sum_{i\in\mathcal{K}} \left(c_{i0}^{D} + c_{i0}^{S}\right) + \rho_v \leq \beta u_v c^{T} + \sum_{i\in\mathcal{K}} \left(c_{i0}^{D} + c_{i0}^{S}\right),$$

which implies that

$$\pi_{rv} - \beta u_v c_v^{B} + \rho_v \leq \beta u_v c^{T},$$

and that leads to

$$\pi_{rv} \leq \beta u_v (c^{T} + c_v^{B}) - \rho_v < \alpha u_v (c^{T} + c_v^{B}) - \rho_v,$$

which is a contradiction to the assumption of this lemma. □

The conclusion from these three lemmas is that the value π_{rv} indirectly imposes constraints on the optimal solution. Which of the bounds provided by the three lemmas is the tightest depends on the parameters used. The bounds from Lemmas 2.2 and 2.3 are tight if the fixed costs of using a barge are high. However, in the upper bounds in the proof of both these lemmas all containers that were originally assigned to the barge are transported per truck, which could be a weak upper bound. In the upper bound of Lemma 2.1, a smaller number of containers are transported per truck.

2.5 Solution methods

To solve the problem discussed in Section 2.4.1, we present three different algorithms in this section. First of all, we present an optimal ILP formulation in Section 2.5.1. Afterward, in Section 2.5.2 a two-stage heuristic that is based on the ILP formulation from Section 2.5.1 is formulated. Finally, we present in Section 2.5.3 an algorithm that mimics the behavior of experienced planners who plan the containers to barges manually.

2.5.1 ILP formulation

In the ILP formulation, three types of binary decision variables are used: X, Y, and Z. The variable X_{iv} indicates whether container i is transported by vehicle v or not. Furthermore, Y_{rv} is equal to 1 if terminal r is visited by barge v, and zero otherwise. Finally, Z_v represents whether barge v is used or not. Using these variables we can define the following ILP:

$$\begin{aligned}\min \sum_{v=1}^{b}\left(c_v^B\left(u_v-\sum_{i=1}^{n}t_iX_{iv}\right)\right)+\sum_{i=1}^{n}c^TX_{i0}+\sum_{v=0}^{b}\sum_{i=1}^{n}(c_{iv}^D+c_{iv}^S)X_{iv}\\ +\sum_{v=1}^{b}\sum_{r=1}^{R}\pi_{rv}Y_{rv}+\sum_{v=1}^{b}\rho_vZ_v\end{aligned} \tag{2.9}$$

subject to

$$\sum_{i=1}^{n}t_iX_{iv}\leq u_v \qquad v=1,\ldots,b \tag{2.10}$$

$$\sum_{v=0}^{b}X_{iv}=1 \qquad i\in\mathcal{H} \tag{2.11}$$

$$\sum_{v=0}^{b}X_{iv}\leq 1 \qquad i\in\mathcal{L} \tag{2.12}$$

$$X_{iv}\leq Y_{rv} \qquad i=1,\ldots,n \qquad v=1,\ldots,b \qquad r=m_i \tag{2.13}$$

$$Y_{rv}\leq Z_v \qquad r=1,\ldots,R \qquad v=1,\ldots,v \tag{2.14}$$

$$X_{iv}=0 \qquad v=1,\ldots,b \qquad i=1,\ldots,n \qquad d_v\leq a_i\vee d_v\geq b_i \tag{2.15}$$

$$X_{iv}\in\{0,1\} \qquad i=1,\ldots,n \qquad v=0,\ldots,b \tag{2.16}$$

$$Y_{rv}\in\{0,1\} \qquad r=1,\ldots,R \qquad v=1,\ldots,b \tag{2.17}$$

$$Z_v\in\{0,1\} \qquad v=1,\ldots,b. \tag{2.18}$$

The objective function in (2.9) consists of five sums. The first sum makes that for each unit of TEU that is not used on barge v $\left(u_v-\sum_{i=1}^{n}t_iX_{iv}\right)$, a penalty of

c_v^B has to be paid. The second sum contains all shipping costs that are made by truck shipment. The third sum is the total of all demurrage and storage costs. Furthermore, the fourth sum contains the penalties that are paid for visiting a terminal by barge. Finally, the fifth sum corresponds to the fixed costs that need to be paid to use the barges. Constraint (2.10) makes that each barge ships at most its maximum capacity in TEU. Constraint (2.11) forces all high-priority containers to be shipped exactly once, and constraint (2.12) ensures that all low-priority containers are shipped at most once. Constraint (2.13) connects the X_{iv} and Y_{rv} variables. If a container is picked up by a barge, that barge also needs to visit the container's terminal. Similarly, constraint (2.14) connects the Y_{rv} and Z_v variables. Terminal r can only be visited by barge v if barge v is used. Constraint (2.15) ensures that a container is not transported before its arrival at the deep-sea port or after its call date. Finally, constraints (2.16), (2.17), and (2.18) are the integrality constraints.

In practice, one might require some additional properties of this model. We will close this section with two features that can easily be included in the ILP formulation.

First of all, it might be desirable to scale the costs of visiting a terminal exponentially with the number of visits. Since the more terminals are visited, the more likely it is that a barge is delayed. It is possible to model the exponentially increasing costs as an ILP. To this end, a new type of binary decision variable is needed, namely, $\Gamma_{vj} \in \{0, 1\}$, which indicates whether barge v visits exactly j terminals or not. Moreover, let δ_{vj} be the total costs for visiting j terminals with barge v. The sum $\sum_{v=1}^{b} \sum_{r=1}^{R} \pi_{rv} Y_{rv}$ in the objective function (2.9) will have to be replaced by $\sum_{v=1}^{b} \sum_{j=0}^{R} \delta_{vj} \Gamma_{vj}$. Besides, an extra constraint of the type:

$$\sum_{r=1}^{R} Y_{rv} = j\Gamma_{vj} \quad \text{for } v = 1, \ldots, b \quad j = 0, \ldots, R,$$

has to be added. An advantage of the exponential costs for visiting a terminal is that the terminal visits will be divided more equally among the barges. For example, with linear costs visiting eight terminals with one barge and two with the other is equally expensive as visiting five terminals with both barges. In the case of exponential costs, the latter scenario is cheaper. An obvious disadvantage is the need to introduce more decision variables and constraints. Moreover, with exponential costs, it is impossible to distinguish between the terminals: each terminal has the same costs for visiting that terminal as j^{th} terminal. In practice, there are terminals for which it is harder to get an appointment or which impose a higher chance of delay, so we would like to penalize a visit to these terminals more severely.

The second property is that a deep-sea terminal might require a minimum

number of containers to be picked up by barge on a visit. This requirement is caused by the fact that the number of available berths at a deep-sea terminal is limited. If only a few containers are picked up at a single visit, the berth's occupation time of a ship is relatively high because mooring is a time-consuming activity. On the other hand, a constraint to maximize the number of containers picked up by a barge visit might also be needed because it often happens that a terminal can handle only a limited number of containers. Let us denote the maximum number of containers that can be picked by barge v at terminal r as M_{rv} and the minimum number as μ_{rv}. For any combination of r and v the following constraints could be added:

$$\sum_{i:m_i=r} X_{iv} \geq \mu_{rv},$$

$$\sum_{i:m_i=r} X_{iv} \leq M_{rv}.$$

Running example continued

The running example introduced in Section 2.3 is continued. We define the cost parameters as follows: $c^T = 150$, $c_1^B = c_2^B = 25$, the demurrage costs for each container are 60 per day and the storage costs 2 per day. Let the fixed costs for the barges be $\rho_1 = \rho_2 = 500$. Figure 2.2 is the outcome if a terminal visit is penalized with 100, and Figure 2.3 corresponds to a solution if 250 has to be paid for each terminal visit. Note that this problem has many optimal solutions. For example, in Figure 2.3, switching one of the containers on barge 1 with a container on the truck yields the same costs. Moreover, container R3, which is transported on barge 2, can be switched with R4, S3, or S4, yielding the same solution.

2.5.2 Two-stage ILP-based heuristic

An essential aspect of an operational transportation plan for synchromodal transportation is that it can easily be adjusted. If new information becomes available or if specific data changes, the planning should be recalculated. One could think of the arrival of new containers or limitations imposed by the container terminals. Therefore, it is crucial to have a fast algorithm to solve the problem.

The problem formulation from Section 2.5.1 is a generalization of the *Generalized Assignment Problem* (GAP) (Fisher et al., 1986). In the GAP, a set of jobs needs to be assigned for minimum costs to a set of agents, which have a maximum capacity. In our generalization, the containers are the jobs, and the barges and trucks are the agents. Furthermore, we set π_{rv} and ρ_v to zero and give infinite costs for the assignment of a container to a barge that is

Algorithm 2.1: Two-stage ILP-based heuristic.

Input: ILP (2.9)-(2.18)
Step 1:
Solve ILP (2.9)-(2.18) in which the X-variables are relaxed in constraint (2.16), i.e., $0 \leq X_{iv} \leq 1$
Let $\bar{X}$, $\bar{Y}$, and $\bar{Z}$ be the optimal outcome for the, respectively, X-variables, Y-variables, and Z-variables
Step 2:
Fix $\bar{Y}$ and $\bar{Z}$, and force all X-variables to be binary
Solve the resulting ILP and let $\bar{X}$ denote the optimal solution for the X-variables
Output: $\bar{X}$, $\bar{Y}$, and $\bar{Z}$

at the deep-sea terminal on a wrong day. On top of that, all containers are high-priority in the generalization.

The GAP is an NP-hard problem (Fisher et al., 1986), and as we will show in Section 2.6, the running time of the ILP is for larger instances indeed too long for practical purposes. Therefore, a heuristic based on the ILP model from the previous section is developed in this section.

The main advantage of our problem, compared to other models in the literature, is that the route of a barge is not calculated, reducing the number of decision variables by a factor R. Our formulation requires only a Y_{rv}-variable to indicate if terminal r is visited by a barge v. If the route is to be decided, then a variable indicates which terminal is visited after the other terminal. In real-life instances, containers are located at a relatively small number of deep-sea terminals. In the ILP model (2.9)-(2.18), there is an X-variable for each container-barge combination, a Y-variable for each barge-terminal combination, and a Z-variable for each barge. As there are many more containers than terminals and barges, the number of Y-variables and Z-variables is small compared to the number of X-variables. To give an idea, in the instances we consider, the number of containers is about 50 to 100 times larger than the number of terminals and about 100 times larger than the number of barges. Therefore, the number of integrality constraints decreases significantly if the X-variables are relaxed.

This insight gives rise to the two-stage ILP-based heuristic described in Algorithm 2.1. This heuristic determines in Step 1 the barges that are used and which terminals are visited. In Step 2, it finds an allocation of the containers to barges and trucks. For every potential set of visited terminals in Step 1, it is possible to find a feasible allocation of the containers in Step 2 because each container can be assigned to a truck. The assignment of the containers in the second step of Algorithm 2.1 is equivalent to the GAP. In Corollary 2.1, we exploit a property of the GAP to limit the number of fractionally assigned containers in the first step of Algorithm 2.1.

Corollary 2.1. *There exists an optimal solution for step 1 of Algorithm 2.1 for which the number of containers that are not completely assigned to one vehicle is at most the number of barges for which the total capacity is used.*

Proof. Given that $\bar{Y}$ and $\bar{Z}$ are fixed, the value for $\tilde{X}$ is the LP-relaxation of the GAP. In Benders and van Nunen (1983), it is shown that for a linear relaxation of the GAP, the number of fractional assignments is at most the number of machines scheduled to the maximum capacity. Since the number of trucks is unlimited in our problem, the number of containers which is fractionally assigned is at most the number of barges. □

Since the number of barges is small compared to the number of containers, the result of Corollary 2.1 is that the solution after Step 1 is almost feasible, which has two consequences. First, it is easy to obtain a feasible solution for Step 2 from the solution of Step 1. For instance, by assigning the fractional assigned containers to a truck. Moreover, the value of the objective function after Step 1 is likely to be close to the optimal value, so it is probably a tight lower bound. Combining these two properties, it is likely that Step 2 does not require much computation time.

Furthermore, if Algorithm 2.1 selects in Step 1 the optimal terminals to visit, then the heuristic will produce an optimal solution, as we show in Lemma 2.4. We show in Lemma 2.5 that if the visited terminals are different, the difference in the objective function of the two methods can be bounded. In these lemmas, we use X^*, Y^*, and Z^* to denote the optimal solution for the ILP (2.9)-(2.18). Moreover, let $v(X, Y, Z)$ be the value of the objective function for X, Y and Z.

Lemma 2.4. *If $\bar{Y} = Y^*$, then $v\left(\bar{X}, \bar{Y}, \bar{Z}\right) = v\left(X^*, Y^*, Z^*\right)$.*

Proof. The first claim is that if $\bar{Y} = Y^*$, then it must also be that $\bar{Z} = Z^*$. To see that consider that there is at least one terminal r for which $\bar{Y}_{rv}$-variable for barge v is equal to one, then constraint (2.14) implies that $\bar{Z}_v = 1$. If all $\bar{Y}_{rv}$-variables for barge v are zero, the variable $\bar{Z}_v$ could take value zero or one, but since the objective is to minimize costs the $\bar{Z}$ will always be zero. The same arguments hold for Y^* and Z^*, so $\bar{Y} = Y^*$ implies that $\bar{Z}_v = Z^*$.

By the optimality of the ILP, we have that $v\left(\bar{X}, \bar{Y}, \bar{Z}\right) \geq v\left(X^*, Y^*, Z^*\right)$. Moreover, as the $\bar{X}$-variables are the optimal variables, given the variables $\bar{Y}$ and $\bar{Z}$, it must hold that

$$v\left(\bar{X}, \bar{Y}, \bar{Z}\right) = v\left(\bar{X}, Y^*, Z^*\right) \leq v\left(X^*, Y^*, Z^*\right).$$

Since $v\left(\bar{X}, \bar{Y}, \bar{Z}\right)$ is both less than or equal to $v(X^*, Y^*, Z^*)$ and greater than or equal to $v(X^*, Y^*, Z^*)$, these values should be the same. □

Lemma 2.5. *The value of the objective function of the solution of Algorithm 2.1 is bounded by*

$$v(\bar{X},\bar{Y},\bar{Z}) \leq v\left(X^*,Y^*,Z^*\right) + (T-1)\left(\frac{1}{T}bc^T + \sum_{v=1}^{b} c_v^B\right).$$

Proof. In step 1 of Algorithm 2.1, the integrality of the X-variables is relaxed, so the objective function after step 1 is a lower bound for the optimal solution and the solution of the heuristic: $v(\tilde{X},\bar{Y},\bar{Z}) \leq v\left(X^*,Y^*,Z^*\right) \leq v\left(\bar{X},\bar{Y},\bar{Z}\right)$. In the proof of this lemma, the fractional solution $(\tilde{X},\bar{Y},\bar{Z})$ is transformed into a feasible integral solution. Let $\mathcal{F}$ be the set of containers which is fractionally assigned to a barge, i.e., $\mathcal{F} := \{i \in \mathcal{C} | \exists v \in \mathcal{B} : 0 < \tilde{X}_{iv} < 1\}$. By Corollary 2.1, we know that $|\mathcal{F}| \leq b$. We will construct a feasible solution $\left(X',\bar{Y},\bar{Z}\right)$ from the solution after Step 1 from the heuristic, in the following way:

- If $i \in \mathcal{F} \cap \mathcal{H}$, then $X'_{i0} = 1$ and $X'_{iv} = 0$ for $v = 1, \ldots b$;
- If $i \in \mathcal{F} \cap \mathcal{L}$, then $X'_{iv} = 0$ for $v = 0, \ldots, b$;
- If $i \notin \mathcal{F}$, then $X'_{iv} = \tilde{X}_{iv}$ for $v = 0, \ldots, b$.

In other words, in the solution X' none of the containers in $\mathcal{F}$ are assigned to a barge. All high-priority containers in $\mathcal{F}$ are assigned to a truck and all low-priority containers in $\mathcal{F}$ are not transported. We will first show that the sum of the demurrage and storage costs in X' is not higher than in $\tilde{X}$. We will only focus on the containers in the set $\mathcal{F}$ because the other containers have the same demurrage and storage costs in X' and $\tilde{X}$. The increase in storage and demurrage costs for X' compared to $\tilde{X}$ equals for every high-priority container i: $c_{i0}^D + c_{i0}^S\left(1-\tilde{X}_{i0}\right)$. On the other hand, for every container $i \in \mathcal{F}$, the costs $\sum_{v=1}^{b}\left(c_{iv}^D + c_{iv}^S\right)\tilde{X}_{iv}$ are paid in the solution $\tilde{X}$ but not in X'. Hence, the difference between the storage and demurrage costs in X' and $\tilde{X}$ is given by

$$\sum_{i\in\mathcal{F}\cap\mathcal{H}}\left(c_{i0}^D + c_{i0}^S\right)\left(1-\tilde{X}_{i0}\right) - \sum_{i\in\mathcal{F}}\sum_{v=1}^{b}\left(c_{iv}^D + c_{iv}^S\right)\tilde{X}_{iv},$$

which can be reformulated to

$$\sum_{i\in\mathcal{F}\cap\mathcal{H}}\left(c_{i0}^D + c_{i0}^S - \sum_{v=0}^{b}\left(c_{iv}^D + c_{iv}^S\right)\tilde{X}_{iv}\right) - \sum_{i\in\mathcal{F}\cap\mathcal{L}}\sum_{v=1}^{b}\left(c_{iv}^D + c_{iv}^S\right)\tilde{X}_{iv}.$$

It can be shown that this expression is negative:

$$\begin{aligned}
&\sum_{i\in\mathcal{F}\cap\mathcal{H}}\left(c_{i0}^{D}+c_{i0}^{S}-\sum_{v=0}^{b}\left(c_{iv}^{D}+c_{iv}^{S}\right)\tilde{X}_{iv}\right)-\sum_{i\in\mathcal{F}\cap\mathcal{L}}\sum_{v=1}^{b}\left(c_{iv}^{D}+c_{iv}^{S}\right)\tilde{X}_{iv}\\
\leq&\sum_{i\in\mathcal{F}\cap\mathcal{H}}\left(c_{i0}^{D}+c_{i0}^{S}-\sum_{v=0}^{b}\left(c_{iv}^{D}+c_{iv}^{S}\right)\tilde{X}_{iv}\right)\\
\leq&\sum_{i\in\mathcal{F}\cap\mathcal{H}}\left(c_{i0}^{D}+c_{i0}^{S}-\min_{v=0,\ldots,b}\{c_{iv}^{D}+c_{iv}^{S}\}\sum_{v=0}^{b}\tilde{X}_{iv}\right)\\
=&\sum_{i\in\mathcal{F}\cap\mathcal{H}}\left(c_{i0}^{D}+c_{i0}^{S}-\left(c_{i0}^{D}+c_{i0}^{S}\right)\sum_{v=0}^{b}\tilde{X}_{iv}\right)=0.
\end{aligned}$$

In the final equality, the property from constraint (2.11) is used, which implies that $\sum_{v=0}^{b}\tilde{X}_{iv}=1$, for $i\in\mathcal{F}\cap\mathcal{H}$. Concluding, the storage and demurrage costs are in solution X' at most the storage and demurrage costs in solution $\tilde{X}$.

Secondly, we will look at the increase in barge and truck shipping costs in X' compared to $\tilde{X}$, which is equal to

$$\sum_{i\in\mathcal{F}\cap\mathcal{H}}c^{T}\left(1-\tilde{X}_{i0}\right)+\sum_{i\in\mathcal{F}}\sum_{v=1}^{b}t_{i}c_{v}^{B}\tilde{X}_{iv}.$$

To derive a bound for these costs, we divide the set $\mathcal{F}\cap\mathcal{H}$ into the following two subsets: $\mathcal{J}:=\{i\in\mathcal{F}\cap\mathcal{H}:\tilde{X}_{i0}>0\}$ and $\mathcal{K}:=\{i\in\mathcal{F}\cap\mathcal{H}:\tilde{X}_{i0}=0\}$. The containers in set $\mathcal{J}$ are partially assigned to a truck after the first step of Algorithm 2.1. The containers in set $\mathcal{K}$ are fractionally assigned to at least two barges. Without loss of generality, we assume that u_v is integral, so if a fraction of a container is assigned to a truck it should always be at least one TEU. Hence, for all $i\in\mathcal{J}$, the value $\tilde{X}_{iv}$ has to be at least $\frac{1}{T}$. Using a similar argument, the fraction of TEU from container i on barge v is always at most t_i-1, which gives us the following upper bound for the truck and barge transportation costs:

$$\begin{aligned}
&\sum_{i\in\mathcal{J}}c^{T}\left(1-\tilde{X}_{i0}\right)+\sum_{i\in\mathcal{K}}c^{T}+\sum_{i\in\mathcal{F}}\sum_{v=1}^{b}t_{i}c_{v}^{B}\tilde{X}_{iv}\\
\leq&\sum_{i\in\mathcal{J}}c^{T}\left(\frac{T-1}{T}\right)+\sum_{i\in\mathcal{K}}c^{T}+\sum_{i\in\mathcal{F}}\sum_{v=1}^{b}(t_{i}-1)c_{v}^{B} \qquad (2.19)\\
\leq&\left(\frac{T-1}{T}|\mathcal{J}|+|\mathcal{K}|\right)c^{T}+(T-1)\sum_{v=1}^{b}c_{v}^{B}.
\end{aligned}$$

To derive a bound regardless of the sizes of $\mathcal{J}$ and $\mathcal{K}$, the maximum of $\left(\frac{T-1}{T}|\mathcal{J}| + |\mathcal{K}|\right)$ has to be calculated. From Corollary 2.1 it follows that $|\mathcal{F}| \leq b$, and since every container $i \in \mathcal{K}$ is at least assigned to two different barges, we know that $|\mathcal{J}| + 2|\mathcal{K}| \leq b$. In case $T = 1$, the set $\mathcal{F}$ is empty because u_v is integral and each barge v has at most one fractionally assigned container. Hence, T should be at least two, which implies that $\frac{T-1}{T} \geq \frac{1}{2}$. Given this fact and the constraint $|\mathcal{J}| + 2|\mathcal{K}| \leq b$, the maximum of $\frac{T-1}{T}|\mathcal{J}| + |\mathcal{K}|$ is attained at $|\mathcal{J}| = b$ and $|\mathcal{K}| = 0$. Thus the expression in (2.19) can be further bounded by

$$\left(\frac{T-1}{T}|\mathcal{J}| + |\mathcal{K}|\right) c^T + (T-1)\sum_{v=1}^{b} c_v^B \quad \leq \frac{T-1}{T} b c^T + (T-1)\sum_{v=1}^{b} c_v^B.$$

Combining the two bounds for the storage and demurrage costs and the barge and truck transportation costs, the following bound for the value $v(\bar{X}, \bar{Y}, \bar{Z})$ can be derived:

$$\begin{aligned} v\left(\bar{X}, \bar{Y}, \bar{Z}\right) \leq v\left(X', \bar{Y}, \bar{Z}\right) &\leq v\left(\tilde{X}, \bar{Y}, \bar{Z}\right) + \frac{T-1}{T} b c^T + (T-1)\sum_{v=1}^{b} c_v^B \\ &\leq v\left(X^*, Y^*, Z^*\right) + (T-1)\left(\frac{1}{T} b c^T + \sum_{v=1}^{b} c_v^B\right), \end{aligned}$$

which proves the lemma. □

The bound derived in Lemma 2.5 is tight, as the following example illustrates. Consider a small example with one barge with $u_1 = 3$ and two terminals. Let the planning period be a single day. At terminal 1, two containers with a size of two TEU are located and at terminal 2 there is one container of size one TEU. All those three containers are high-priority and since the planning period is one day, the demurrage and storage costs for shipping them per truck or per barge are the same. Therefore, we ignore those costs for the rest of the example. Let the penalty costs for visiting a terminal be $\pi_{11} = \pi_{21} = \frac{1}{2}c^T + \epsilon$ for $\epsilon > 0$. The optimal solution for this problem instance is to visit both terminals and ship one container from terminal 1 and the container from terminal 2 per barge and the other container from terminal 1 per truck. The total costs of this solution are: $2c^T + 2\epsilon - 3c_1^B$.

However, the first step of the heuristic only visits terminal 1. At this terminal, one container is assigned integrally to the barge and only half of the other container is assigned to the barge. The other half of the container and the container at terminal 2 are transported by truck. Hence, the barge in the second step of the heuristic can only visit terminal 1 and load one of the two

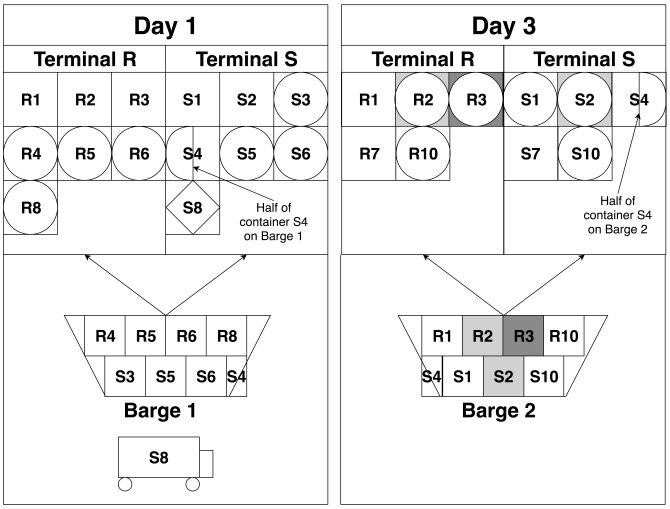


Figure 2.4 Assignment of the containers from the running example after the first step of Algorithm 2.1.

containers at that terminal. The other two containers are shipped per truck, resulting in the following costs: $2\frac{1}{2}c^T + \epsilon - 2c_1^B$. All in all, the difference between the solution produced by the two-stage heuristic and the optimal solution is: $\frac{1}{2}c^T - \epsilon + c_1^B$. As we can take ϵ arbitrarily small, this is equivalent to the difference $(T-1)\left(\frac{1}{T}bc^T + \sum_{v=1}^{b} c_v^B\right)$ from Lemma 2.5.

Running example continued

Similar as in Section 2.5.1, the running example instance is solved for both $\pi_{rv} = 100$ and $\pi_{rv} = 250$. Contrary to the optimal ILP solution, the two-stage heuristic produces the same solution for the two penalties. For both penalties, the solution after Step 1 of the heuristic is the same and is given in Figure 2.4. After the first stage of Algorithm 2.1, all containers but container S4 are assigned in the same way as in Figure 2.3. Half of container S4 is assigned to barge 1, and half of container S4 is assigned to barge 2. With this assignment, the full capacity of fifteen TEU of the two barges is used. In the second step of Algorithm 2.1, container S4 is assigned to a truck, because it is not possible to assign eight containers with a size of two TEU integrally to one of the two barges.

All in all, the heuristic produces for both $\pi_{rv} = 100$ and $\pi_{rv} = 250$ the same solution as in Figure 2.3. So for $\pi_{rv} = 100$, the terminal visits after Step 1 are the same as the optimal visits and thus by Lemma 2.4 the heuristic produces an optimal solution. In the setting that $\pi_{rv} = 250$, the terminal visits after Step 1

Algorithm 2.2: Algorithm to select barges and terminal visits.

```
Input: characteristics of all containers and barges
Sort barges in non-decreasing order of the day they are present at the deep-sea port
for All barges do
    Assign unassigned containers to barge according to Algorithm 2.3
    for All terminals do
        if Barge loads sufficient containers at terminal with respect to Lemma 2.1
          then
            Visit terminal
        end
    end
    Assign all unassigned containers located at visited terminals according to
      Algorithm 2.3
    if Costs of shipping containers assigned to barge plus the barge rent is smaller
      than costs of trucking all high-priority containers assigned to barge then
        Use barge
    end
end
for All unassigned high-priority containers do
    Transport container by truck
end
Output: assignment of containers to barges and trucks
```

are not optimal, and the final solution of the heuristic is not optimal, either. The optimal value of the objective function is 2,668, whereas the two-stage heuristic produces a solution with a value of 2,754. The difference between those two solutions, namely 86, is significantly smaller than the bound given by Lemma 2.5, which is $(2-1)(\frac{1}{2} * 2 * 150 + 25 + 25) = 200$.

2.5.3 Planner algorithm

Current practice, at the inland terminal we have collaborated with, is that experienced planners make a transportation plan by hand. Based on interviews with practitioners, we have developed an algorithm that imitates the behavior of that planner. This algorithm is a greedy algorithm that uses more or less the first-come-first-serve approach. It was also pointed out by Van Riessen et al. (2016) and Wang et al. (2016) that these kinds of methods are often used in real-life planning. This planner algorithm is described in Algorithms 2.2 and 2.3. In Algorithm 2.2, it is described how the planner decides which terminals to visit and which barges to use. Algorithm 2.3 selects for a specific barge the containers to assign to that barge.

In Algorithm 2.2, the barges are considered in chronological order of the day they are at the deep-sea port. It uses Algorithm 2.3 as a subroutine to determine the containers that will be assigned to the current barge. Afterward, Lemma 2.1 is used to decide which terminals are visited. A terminal is only visited if sufficient containers are loaded at that terminal. If there are terminals

Algorithm 2.3: Algorithm to select containers for a given barge.

```
Input: characteristics of all containers and one barge
Select containers which have ETA and call date such that they could be
  transported on barge
Sort containers in non-decreasing order of their end of demurrage free period, ETA
  and terminal
for All unscheduled containers do
    if Call date of container is between arrival of current barge and next barge &
      current barge has capacity to fit container then
        Assign container to barge
    end
end
for All unscheduled containers do
    if Demurrage free period of container ends before next barge arrives & current
      barge has capacity to fit container then
        Assign container to barge
    end
end
for All unscheduled high-priority containers do
    if Current barge has capacity to fit container then
        Assign container to barge
    end
end
for All unscheduled low-priority containers do
    if Current barge has capacity to fit container then
        Assign container to barge
    end
end
Output: set of containers assigned to barge
```

at which insufficient containers are loaded, the barge is not visiting them, so it might be that some capacity on the barge becomes available. Therefore, Algorithm 2.3 is rerun to see if there are unassigned containers available on terminals that are already visited. If all containers are assigned to the barge, the planner algorithm decides if it will use the current barge. If it is more expensive to ship the high-priority containers assigned to the barge per truck than by the barge, the barge is used. Otherwise, we do not use the barge. After all barges are considered, it might be that there are still some unassigned containers. If these containers have a high-priority, they are shipped by a truck.

Algorithm 2.3 uses a single barge as input and selects only the containers that could be transported on that barge. These containers are sorted based on the end of their demurrage free period, ETA, and deep-sea terminal. Afterward, the algorithm goes four times through all containers and checks if a container can be assigned to the barge. In every for-loop containers are only added to the barge if it has free capacity left. In the first for-loop of Algorithm 2.3, containers with a call date such that this barge can transport them, but not the next barge, are added to the barge. In the second for-loop, the algorithm goes

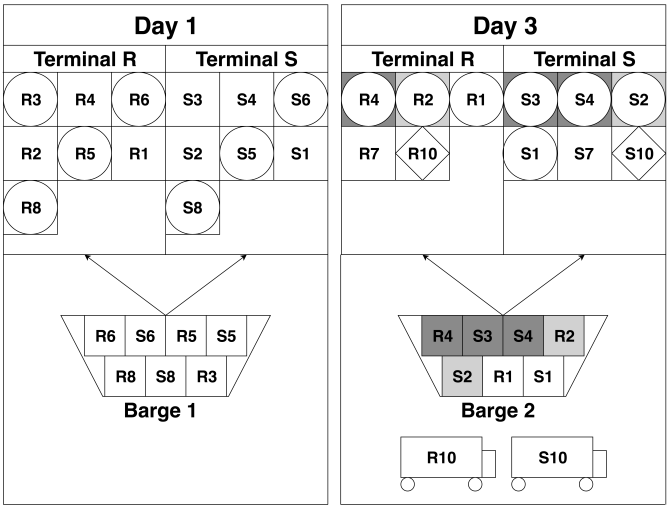


Figure 2.5 Assignment of containers from the running example via the planner algorithm.

again through all unscheduled containers, and if the demurrage free period ends before the next barge arrives, it is scheduled on the current barge. The third part of Algorithm 2.3 goes through all unscheduled high-priority containers, and these containers are added to the barge. If there is still capacity left on the barge, the fourth for-loop assigns low-priority containers to the barge.

Running example continued

In Figure 2.5, containers from the running example are assigned according to the planner algorithm. The containers are displayed at the terminal in this in the way they are sorted at the beginning of Algorithm 2.3. The order from left to right and top to bottom corresponds to the sorting of the containers at the beginning of Algorithm 2.3. The demurrage free period of containers 3, 4, and 6 ends on day 1, and they have the same ETA, so for these containers, the ties are just broken by their container number. After the containers for which the demurrage costs are relevant, container 5 is the first container because its ETA is the lowest.

In the barges, the containers are shown from left to right and top to bottom in the order they are assigned to that barge. Container R6 is the first container to be assigned to barge 1 because it is the first container at Terminal R whose call date is before the arrival day of the next barge. Container S6 is the next container because it has the same characteristics as container R6, but it is only located at another terminal. All containers in Figure 2.5 that

are assigned to a barge are assigned in the first call of Algorithm 2.3 in Algorithm 2.2. Container R3 is the only container assigned to a barge in the second for-loop of Algorithm 2.3, in which the demurrage free period is decisive, all other containers are already assigned in the first for-loop of Algorithm 2.3.

The planner algorithm produces a solution in which four terminals are visited. The minimum containers picked up at a terminal by barge is three. According to Lemma 2.1 with previously defined cost parameters, three is higher than the minimum containers to be picked up given $\pi_{rv} = 100$ or $\pi_{rv} = 250$. Hence, all terminals are visited, and Algorithm 2.3 will not be called a second time in Algorithm 2.2. In the running example, we have assumed that the fixed costs are set low enough that both barges are used. In the last for-loop of Algorithm 2.2 containers R10 and S10 are assigned to a truck.

As the planner algorithm is visiting four terminals in total, it is interesting to see how it performs compared to the outcome of Figure 2.3, which is the optimal solution for visiting four terminals. In both scenarios, fourteen containers are shipped per barge and two per truck. However, in Figure 2.3, demurrage costs had to be paid for four days, and in Figure 2.5 there are eight demurrage days. The planner algorithm is a greedy algorithm, which is not able to look into the future. In this example, that results in the fact that it cannot detect that it already has to ship a container by a truck on the first day. At the end of the planner algorithm, it is decided to transport containers per truck on day 3, which results in more demurrage costs.

2.6 Numerical results

In this section, the performance of the three methods introduced in Section 2.5 is compared. We are interested in how much improvement can be made by implementing the ILP model from Section 2.5.1 compared to the planner algorithm from Section 2.5.3 that models the current practice. Secondly, both the running time and the solution quality of the two-stage heuristic are compared with the ILP.

2.6.1 Medium-sized instances

To evaluate the performance of the three methods, we have used twelve instances based on real-life data. Each instance consists of all containers that are available for transportation and the barges that can be used for transportation. The planning horizon for each instance is set to a week. In Table 2.2, some key properties of the different instances are given. In the second column of Table 2.2, the number of containers for each instance is given. This number varies roughly between 500 and 1,000. In the third column, the total number of TEU of these containers is shown. The percentage of high-priority containers,

Instance	Number containers (n)	Total TEU	% high prio containers	Number barges (b)	Total capacity barges
M1	670	957	35%	4	886
M2	1,163	1,636	38%	6	1,329
M3	549	843	91%	4	886
M4	651	990	90%	4	941
M5	753	1,083	60%	4	850
M6	863	1,279	95%	4	1,004
M7	892	1,221	91%	4	945
M8	596	877	96%	4	945
M9	503	770	91%	4	945
M10	1,064	1,584	91%	5	1,027
M11	855	1,177	72%	4	945
M12	924	1,300	83%	4	945

Table 2.2 Summary of properties of twelve medium-sized instances.

i.e., the containers that need to be shipped, differs between 35% and 96%. In column 5, the number of barges is given, which is most of the time equal to four, but because the capacity of the barges is not always the same, the maximum capacity in TEU shown in the last column differs.

Besides the container and barge characteristics, the costs that are used need to be defined. For transporting a container we use $c^T = 150$ and $c_v^B = 25$ for every barge. The demurrage costs are 40 for every day after the end of the demurrage free period for containers of one TEU and 60 for larger containers. The storage costs are 1 per TEU per day that a container is stored at the inland terminal. Moreover, the fixed costs of using any barge are $\rho_v = 4,500$. Finally, we set π_{rv} such that, according to Lemma 2.1, at least five containers are picked up at every terminal. Hence, $\pi_{rv} = 1,000$ for every barge v and terminal r.

In Table 2.3, the optimal value of the objective function of the ILP model is compared to the outcome of the two-stage heuristic and the planner algorithm. In the second column, the optimal solution of the ILP model is given. In the third and fourth column, the objective function of the two-stage heuristic and the percentage difference with the optimal solution is given. Finally, in the last two columns, the costs of the solution from the planner algorithm are given and the percentage difference with the optimal solution. The first thing to note from Table 2.3 is that the two-stage heuristic produces for all but one of the instances the optimal solution. For the instance for which the solution of the two-stage heuristic is not optimal, it is only 0.2% more expensive than the optimal solution. Second, the value of the planner algorithm is on average 20% higher than the optimal solution. Nevertheless, the solution quality of the planner algorithm differs substantially per instance. For example, for M2 and M9 it produces a solution that is within 2% of the optimal solution, but for

Instance	Optimal solution	Two-stage heuristic	Δ%	Planner algorithm	Δ%
M1	58,213	58,213	0	69,002	18.5
M2	57,785	57,785	0	58,651	1.5
M3	48,429	48,429	0	54,954	13.5
M4	41,812	41,884	0.2	45,942	9.9
M5	47,575	47,575	0	55,679	17.0
M6	68,628	68,628	0	90,075	31.3
M7	63,833	63,833	0	76,198	19.4
M8	52,611	52,611	0	63,034	19.8
M9	42,476	42,476	0	43,320	1.8
M10	82,399	82,399	0	119,431	44.9
M11	71,428	71,428	0	96,970	35.8
M12	65,263	65,263	0	85,814	31.5

Table 2.3 The objective function of the solution of the planner heuristic and two-stage heuristic compared with the optimal solution.

instance M10 the difference between the optimal solution and the solution of the planner algorithm is almost 45%.

The total costs are split into different categories in Table 2.4 to understand which aspect of the planning the ILP outperforms the planner algorithm. In the second column of Table 2.4, the total costs for the planner solution for each category are given. In the third column, the planner's total costs are shown, and in the fourth column, the difference between the optimistic planner and the optimal solution is calculated. In general, the planner algorithm ships more containers per barge and fewer containers per truck. However, in doing that, the planner algorithm's solution has 60% more terminal visits by barge than the optimal solution. On top of that, the planner algorithm results in almost four times as many demurrage costs. An intuitive explanation is that in the optimal solution, a container is shipped more often per truck in order not to visit a terminal or to reduce the demurrage costs than in the solution from the planner algorithm. All in all, the ILP method potentially yields a great amount of cost savings.

The running time for the ILP for all twelve medium-sized instances is less than three seconds. The two-stage heuristic is, on average, about 1.4 times faster. As the running time for the ILP is not that long, one could argue that a heuristic solution is not needed for these instances. In the next section, we will consider larger instances to see if the ILP is still a suitable solution method.

2.6.2 Large-sized instance

The ILP could solve the medium-sized instances from the previous section in a reasonable time. This section will investigate how well the running time of the ILP scales with the input sizes. For that purpose, we have constructed ten

	Optimal solution	Planner algorithm	
Type costs	Total costs	Total costs	Δ
Unused TEU barge penalty	77,250	56,475	20,775
Truck costs	211,800	173,850	37,950
Demurrage costs	47,780	176,340	-128,560
Storage costs	52,622	52,314	308
Terminal visit penalty	131,000	211,000	-80,000
Fixed barge costs	180,000	189,000	- 9,000

Table 2.4 Costs for the optimal ILP and the planner algorithm split out per category.

instances with 1,500 containers. These containers are randomly selected from all medirum-sized instances. Since a large number of containers is selected randomly, the characteristics of the containers in the large-sized instances are quite similar for every instance. Moreover, the barge schedule consists of six barges and is the same for every instance. The cost parameters are the same for the large-sized instances as for the medium-sized instances.

The results for these large-sized instances are given in Table 2.5. In the second, third, and fourth columns of the table, the running times of the, respectively, ILP, two-stage heuristic, and planner algorithm are given. In the fifth, sixth, and seventh column, the objective function of the solution from these three methods is given. Although the instances are rather similar, the running time of the ILP differs substantially. For instances L7 and L10, the ILP did not find the optimal solution after three hours when the algorithm was stopped. On top of that, instances L5 and L9 took almost five minutes to produce the optimal solution. That might be too long if a planner needs to recalculate the consequences of a change to the schedule.

The running time of the two-stage heuristic is for all ten instances about ten seconds, even for the instances for which the ILP could not find the optimal solution in three hours. The planner algorithm produces for all instances in a fraction of a second a solution, which is represented by $<< 1$ in Table 2.5. However, the value of the objective function of the planner algorithm is about 50% higher than for the other two algorithms. Similarly, as was shown in Table 2.4, the solution of the planner algorithm for the large-sized instances has especially more demurrage costs and terminal visits.

The objective function for all ten instances is almost the same, which is not surprising as the instances are similar. Out of the eight instances for which the optimal solution is known, the two-stage heuristic produces the optimal solution seven times. The two-stage heuristic does not provide the optimal solution for instance L8, but its solution is only 0.03% worse. Moreover, the two-stage heuristic produces the solutions for these eight instances on average almost ten times faster than the ILP. Concluding, the ILP method does not work well for

	Running time (sec.)			Objective function		
Instance	ILP	Two-stage	Planner	ILP	Two-stage	Planner
L1	13	8	$<< 1$	96,595	96,595	152,703
L2	32	7	$<< 1$	99,096	99,096	161,385
L3	24	8	$<< 1$	98,167	98,167	157,132
L4	13	7	$<< 1$	98,788	98,788	145,083
L5	297	11	$<< 1$	101,777	101,777	162,332
L6	31	7	$<< 1$	100,406	100,406	158,824
L7	-	7	$<< 1$	-	102,378	162,919
L8	68	8	$<< 1$	101,777	101,808	167,846
L9	280	10	$<< 1$	101,055	101,055	153,598
L10	-	10	$<< 1$	-	99,712	161,736

Table 2.5 Running times and objective function of the ILP, the two-stage heuristic and the planner algorithm for ten large-sized instances.

the large-sized instances. For two instances, it could not produce the optimal solution. Furthermore, for all instances, the two-stage heuristic provides an almost always optimal solution in a fraction of the time of the ILP.

2.7 Conclusion

In this chapter, we have proposed an operational model to minimize the storage and demurrage costs, the truck transportation costs, the empty space on barges and the number of terminals visited by barge. In order to fill the barges as good as possible, we have introduced the concept of high-priority and low-priority containers. The high-priority containers need to be transported, whereas the low-priority containers can be used to fill the barges. To evaluate the benefits of the ILP, we have introduced the planner algorithm that produces a solution in a similar way as an experienced planner.

The potential cost savings for the medium-sized instances are about 20% and for large-sized instances it is about 50%. The fact that the number of variables for assigning a container to a barge is much larger than the number of variables indicating whether a barge is used and a terminal is visited by a barge, is used in the two-stage heuristic. The theoretical difference between the optimal solution and the solution of the heuristic is given. Nevertheless, computational experiments show that the two-stage heuristic almost always finds the optimal solution. Moreover, for the large-sized instances the two-stage heuristic finds the solution much faster than the ILP, which could not find the optimal solution within three hours for some instances.

Unfortunately, the IT infrastructure at the collaborating inland terminal was not capable of making a comparison between the actual transportation plan and our proposed plan. However, it would be interesting to make such a comparison. The benefit of using algorithms does not only lie in finding a better solution

but also in the speed that such a solution is obtained. As a result, using algorithms it is also possible to compare many different scenarios. For instance, the consequences of not visiting a specific terminal with a barge.

We have only included the import flow of containers because that is the dominant flow in most of Europe (Fazi et al., 2015). In the next chapter, we study an extension that also includes the export flow. Moreover, in the next chapter, we also include a stochastic constraint on the maximum number of containers that can be picked up by a barge.

3

Planning hinterland transportation in congested deep-sea terminals

3.1 Introduction

In this chapter, we focus on the consequences of congestion for barges at deep-sea ports. The growing volume of containers transshipped in combination with the increasing size of deep-sea vessels puts immense pressure on the operation of the deep-sea terminals. Hence, the number of containers that can be loaded and unloaded on a barge is limited. We will refer to the number of containers that can be loaded and unloaded at a terminal as the number of *moves*. Unfortunately, this number of moves at the deep-sea terminals is often unknown when the transportation plan needs to be made because of the delay of deep-sea vessels. These deep-sea vessels could be delayed, because of, for instance, bad weather conditions or a late departure at a previous port.

The service of deep-sea vessels has priority over that of barges. Thus, the delay of deep-sea vessels also influences barges' service, which has become rather unreliable. For example, in the port of Rotterdam, waiting times of more than eight hours are not uncommon. At the moment, barges spend about 30-40% of the time they are in the port of Rotterdam waiting to be served at container terminals. Furthermore, in the last year, 20% of the barges were delayed (Port of Rotterdam Authority, 2020b). In this chapter's problem, both the import and export flow of containers must be assigned to barges and trucks. If many containers are loaded, the expected costs for late minute adjustments, that are incurred when the number of moves is insufficient, are high. On the other hand, if only a few export containers are shipped per barge, a large number will be shipped per truck, which results in higher transportation costs.

This chapter's contribution is threefold: first, we present the first problem in the literature in which the uncertain service of barges at deep-sea terminals is studied. In this chapter, we formulate this problem as a two-stage stochas-

This chapter is based on B.G. Zweers, S. Bhulai, and R.D. van der Mei. Planning hinterland container transportation in congested deep-sea terminals. *Flexible Services and Manufacturing Journal*, 2020b. doi: 10.1007/s10696-020-09387-3.

tic problem with recourse. The second contribution is that we give a *Sample Average Approximation* (SAA) method to solve this problem. This method uses Monte Carlo sampling to solve stochastic optimization problems. To the best of our knowledge, we are the first to apply SAA to the field of hinterland container transportation. This technique allows us to solve more realistic problems than the existing methods in the literature. Although the SAA method can produce (almost) optimal solutions for small instances, its running time for larger instances is longer. Therefore, our third contribution is that we propose a heuristic based on *Stochastic Programming* (SP).

The remainder of this chapter is organized as follows. The relevant literature is discussed in Section 3.2. In Section 3.3, the problem is described in more detail, and it is translated into a mathematical model in Section 3.4. Three different solution methods will be discussed in Section 3.5, and the numerical results for these three methods are given in Section 3.6. Finally, the conclusions will be presented in Section 3.7.

3.2 Literature review

In this section, we focus on the operational problems in the multimodal literature that include stochasticity because the relevant literature on deterministic problems has already been discussed in Section 2.2. Moreover, we also focus on the relevant literature on the SAA method.

In stochastic assignment problems, part of the information is uncertain when the transportation plan is made. To the best of our knowledge, there are only three papers in the literature that consider stochastic assignment problems, namely Pérez Rivera and Mes (2017), Zhang et al. (2016), and Zuidwijk and Veenstra (2015). In these three papers, the unknown factor is the number of containers that needs to be transported. This uncertainty is caused by the fact that a deep-sea vessel's arrival could be delayed or that the release of a container by customs might take longer.

In Zuidwijk and Veenstra (2015), this problem is studied for the first time. They consider a problem in which there is only a single barge that visits one terminal. The moment the barge visits the terminal has to be decided. In this decision, a trade-off is made between the number of containers transported by barge and the barge arrival time at the inland terminal. Optimal Pareto-frontiers are given for different information scenarios, and with that, the value of information can be determined.

Zhang et al. (2016) study a problem in which a hinterland operator needs to transport the containers on a deep-sea vessel that is approaching the port. The planner needs to decide on the amount of capacity for each of the different modes of transportation. Each mode of transportation incurs different costs

but also has a different speed. Some containers need to be shipped with a fast and expensive mode of transportation because otherwise, they will be too late at their final destination. In contrast, other containers can go on the slow and cheap mode of transportation. The total demand for containers is known, but the number of containers that needs to go on each mode of transportation is unknown. Similar to Zuidwijk and Veenstra (2015), Zhang et al. (2016) also study a theoretical problem that they can solve to optimality and look into different information scenarios.

The problem described in Pérez Rivera and Mes (2017) is the most similar to our problem. In this problem, a barge makes every day a round-trip from a single inland terminal to a set of deep-sea terminals. They formulate a Markov Decision Problem to decide which terminals to visit with a barge and how many containers of each type should be delivered to or be picked up from a certain terminal. For each terminal that is visited by a barge, a penalty has to be paid. The uncertainty in this model lies in the fact that the demand for each container type is unknown.

The three problems studied by Pérez Rivera and Mes (2017), Zhang et al. (2016), and Zuidwijk and Veenstra (2015) assume that the barge transportation costs are the same for all containers on that barge. This assumption mainly serves to make calculations easier, but it is often unrealistic in practice. To include different costs per container, we will use the SAA method, introduced by Kleywegt et al. (2001). In Shapiro et al. (2009), a theoretical overview of the SAA method is given, and we refer to Kim et al. (2015) for an overview that is more orientated to the application of the SAA method.

The SAA method has been applied to many different areas, but we focus on problems in transportation. In Verweij et al. (2003), the SAA method is used to solve the Stochastic Vehicle Routing Problem. Another transportation problem, the Stochastic Multiperiod Location Transportation Problem, is also solved using SAA in Klibi et al. (2010). In this problem, the design of a transportation network and the routing decisions over multiple planning periods inside this network are taken into account. As the SAA method is hard to solve for large instances of this problem, also heuristics are proposed to solve the problem hierarchically. Another application of the SAA that is related to our work is in Long et al. (2012) in which SAA is used to solve a problem in empty container repositioning.

Finally, Toktas et al. (2006) study the GAP with stochastic capacities, and our problem can be seen as a generalization of this problem. In Toktas et al. (2006), simple heuristics to deal with the uncertainty in the capacity constraints are compared. We will use the two methods that perform best in their study to our problem and discuss them in more detail in Section 3.5.3.

3.3 Problem formulation

We consider a problem in which a single inland terminal needs to export containers to multiple deep-sea terminals and import containers from the same deep-sea terminals. This problem is slightly more stylized than the problem of Chapter 2 because we want to focus on the stochasticity of the number of moves and the export flow of containers. For both the import and export containers, we assume that every container has the following properties:

- given size in TEU;
- fixed transportation costs for each barge and truck;
- deep-sea terminal at which it needs to be picked up (import containers) or needs to be delivered (export containers).

In this problem, we assume that all containers need to be transported, i.e., all containers are high-priority containers (see Definition 2.1). The costs for transporting a container are fixed for each barge and truck because the inland terminal has a contract with barge operators and trucking companies to ship containers for a given price. It is important to note that the costs for each container and barge combination are potentially different because they containers *demurrage*, *storage*, and *detention* costs. Demurrage costs have to be paid for import containers located longer than a specified period at a deep-sea terminal. Moreover, import and containers might need to be stored at the inland terminal, which results in storage costs. Furthermore, an export container needs to be back at the deep-sea terminal after a given number of days, because otherwise, detention costs need to be paid per day that it is late. Especially, the demurrage and detention costs are substantial and influence the transportation plan. Allowing for different transportation costs for different barges also gives us the flexibility to include the fact that some containers cannot be transported on specific barges because it has, for instance, not arrived at the deep-sea terminal. If the costs for transporting a container on a specific barge is set high enough, the container will never be assigned to that barge.

Considering the barges, we make the following assumptions. Every barge

- makes a round trip starting at the inland terminal, going to the deep-sea port and returning at the inland terminal;
- has a capacity in TEU;
- has a maximum number of terminals it can visit in a round trip;
- has an uncertain number of moves at every terminal.

The barge operator decides the moment that a barge is leaving the inland terminal and also the moment it needs to leave the deep-sea port. In a large deep-sea port, there are too many terminals to be all visited by a single barge. That is why the number of terminals that can be visited by a barge is limited. The inland terminal can decide on the terminals that are visited, but the barge operator decides the barge's route.

The congestion at the deep-sea port causes deep-sea terminals to limit the number of moves that can be done for every barge. The inland terminal is allowed to use all these moves solely to unload containers or load containers, but also each possible combination of the two. We have experimented with imposing a penalty for a terminal visit by barge. However, the uncertainty in the number of moves resulted that it was difficult to set the penalty such that the outcome was desirable. It was the case that often either no terminals were visited, or all terminals were visited.

For truck transportation, we make the following two assumptions:

- each truck can only ship a single container;
- there is an unlimited number of trucks.

Trucking a container is more expensive than shipping a container on a barge because each truck can only transport a single container. Since truck transportation is relatively expensive, not many trucks will be used in general for the transportation of containers. Hence, the assumption that the number of available trucks is unlimited does not restrict us much from reality. Furthermore, the benefit of this assumption is that there always exists a feasible solution. We do not consider other modes of transportation, such as trains, in this problem, although they could be incorporated in our framework.

In general, inland terminals are not located in the direct neighborhood of a deep-sea terminal. Therefore, the transport plan is made several days in advance. However, at that time, the number of moves at a deep-sea terminal is unknown, or it is unreliable. Hence, the number of moves is modeled as a stochastic variable. In this problem, we assume that the total number of containers that can be loaded and unloaded at a terminal is only revealed after all barges have left the inland terminal. At that time, all barges are already loaded with export containers.

In case a barge is shipping more export containers for a particular terminal than the number of moves, the barge cannot unload all its containers in the allocated slot. For each container that cannot be unloaded, a *recourse action* is required, which could be unloading the container at another terminal or returning it to the inland terminal. In both cases, extra transportation costs are

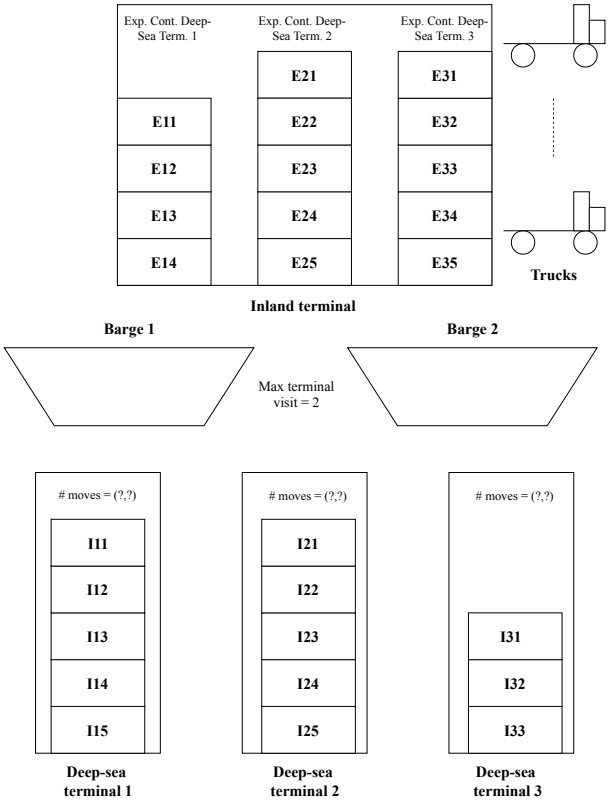


Figure 3.1 All available containers and barges.

incurred, and the planner at the inland terminal has to do extra work, which may be costly. The main problem we are facing is determining which terminals to visit with each barge and which export containers to load on each barge. Below, we give an illustration of the problem.

Problem illustration

In Figures 3.1, 3.2, 3.3, and 3.4 an illustration is given of the problem we consider and a possible solution. It is important to note that this is not necessarily the optimal solution. In Figure 3.1, the input data of the problem is given. For simplicity, we ignore the cost structure in this example. We consider an example with two barges, and both barges can visit at most two of the three deep-sea terminals. Besides that, the capacity of both barges is eight containers. The containers at the inland terminal are grouped with respect to the

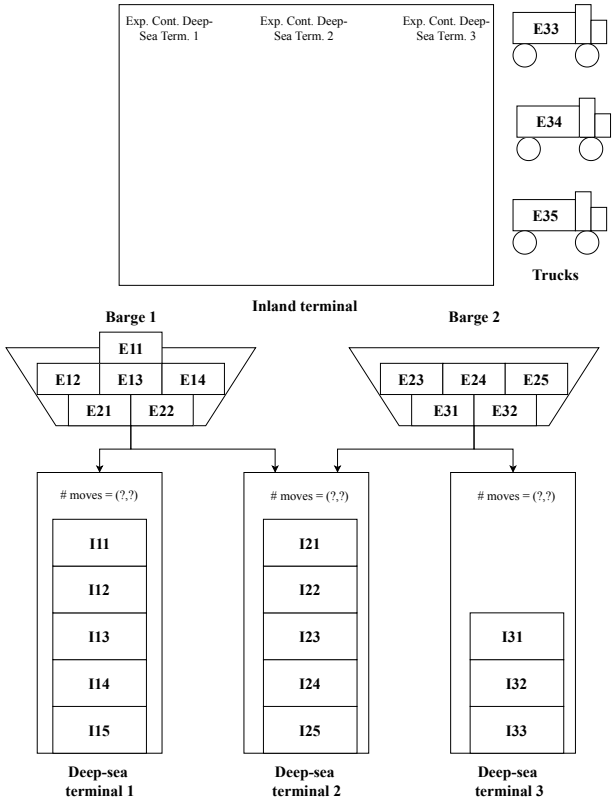


Figure 3.2 Decision for deep-sea terminal visits by barge and shipment of export containers.

deep-sea terminal to which they need to go. For instance, container E12 is the export container with number 2 that needs to go to the first deep-sea terminal. We assume that every container can be shipped on both of the barges. Every deep-sea terminal has for every barge a certain number of moves available. Nevertheless, at the beginning of the planning phase, this number is unknown, and therefore it is represented by a question mark in Figure 3.1.

In Figure 3.2, it has been decided which deep-sea terminals are visited by which barge and which export containers are loaded on which barge. Barge 1 is visiting deep-sea terminals 1 and 2, and barge 2 is visiting deep-sea terminal 2 and 3. In Figure 3.2, it is also depicted which export containers are loaded on the two barges. Although there is capacity left on the second barge, some containers for deep-sea terminal 3 are shipped per truck (containers E33, E34,

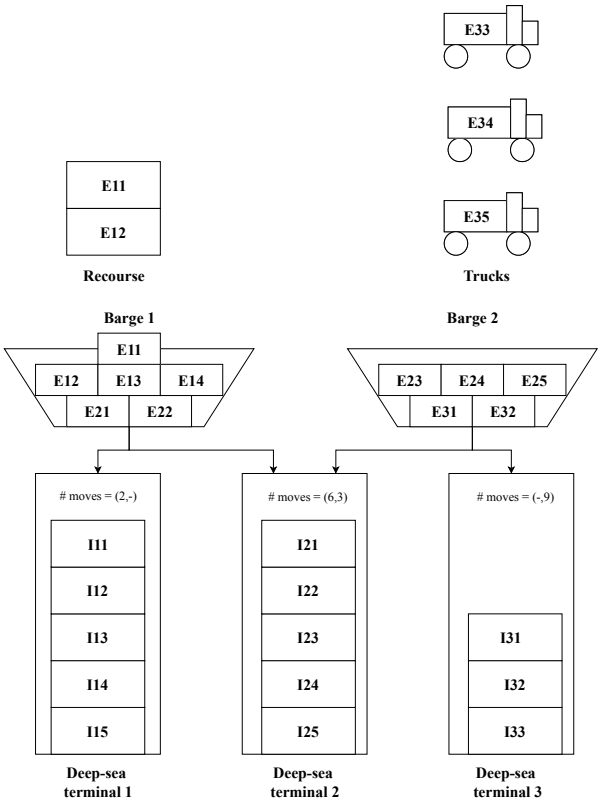


Figure 3.3 Revealment number moves and recourse actions.

and E35). These containers are shipped per truck because the number of moves at the deep-sea terminals is still unknown at this stage.

In Figure 3.3, the numbers of moves at the deep-sea terminals are revealed. The number of moves at a deep-sea terminal is only revealed if a barge is visiting the terminal. For instance, deep-sea terminal 1 is only visited by barge 1, and thus, the number of moves for this barge is revealed (2), but the number of moves for the second barge is not relevant. Four export containers for deep-sea terminal 1 were loaded on barge 1, but the number of moves is only two, meaning that for two of the four containers a recourse action is required.

In Figure 3.4, it is shown, for each barge, how many moves are left for the import containers. The number of moves left is the same as the number of moves in Figure 3.3 minus the number of export containers unloaded at that terminal. For instance, the number of moves for barge 2 at deep-sea terminal 2

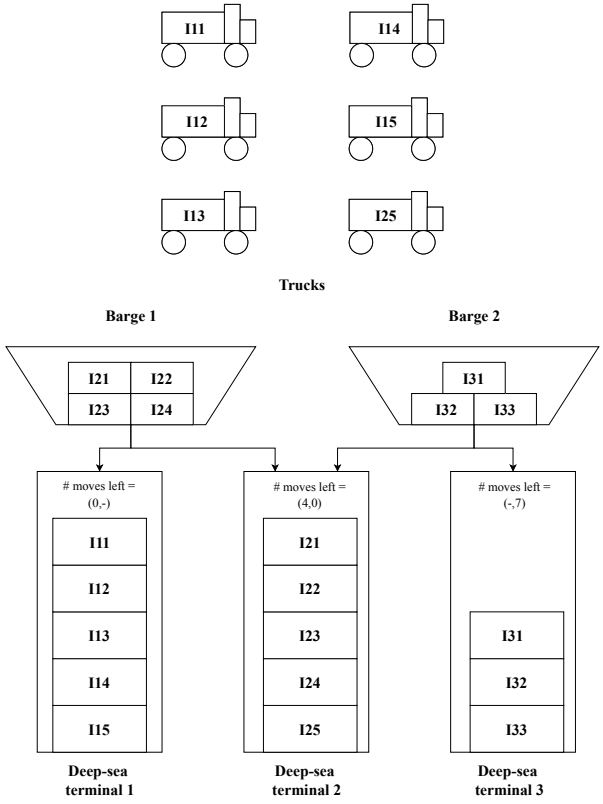


Figure 3.4 Decision for shipment of import containers.

was three, but since also three export containers were unloaded at the deep-sea terminal, no more moves are left for barge 2. As the first barge has four moves left at the deep-sea terminal 2, that barge can ship four of the five import containers located at the second deep-sea terminal. The fifth container has to be shipped per truck, similar to the five containers located at deep-sea terminal 1 because no moves are left at that terminal for barge 1.

The second barge has seven moves left at the third deep-sea terminal, but only three import containers are located at that terminal. In retrospect, it would have been better to load more export containers for terminal 3 on the second barge, because the number of moves was sufficiently large. The assignment of the import containers that is done in Figure 3.4 is relatively easy compared to the assignment of the export containers in Figure 3.2 because the number of moves has been revealed. Hence, the problem is entirely deterministic.

$\mathcal{I}(k)$	Set of import containers located at terminal k
$\mathcal{E}(k)$	Set of export containers with destination terminal k
n^i	Total types of import containers
n^e	Total types of export containers
m	Total number of barges
l	Total number of terminals
c^i_{tj}	Costs of transporting import container of type t with barge j
c^i_{t0}	Costs of transporting import container of type t per truck
c^e_{tj}	Costs of transporting export container type t with barge j
c^e_{t0}	Costs of transporting export container type t per truck
q_{jk}	Recourse costs for a move at terminal k for barge j
u_j	Capacity of barge j in TEU
N_j	Maximum number of terminal to be visited by barge j
d^e_t	Number of export containers of type t
d^i_t	Number of import containers of type t
w^e_t	Size of export container type t in TEU
	w^i_t Size of import container type t in TEU
Φ_{jk}	Random variable indicating the maximum number of import and
	export containers that can be collected from terminal k by barge j
$F(\cdot)$	Cumulative distribution function for number of moves for barge j at terminal k
ϕ_{jk}	Realization of random variable Φ_{jk}

Table 3.1 Notation for the input parameters used in Chapter 3.

3.4 Mathematical model

In this section, the problem described in Section 3.3 is formulated as a two-stage stochastic problem. We consider a problem in which there are m barges, and we use j as an index for the barges. Each barge j has a capacity of u_j TEU and can visit at most N_j terminals. Furthermore, there are n^e types of export containers and n^i types of import containers. We use t as an index both for the type of import and export containers. To a type of container belong all containers with the same characteristics. For an import (export) container these characteristics are: the size in TEU of a container w^i_t (w^e_t), the cost of transporting a container of type t on barge $j = 1, \ldots, m$ is c^i_{tj} (c^e_{tj}) and per truck it is c^i_{t0} (c^e_{t0}). Moreover, all containers in a type are located at the same terminal. The total number of import container and export containers of type t is given by, respectively, d^i_t and d^e_t.

In total, there are l terminals, and we use the index k to refer to a terminal. The set of import and export containers at terminal k is, respectively, $\mathcal{I}(k)$ and $\mathcal{E}(k)$. The number of moves by barge j at terminal k is given by the stochastic variable Φ_{jk}. We use ϕ_{jk} to refer to a realization of that stochastic variable.

Finally, if more than ϕ_{jk} containers from the sets $\mathcal{I}(k)$ and $\mathcal{E}(k)$ are assigned to barge j, then a cost of q_{jk} per container above the threshold ϕ_{jk} is incurred. The notation for the input parameters is summarized in Table 3.1.

The remainder of this section is organized as follows: first, we formulate in Section 3.4.1 a model in which we assume that the number of moves at a terminal is deterministic. In Section 3.4.2, the two-stage stochastic problem with recourse is formulated.

3.4.1 Deterministic model

We first assume that the number of moves at terminal k for barge j is the deterministic value ϕ_{jk}. We use three different types of decision variables: X^e_{tj}, X^i_{tj}, and Y_{jk}. The variable Y_{jk} is a binary variable indicating that terminal k is visited by barge j. The variable X^e_{tj} indicates the number of export containers of type t that are transported on barge j, if $j = 1, \ldots, m$; and if $j = 0$, it gives the number of export containers of type t on a truck. The variables X^i_{tj} represent the same, but then for import containers. This deterministic problem can be modeled as an ILP in the following way:

$$\min \quad \sum_{t=1}^{n^i} \sum_{j=0}^{m} c^i_{tj} X^i_{tj} + \sum_{t=1}^{n^e} \sum_{j=0}^{m} c^e_{tj} X^e_{tj} \tag{3.1}$$

subject to

$$\sum_{j=0}^{m} X^i_{tj} = d^i_t \qquad t = 1, \ldots, n^i \tag{3.2}$$

$$\sum_{j=0}^{m} X^e_{tj} = d^e_t \qquad t = 1, \ldots, n^e \tag{3.3}$$

$$\sum_{t=1}^{n^i} w^i_t X^i_{tj} \leq u_j \qquad j = 1, \ldots, m \tag{3.4}$$

$$\sum_{t=1}^{n^e} w^e_t X^e_{tj} \leq u_j \qquad j = 1, \ldots, m \tag{3.5}$$

$$\sum_{t \in \mathcal{I}(k)} X^i_{tj} + \sum_{t \in \mathcal{E}(k)} X^e_{tj} \leq \phi_{jk} \qquad j = 1, \ldots, m \qquad k = 1, \ldots, l \tag{3.6}$$

$$\sum_{k=1}^{l} Y_{jk} \leq N_j \qquad j = 1, \ldots, m \tag{3.7}$$

$$X^i_{tj} \leq d^i_t Y_{jk} \qquad j = 1, \ldots, m \qquad k = 1, \ldots, l \qquad t = 1, \ldots, n^i \tag{3.8}$$

$$X^e_{tj} \leq d^e_t Y_{jk} \qquad j = 1, \ldots, m \qquad k = 1, \ldots, l \qquad t = 1, \ldots, n^e \tag{3.9}$$

$$X^i_{tj} \in \mathbb{N}_0 \qquad t = 1, \dots n^i \qquad j = 1, \dots, m \qquad (3.10)$$

$$X^e_{tj} \in \mathbb{N}_0 \qquad t = 1, \dots n^e \qquad j = 1, \dots, m \qquad (3.11)$$

$$Y_{jk} \in \{0, 1\} \qquad j = 1, \dots, m \qquad k = 1, \dots, l. \qquad (3.12)$$

The objective function (3.1) consists of two sums. The first sum represents the total costs for barge and truck transportation of the import containers. The second sum is the same as the first sum, except for the fact that it represents the total costs for the export containers. Constraints (3.2) and (3.3) ensure that for each type of import, respectively export container, the entire demand is transported. Moreover, constraints (3.4) and (3.5) enforce that the total number of import and export containers assigned to a barge does not violate the capacity of a barge. The number of containers on a barge that can be transshipped at a terminal is limited by constraint (3.6). Furthermore, in constraint (3.7) the number of terminals that can be visited by a barge is limited. Constraints (3.8) and (3.9) couple the X-variables and Y-variables. Constraints (3.10) and (3.11) ensure that the number of export and import containers on a truck and barge is non-negative. The Y-variables are ensured to be binary in constraint (3.12).

The capacity of the barge is considered for all import and export containers in constraints (3.4) and (3.5). However, there is no constraint to enforce that the number of export containers unloaded at, or after, the k^{th} terminal and the number of loaded import containers before visiting the k^{th} terminals is not exceeding the capacity of the barge. Hence, it might happen that, after a terminal visit, the number of containers on the barge exceeds the capacity of the barge. Nevertheless, if constraints (3.4) and (3.5) are satisfied, there exists a route visiting the terminals such that the capacity constraint is never violated during the entire trip.

To see that, let us denote the difference between the export containers unloaded at terminal k by barge j and the number of import containers loaded at terminal k by barge j as $\delta_{jk} := \sum_{t \in \mathcal{E}(k)} X^e_{tj} - \sum_{t \in \mathcal{I}(k)} X^i_{tj}$. If barge j visits the terminals in non-increasing order of δ_{jk}, then the total number of containers on barge j will decrease after each terminal visit as long as $\delta_{jk} > 0$ and thus the capacity will never be exceeded. If δ_{jk} is negative for a terminal, more import containers are loaded than export containers are unloaded for that terminal, which could potentially lead to a violation of the capacity of the barge. Nevertheless, for all following terminals that are visited by the barge j, the total number of unloaded export containers is smaller than the import containers that are loaded at those terminals. Since all import containers satisfy the capacity constraint, it is impossible that, after a visit to a terminal, the total remaining export containers in combination with the already loaded import containers are violating the capacity constraint of a barge.

3.4.2 Two-stage stochastic problem with recourse

To incorporate the uncertainty in the number of moves for a barge at a terminal, we model the problem as a *two-stage stochastic problem with recourse*. For an introduction to stochastic problems with recourse, we refer to Birge and Louveaux (2011). The general idea of a two-stage stochastic problem is that the decision variables can be divided into two groups: the *first-stage* and the *second-stage* decision variables. The first-stage decision variables have to be decided before any of the realizations of the stochastic variables are known, whereas the value for the second-stage decision variables can be determined when the realizations of the stochastic variables are known. In our problem, the first-stage decisions are the decisions about which terminals to visit by which barge and which export containers to ship on which barge or truck. The second-stage decisions are the decisions concerning the transportation of import containers.

In a recourse problem, it is assumed that besides the second-stage decision variables, also *recourse actions* can be taken after the realization of the random variables. Usually, these actions have high costs and are only chosen to 'repair' the first-stage decisions in case the realization of the random variables is 'bad'. In our problem, the recourse action is the opportunity to unload export containers at another terminal and use trucks to ship them to the original destination. This action is chosen if, for a certain terminal, more export containers are on a barge than the number of moves at that terminal.

In the ILP formulation of (3.1)-(3.12) above, the realization ϕ_{jk} was assumed to be deterministic and known before any decision is made. However, in the two-stage stochastic problem, the realization ϕ_{jk} is only known when the decisions for the X^i-variables need to be made. The decisions for X^e and Y need to be made in such a way that the expected costs in the second stage are minimized. In other words, the two-stage stochastic problem can be formulated as follows:

$$\min \quad \sum_{t=1}^{n^e} \sum_{j=0}^{m} c^e_{tj} X^e_{tj} + \mathbb{E}\left[Q(X^e, Y, \Phi)\right] \tag{3.13}$$

subject to

$$\text{Constraints (3.3), (3.5), (3.7), (3.9), (3.11), and (3.12),} \tag{3.14}$$

in which $Q(X^e, Y, \phi)$ is defined as:

$$Q(X^e, Y, \phi) := \quad \min \sum_{t=1}^{n^i} \sum_{j=0}^{m} c^i_{tj} X^i_{tj} + \sum_{j=1}^{m} \sum_{k=1}^{l} q_{jk} Z_{jk} \tag{3.15}$$

subject to

$$\sum_{j=1}^{m}\sum_{t=1}^{n^i} X^i_{tj} + \sum_{j=1}^{m}\sum_{t=1}^{n^e} X^e_{tj} - Z_{jk} \leq \phi_{jk} \qquad j = 1,\ldots,m \qquad k = 1,\ldots,l \quad (3.16)$$

Constraints (3.2), (3.4), (3.8), and (3.10) (3.17)

$$Z_{jk} \in \mathbb{N}_0 \qquad j = 1,\ldots,m \quad k = 1,\ldots,l. \quad (3.18)$$

The problem in (3.13)-(3.14) is the first stage problem and the second stage problem is given in (3.15)-(3.18). The function $Q(X^e, Y, \phi)$ represents the optimal costs for the import containers given the first-stage decision variables X^e and Y, and realization ϕ. In the formulation (3.15)-(3.18), the variable Z_{jk} indicates the number of export containers on barge j that could not be unloaded at terminal k because the maximum number of moves has been reached at that terminal. For these containers, a recourse action is required.

The second stage is a deterministic assignment problem that can be solved relatively easily to optimality. Nevertheless, computing $\mathbb{E}[Q(X^e, Y, \Phi)]$ is computationally very expensive because the number of realizations can be enormous, and it is not possible to derive a closed-form expression for $Q(X^e, Y, \phi)$. Therefore, in the following section, we describe three different methods to solve this two-stage stochastic problem.

3.5 Solution methods

In this section, we describe five different methods to solve the problem (3.13)-(3.18). Once the first-stage decision variables are fixed, and the realization of ϕ is known, determining the optimal values for the second-stage variables is a simple assignment problem. This problem is similar to the second step of the two-stage algorithm of Chapter 2, and we have seen that this can be computed efficiently. Therefore, we will only describe methods to find solutions for the first-stage variables. Both the terminals to visit by each barge and the transport decision of each export container are first-stage decision variables. Nevertheless, from the decision which export containers to transport on which barge follows which terminals are visited by a barge. Hence, only reporting the transportation plan of the export containers is sufficient as a first-stage decision.

The first method that we use to solve our problem is the SAA method. The solution of this method is based on multiple realizations of the random variable Φ_{jk}. The solution obtained by the SAA method converges to the optimal solution if the number of realizations increases. Nevertheless, the running time of this method also increases with a growing number of realizations. If all

input data is known long before the planning process has to finish, then it is not problematic if the running time of the SAA method is long. However, in many situations, the required information is only available a short amount of time before the planning has to be made. Hence, also a faster method than the SAA method is required.

We use *Stochastic Programming* (SP) to derive such a method, and thus, we refer to this method as the *SP-based* method. The idea behind this method is the following: if we simplify our problem enough, it can be solved to optimality using stochastic programming. In the SP-based method, the characteristics of this optimal solution are used to derive a fast solution for the original problem. Although the solution SP-based method is based on the solution for a simplified problem, the idea is that this simplified problem resembles enough of the original problem, such that the SP-based solution is close to the optimal solution.

To compare the performance of the SP-based method, we also describe in this section three fast methods that have been proposed before in the literature. These methods are applied with success to other stochastic assignment problems but are less tailored towards our problem.

The remainder of this section is organized as follows: in Section 3.5.1, we give the SAA method, and the SP-based method is given in Section 3.5.2. Afterward, in Section 3.5.3, we give three methods that perform well in the study of Toktas et al. (2006).

3.5.1 Sample Average Approximation method

The main idea of SAA is that in many stochastic optimization problems, the expectation in the objective function is hard to compute exactly, but that for a given realization of the random variable, the problem is easier to solve. This is also the case in our problem, in which the ILP given in (3.1)-(3.12) is (relatively) easy to solve and the stochastic problem (3.13)-(3.18) is hard to solve. The goal of the SAA method is to approximate $\mathbb{E}[Q(X^e, Y, \Phi)]$ by using a set of N vectors of realizations $\bar{\phi}_N := (\phi^1, \phi^2, \ldots, \phi^N)$. We refer to a vector of realizations ϕ^n as a *scenario* and use the index n to refer to a scenario. A scenario ϕ^n consists of one realization ϕ_{jk} of Φ_{jk} for every $j = 1, \ldots, m$ and $k = 1, \ldots, l$. For each scenario, the value for $Q(X^e, Y, \phi^n)$ can be evaluated, so $\mathbb{E}[Q(X^e, Y, \Phi)]$ can be approximated by $\frac{1}{N}\sum_{n=1}^{N} Q(X^e, Y, \phi^n)$.

In the SAA-method, for each scenario n, the assignment of the import containers depends on the value ϕ^n. Hence, a decision variable X_{tj}^{in} is needed that indicates the number of import containers of type t transported on barge j for scenario n. On top of that, a variable Z_{jk}^{n} is needed to indicate the number of containers on barge j for terminal k for which a recourse action is required in scenario n. Nevertheless, the export containers and the set of visited terminals

should be the same for every realization. Thus, the variables X^e_{tj} and Y_{jk} do not depend on the realization of ϕ^n. Using these variables, the SAA method for our problem can be formulated as the following ILP:

$$\min \quad \sum_{t=1}^{n^e}\sum_{j=0}^{m} c^e_{tj}X^e_{tj} + \frac{1}{N}\sum_{n=1}^{N}\left(\sum_{t=1}^{n^i}\sum_{j=0}^{m} c^i_{tj}X^{in}_{tj} + \sum_{j=1}^{m}\sum_{k=1}^{l} q_{jk}Z^n_{jk}\right) \tag{3.19}$$

subject to

$$\sum_{j=0}^{m} X^{in}_{tj} = d^i_t \qquad t = 1,\dots n^i \qquad n = 1,\dots,N \tag{3.20}$$

$$\sum_{t=1}^{n^i} w^i_t X^{in}_{tj} \le u_j \qquad j = 1,\dots,m \qquad n = 1,\dots,N \tag{3.21}$$

$$X^{in}_{tj} \le d^i_t Y_{jk} \qquad j = 1,\dots,m \qquad k = 1,\dots,l$$
$$t = 1,\dots,n^i \qquad n = 1,\dots,N \tag{3.22}$$

$$\sum_{t\in\mathcal{I}(k)} X^{in}_{tj} + \sum_{t\in\mathcal{E}(k)} X^e_{tj} - Z^n_{jk} \le \phi^n_{jk}$$
$$j = 1,\dots,m \qquad k = 1,\dots,l \qquad n = 1,\dots,N \tag{3.23}$$

Constraints (3.3), (3.5), (3.7), (3.9), (3.11) (3.24)

$$X^{in}_{tj} \in \mathbb{N}_0 \qquad t = 1,\dots,n^i \qquad j = 1,\dots,m \qquad n = 1,\dots,N \tag{3.25}$$

$$Z^n_{jk} \in \mathbb{N}_0 \qquad j = 1,\dots,m \qquad k = 1,\dots,m \qquad n = 1,\dots,N. \tag{3.26}$$

Let us denote the optimal solution of the problem (3.19)-(3.26) by the quadruple $(X^{e^*}, X^{in^*}, Y^*, Z^{n^*})$. It is important to realize that this solution is optimal for the objective function (3.19), but it is not necessarily the optimal solution for the objective function (3.13). We will refer to the optimal first stage decisions of the problem (3.19)-(3.26) as $\hat{x}_N$ and to the value of the objective function (3.19) under $\hat{x}_N$ and $\bar{\phi}_N$ as $\hat{z}_N(\hat{x}_N, \bar{\phi}_N)$.

Let us denote the solution produced by the SAA method by x^{SAA}. One could take for x^{SAA} the value of $\hat{x}_N$, but in order to get a better solution than $\hat{x}_N$ the problem (3.19)-(3.26) can be solved multiple times. We could solve the problem (3.19)-(3.26) for M different sets of N samples, which we denote by $\bar{\phi}^1_N, \bar{\phi}^2_N, \dots, \bar{\phi}^M_N$. This will result in M solutions, $\hat{x}^1_N, \hat{x}^2_N, \dots, \hat{x}^M_N$. From these M solutions, only one can be executed in practice. To select the best of these M solutions, we use a different sample $\bar{\phi}_{N'}$ consisting of N' scenarios to calculate $\hat{z}_{N'}(\hat{x}^m_N, \bar{\phi}_{N'})$, for $m = 1, 2, \dots, M$. One could see the original M samples of size N as a training set and this new set with a sample of size N' as a validation

set. It is common practice in SAA to select the solution x^{SAA} that has the lowest objective function under the validation set. In other words,

$$x^{SAA} := \arg\min \left\{ \hat{z}_{N'} \left(\hat{x}_N, \bar{\phi}_{N'} \right) : \hat{x}_N \in \{ \hat{x}_N^1, \hat{x}_N^2, \ldots, \hat{x}_N^M \} \right\}.$$

An advantage of the SAA method is that it can also be used to calculate a lower bound and an upper bound for the optimal solution, and these bound can then be used to calculate an optimality gap. We first show how a lower bound for the optimal solution can be obtained. As the solution $\hat{x}_N^m$ was determined by using the sample $\bar{\phi}_N^m$, the value $\hat{z}_N(\hat{x}_N^m, \bar{\phi}_N^m)$ is negatively biased estimator for the optimal value of the real objective function (3.13), which we denote by z^*. In other words, $\hat{z}_N(\hat{x}_N^m, \bar{\phi}_N^m)$ is a lower bound for the optimal solution and the following inequality holds (Mak et al., 1999):

$$\mathbb{E}\left[\hat{z}_N(\hat{x}_N^m, \bar{\phi}_N^m) \right] \leq z^*.$$

If we take the average over all M values for $\hat{z}_N(\hat{x}_N^m, \bar{\phi}_N^m)$ for $m = 1, \ldots, M$, we get $\bar{t}_N^M := \frac{1}{M} \sum_{m=1}^M \hat{z}_N(\hat{x}_N^m, \bar{\phi}_N^m)$, which is in expectation equal to $\hat{z}_N(\hat{x}_N^m, \bar{\phi}_N^m)$. Therefore, we use $\bar{t}_N^M$ as an estimation for the lower bound of the objective function of the problem in (3.13)-(3.18).

Let us now focus on the upper bound for the optimal solution. As the solution $\hat{x}_N^m$ is a feasible solution for (3.13)-(3.18), the value $\hat{z}_{N'}(\hat{x}_N^m, \bar{\phi}_{N'})$ is an upper bound for the optimal objective function. Similar to $\bar{t}_N^M$, we define $\bar{v}_{N'}^M$ as the average over all M objective functions given the scenarios in the validation set, i.e, $\bar{v}_{N'}^M := \frac{1}{M} \sum_{m=1}^M \hat{z}_{N'}(\hat{x}_N^m, \bar{\phi}_{N'})$. All in all, we have that $\mathbb{E}[\bar{v}_{N'}^M] \geq z^*$. Therefore, an estimation for the *optimality gap* is given by $\bar{v}_{N'}^M - \bar{t}_N^M$. Since both the $\bar{t}_{N'}^M$ and $\bar{v}_N^M$ are estimates, the Central Limit Theorem can be used to take accuracy into account. An estimator of the optimality gap that takes accuracy into account is (Kleywegt et al., 2001):

$$\bar{v}_{N'}^M - \bar{t}_{N'}^M + z_\alpha \frac{\bar{S}^M}{\sqrt{M}}, \tag{3.27}$$

in which $z_\alpha := \Phi^{-1}(1-\alpha)$, where $\Phi(z)$ is the cumulative distribution function of the standard normal distribution and $\frac{\bar{S}^M}{\sqrt{M}}$ is defined as:

$$\sqrt{\frac{1}{M(M-1)} \sum_{m=1}^M \left(\left(\hat{z}_{N'}(\hat{x}_N^m, \bar{\phi}_{N'}) - \hat{z}_N(\hat{x}_N^m, \bar{\phi}_N^m) \right) - \left(\bar{v}_{N'}^M - \bar{t}_N^M \right) \right)^2}. \tag{3.28}$$

All in all, the SAA method consists of two phases: a *training* and a *validation* phase. In the training phase, M different problems with N scenarios are solved and in the validation phase, the quality of these solutions is investigated for N'

d^i	Total number of import containers
d^e	Total number of export containers
c^i	Costs of transporting import container by barge
c_t^i	Costs of transporting import container per truck
c^e	Costs of transporting export container by barge
c_t^e	Costs of transporting export container per truck
q	Recourse costs
Φ	Random variable indicating the number of moves
$F(\cdot)$	Cumulative distribution function for the number of moves
x	Number of export containers transported by barge

Table 3.2 Notation used in simplified problem in Section 3.5.2.

different scenarios. It should be noted that the validation phase is less computationally intensive than the training phase. In the training phase, a decision for both the first-stage and the second-stage variables needs to be made. To do so, all scenarios in the training set have to be considered simultaneously. On the other hand, in the validation phase, the first-stage decision variables and the realization of the stochastic variables are fixed and only a decision for the second-stage decision variables is needed. Consequently, each of the N' scenarios can be investigated independently.

3.5.2 Stochastic Programming method

In this section, we explain how the SP-based method is derived. We explain first how to get an optimal solution for the simplified problem, and afterward this solution will be used to derive a heuristic for the problem (3.13)-(3.18).

Optimal solution simplified problem

Let us consider a situation with only one barge and one terminal. The number of export containers to be delivered to that terminal is given by d^e, and d^i denotes the number of import containers that must be collected from that terminal. Moreover, we assume that the barge transportation costs for all import and export containers are c^i and c^e, respectively. Furthermore, the costs of transporting an import or an export container with a truck are denoted by c_t^i and c_t^e, respectively. The number of moves at the terminal is the stochastic variable Φ, distributed according to the cumulative distribution function $F(\cdot)$. Finally, the costs of a recourse action for an export container are q. The notation for this simplified problem is summarized in Table 3.2.

Let us use x to denote the decision variable on the number of export containers on the barge. We assume that this decision variable is continuous. From the decision x, the number of export containers on a truck, $d^e - x$, follows di-

rectly. Furthermore, we assume that the variable Φ is that restrictive that we can ignore the capacity of a barge. If x export containers are transported, and ϕ is the realization of the number of moves, then no import containers can be shipped by barge if $x \geq \phi$. In case $x < \phi$, at most $\phi - x$ import containers can be shipped per barge. Hence, given x, the expected number of import containers per barge is equal to $\mathbb{E}[\min\{d^i, \max\{0, \Phi - x\}\}]$. Moreover, the expected number of export containers for which we need to perform a recourse action is given by $\mathbb{E}[\max\{0, x - \Phi\}]$.

For $0 \leq x \leq d^e$, the expected total costs $T(x)$ for this problem can be given by the following expression:

$$\begin{aligned} T(x) = c^e x + c_t^e(d^e - x) + c^i \mathbb{E}[\min\{d^i, \max\{0, \Phi - x\}\}] \\ + c_t^i(d^i - \mathbb{E}[\min\{d^i, \max\{0, \Phi - x\}\}]) + q\mathbb{E}[\max\{0, x - \Phi\}]. \end{aligned} \tag{3.29}$$

Equation (3.29) consists of five terms: the first two terms are deterministic because they represent, respectively, the barge and truck costs for the export containers. The third part of the sum gives the expected barge costs for the import containers and the expected truck costs for the import containers are given by the fourth part. Finally, the last term in Equation (3.29) corresponds to the recourse costs for the export containers.

To simplify the expression as given in Equation (3.29), we rewrite the two expectations in that sum, $\mathbb{E}[\min\{d^i, \max\{0, \Phi - x\}\}]$ and $\mathbb{E}[\max\{0, x - \Phi\}]$. First of all, $\mathbb{E}[\min\{d^i, \max\{0, \Phi - x\}\}]$ can be rewritten as

$$\begin{aligned} \mathbb{E}[\min\{d^i, \max\{0, \Phi - x\}\}] &= \int_0^\infty \min\{d^i, \max\{0, t - x\}\} dF(t) \\ &= \int_0^x 0 dF(t) + \int_x^{d^i + x} (t - x) dF(t) + \int_{d^i + x}^\infty d^i dF(t) \\ &= 0 + \left(d^i F(d^i + x) - \int_x^{d^i + x} F(t) dt \right) + d^i \left(1 - F(d^i + x)\right) \\ &= d^i - \int_x^{d^i + x} F(t) dt. \end{aligned}$$

In the derivation above, the expression $\min\{d^i, \max\{0, \Phi - x\}\}$ is split into three parts: if $\Phi \leq x$, it is equal to 0, if $x \leq \Phi \leq x + d^i$ then it equals $\Phi - x$, and if $\Phi \geq d^i + x$ then it takes the value d^i. The first integral is equal to zero, integration by parts can be used to derive the second integral, and the final integral of this equation can be simplified using the fact that $F(\cdot)$ is a cumulative distribution function.

The expression $\mathbb{E}[\max\{0, x - \Phi\}]$ can be rewritten, by changing the order

of integration, as follows:

$$\begin{aligned}\mathbb{E}[\max\{0, x-\Phi\}] &= \int_0^x (x-z)dF(z) = \int_0^x \int_z^x dt dF(z) \\ &= \int_0^x \int_0^t dF(z)dt = \int_0^x F(t)dt.\end{aligned}$$

Combining the two equalities above, Equation (3.29) can be simplified into

$$\begin{aligned}T(x) &= c^e x + c_t^e(d^e - x) + c^i\left(d^i - \int_x^{d^i+x} F(t)dt\right) \\ &\quad + c_t^i\left(\int_x^{d^i+x} F(t)dt\right) + q\int_0^x F(t)dt.\end{aligned}$$

The optimization problem can now be given as

$$\min_x \quad T(x) \tag{3.30}$$

subject to

$$g_1(x) = x - d^e \leq 0 \tag{3.31}$$

$$g_2(x) = -x \leq 0. \tag{3.32}$$

This is a problem with simple recourse and thus the expected recourse costs are given by a function that is convex in x (Birge and Louveaux, 2011). In other words, we can use the Karush-Kuhn-Tucker (KKT) conditions, to calculate the optimal number of export containers on a barge. The KKT conditions are given by the following set of equalities:

$$\begin{aligned}&\frac{dT(x)}{dx} = -\lambda_1\frac{dg_1(x)}{dx} - \lambda_2\frac{dg_2(x)}{dx} \\ &\lambda_1 g_1(x) = 0 \\ &\lambda_2 g_2(x) = 0 \\ &\lambda_1, \lambda_2 \geq 0 \\ &g_1(x), g_2(x) \leq 0,\end{aligned}$$

which can be reformulated as

$$\begin{aligned}&c^e - c_t^e + \left(c_t^i - c^i\right)F\left(d^i + x\right) + \left(c^i - c_t^i + q\right)F(x) \quad = -\lambda_1 + \lambda_2 \\ &\lambda_1\left(x - d^e\right) = 0 \\ &\lambda_2(-x) = 0 \\ &\lambda \geq 0 \\ &(x - d^e) \leq 0 \\ &-x \leq 0.\end{aligned} \tag{3.33}$$

The system of equalities (3.33) above has four different types of solutions. Namely, (i) $\lambda_1 > 0$ and $\lambda_2 > 0$, (ii) $\lambda_1 = 0$ and $\lambda_2 > 0$, (iii) $\lambda_1 > 0$ and $\lambda_2 = 0$, and (iv) $\lambda_1 = \lambda_2 = 0$. A feasible solution for (3.33) with both $\lambda_1 > 0$ and $\lambda_2 > 0$ is only possible in the trivial case in which $d^e = 0$. If $\lambda_1 = 0$ and $\lambda_2 > 0$, then the third equality gives us that $x^* = 0$. Equivalently, if $\lambda_1 > 0$ and $\lambda_2 = 0$, then, because of the second equality, we know that x^* should be d^e. Finally, if $\lambda_1 = \lambda_2 = 0$, then the first equality implies that the optimal x^* should satisfy

$$c^e - c_t^e + \left(c_t^i - c^i\right) F\left(d^i + x^*\right) + \left(c^i - c_t^i + q\right) F(x^*) = 0. \tag{3.34}$$

Although it is not possible to derive an analytical expression for x^* for general distributions, the equality of Equation (3.34) can be solved using, for instance, the Newton-Raphson method. Since the cumulative distribution function $F(\cdot)$ is a non-decreasing function, we know that the expression in Equation (3.34), is also non-increasing in x. Consequently, there is at most one value $x^* \geq 0$ for which the equality in Equation (3.34) holds. Hence, there exists an easy method to find the solution for the system of equalities in (3.33). If

$$\begin{aligned}\frac{dT(0)}{dx} &= c^e - c_t^e + \left(c_t^i - c^i\right) F\left(d^i\right) + \left(c^i - c_t^i + q\right) F(0) \\ &= c^e - c_t^e + \left(c_t^i - c^i\right) F\left(d^i\right) \geq 0,\end{aligned}$$

then $x^* = 0$. Otherwise, the solution to the equality in Equation (3.34) has to be calculated. If that solution is smaller than d^e, then it is the optimal solution, otherwise $x^* = d^e$.

Heuristic Stochastic Programming method

In the previous section, we have given the optimal solution for a simplified problem. There are two aspects in which the problem of the previous section was easier than the problem described in Section 3.4. First of all, the assumption that there is only a single barge visiting a single terminal and second, that the transportation costs are the same for all containers. Nevertheless, the solution to (3.30)-(3.32) can be used as a basis to find a solution to the problem described in Section 3.4. The goal of the SP-based method is to find a constraint for the number of export containers on a barge for a specific terminal.

Let us first focus on terminal k and barge j. Let us assume that we have an estimate for the number of import and export containers on terminal k that could be transported on barge j (i.e., δ_{jk}^i and δ_{jk}^e), an estimate for the costs of shipping an import and an export container from terminal k on barge j (i.e., γ_{jk}^i and γ_{jk}^e), and an estimate for the transportation costs per truck for the import (i.e., $\gamma_{0_k}^i$) and export containers (i.e., $\gamma_{0_k}^e$) on terminal k. Furthermore, we

Algorithm 3.1: Algorithm to find a constraint for the maximum number of export containers for a deep-sea terminal to load on a barge.

Input parameters: δ^i_{jk}, γ^i_{jk} and $\gamma^i_{0_k}$.
for $j = 1, \ldots, m$ **do**
  for $k = 1, \ldots, l$ **do**
    Let $E_k := |\mathcal{E}(k)|$.
    Let the containers $t \in \mathcal{E}(k)$ be numbered $0, 1, \ldots, E_k - 1$ in decreasing order of $c^e_{t0} - c^e_{tj}$.
    Set $p = 0$
    while $p < E_k - 1$ *and* $c^e_{pj} - c^e_{p0} + (\gamma^i_{0_k} - \gamma^i_{jk})F_{jk}(\delta^i_{jk} + p + 1) + (\gamma^i_{jk} - \gamma^i_{0_k} + q_{jk}F_{jk}(p+1) < 0$ **do**
      $p = p + 1$.
    end
    $M_{jk} = p$.
  end
end
Output: M

assume that barge j is the only barge visiting terminal k. Hence, if containers from terminal k are not on barge j, they are transported by truck. Finally, we denote the sum of all export containers for terminal k that are on barge j by X^e_{jk}. Using these assumptions, the total transportation costs for all containers on terminal k can be approximated in the same way as in the optimization problem (3.30)-(3.32). Therefore, we can use an equality similar to the equality in Equation (3.34), to find a solution for X^e_{jk}, namely

$$\gamma^e_{jk} - \gamma^e_{0_k} + \left(\gamma^i_{0_k} - \gamma^i_{jk}\right) F_{jk}(\delta^i_{jk} + X^e_{jk}) + (\gamma^i_{jk} - \gamma^i_{0_k} + q_{jk})F_{jk}(X^e_{jk}) = 0. \quad (3.35)$$

Besides, deciding how many export containers for terminal k will be on barge j, also the specific containers that will be loaded on this barge have to be selected. These two decisions cannot be considered independently. If only one barge is visiting a terminal and only a single export container for a terminal is transported on that barge j, then it is easy to determine which container is the best to transport on barge j. If a container is not transported on barge j, then it is transported by truck.

The container that benefits most from being transported on barge j instead of the truck is the container for which the difference between the truck transportation costs and the barge transportation costs is the highest, i.e., $\tau_j = \arg\max_{t \in \mathcal{E}(k)}\{c^e_{t0} - c^e_{tj}\}$. Therefore, if $X^e_{jk} = 1$, the estimates γ^e_{jk} and $\gamma^e_{0_k}$ are not needed and can be replaced by $c^e_{\tau_j j}$ and $c^e_{\tau_j 0}$. Similarly, if $X^e_{jk} = 2$, the container for which the difference between the truck and barge transportation costs is second largest is chosen. Consequently, it becomes clear that the estimators γ^e_{jk} or $\gamma^e_{0_k}$ are not needed but that the real value c^e_{t0} and c^e_{tj} can be used in Equation (3.35). Similar to the solution method described in Section 3.5.2, we can increase the number of export containers for a terminal that is assigned

to a barge until the equality in (3.35) is satisfied.

In Algorithm 3.1, this procedure is formalized. This algorithm is used to derive a constraint for the maximum number of export containers M_{jk} that can be transported with barge j to terminal k. When determining M_{jk}, we assumed that barge j was the only barge visiting terminal k. Consequently, all containers can be selected for transportation on barge j. It might be that a container on terminal k has extremely high truck transportation costs, and is therefore used in calculating the M_{jk}-value for every barge j. Nevertheless, that container can only be transported on one barge. Hence, even if there are multiple barges visiting terminal k, the number of export containers on barge j to terminal k is always less than M_{jk}. With the constraint from Algorithm 3.1, we get the following deterministic ILP:

$$\min \quad \sum_{t=1}^{n^i} \sum_{j=0}^{m} c_{tj}^i X_{tj}^i + \sum_{t=1}^{n^e} \sum_{j=0}^{m} c_{tj}^e X_{tj}^e \tag{3.36}$$

subject to

$$\sum_{t \in \mathcal{E}(k)} X_{tj}^e \leq M_{jk} \qquad j = 1, \ldots, m \quad k = 1, \ldots, l \tag{3.37}$$

$$\sum_{t \in \mathcal{I}(k)}^{n^i} X_{tj}^i + \sum_{t \in \mathcal{E}(k)} X_{tj}^e \leq \mathbb{E}[\Phi_{jk}] \qquad j = 1, \ldots, m \quad k = 1, \ldots, l \tag{3.38}$$

$$\text{Constraints } (3.2) - (3.5) \text{ and } (3.7) - (3.12). \tag{3.39}$$

If we only add constraint (3.37) to the ILP, then the import containers would not be restricted by the number of moves at a terminal. As a consequence, the solution of the ILP would probably overestimate the number of import containers that can be shipped on a barge. Hence, constraint (3.38) is added to limit the number of import containers shipped on a barge.

3.5.3 Other methods

In Toktas et al. (2006), three methods to solve assignment problems with stochastic capacity constraints are given. We explain below how these methods can be used to solve the problem (3.13)-(3.18).

Expectational method

In the *expectational method*, we replace the stochastic variable Φ_{jk} by its expectation. So the first-stage objective function is modified into

$$\min \quad \sum_{t=1}^{n^e} \sum_{j=0}^{m} c_{tj}^e X_{tj}^e + Q(X^e, Y, \mathbb{E}[\Phi]).$$

Since the expectation is a deterministic value, the problem can be formulated as an ILP, similar to the ILP given in (3.1)-(3.12). The only difference is that constraint (3.6) is replaced by

$$\sum_{t\in\mathcal{I}(k)} X^i_{tj} + \sum_{t\in\mathcal{E}(k)} X^e_{tj} \leq \mathbb{E}[\Phi_{jk}] \quad j = 1,\ldots,m \quad k = 1,\ldots,l.$$

This expectational method is rather naive since it only uses the expectation and no other information on the distribution. Nevertheless, it is a good benchmark to see what one can gain by incorporating more knowledge of the distribution.

Risk-averse trimmed mean method

The *risk-averse trimmed mean method* is almost identical to the expectational method. The only difference is that instead of the expectation, a more conservative estimate for the number of moves is used, which is called the risk-averse trimmed mean. This risk-averse trimmed mean is defined as follows:

$$\mathbb{A}_\rho\left[\Phi_{jk}\right] := \mathbb{E}\left[\Phi_{jk} | F(\phi_{jk}) \leq \rho\right].$$

The interpretation of the risk-averse trimmed mean is that the expectation of the values less than or equal to the ρ-quantile is taken. In the risk-averse trimmed mean method, the constraint (3.6) in the ILP (3.1)-(3.12) is replaced by

$$\sum_{t\in\mathcal{I}(k)} X^i_{tj} + \sum_{t\in\mathcal{E}(k)} X^e_{tj} \leq \mathbb{A}_\rho[\Phi_{jk}] \qquad j = 1,\ldots,m \quad k = 1,\ldots,l.$$

The idea behind the risk-averse trimmed mean method is that taking a more conservative estimate for the number of moves than the expectation results in fewer recourse costs. The parameter ρ can be used to indicate the risk one is willing to take. If $\rho = 0$, then $\mathbb{A}_\rho = 0$ and if $\rho = 1$, then $\mathbb{A}_\rho[\Phi_{jk}] = \mathbb{E}[\Phi_{jk}]$.

Comparative performance evaluation method

The *Comparative Performance Evaluation* (CPE) *method* is, similar to the SAA method, a sample based method. The difference between the CPE and SAA method, is that in the former every scenario is solved independently. Let us have a vector of N scenarios $\bar{\phi}_N := \left(\phi^1, \phi^2, \ldots, \phi^N\right)$. For a single scenario ϕ^n, the deterministic problem (3.1)-(3.12) can (relatively) easily be solved, which gives a solution (X^{en}, Y^n). Using that solution, the value $\mathbb{E}\left[Q(X^{en}, Y^n, \Phi)\right]$ can be estimated by: $\frac{1}{N}\sum_{n'=1}^{N} Q\left(X^{en}, Y^n, \phi^{n'}\right)$. The idea of the CPE method is to select out of the N solutions (X^{en}, Y^n) that are based on a single scenario, the solution that has the lowest estimated costs over all scenarios in $\bar{\phi}_N$. In other

words,

$$x^{CPE} := \underset{n=1,\ldots,N}{\arg\min} \left\{ \sum_{t=1}^{n^e} \sum_{j=0}^{m} c_{tj}^{e} X_{tj}^{en} + \frac{1}{N} \sum_{n'=1}^{N} Q\left(X^{en}, Y^{n}, \phi^{n'}\right) \right\}.$$

3.6 Numerical results

The quality of the solution methods described in the previous section is investigated using numerical experiments. In all methods, a deterministic value replaces the stochastic variable Φ_{jk} , and as a result, the problem could be modeled as an ILP. Using a similar argumentation as in Section 2.5.2, it can be shown that fixing the Y_{jk}-variables for $j = 1, \ldots, m$ and $k = 1, \ldots, l$, results in a variant of the GAP. In the LP relaxation for the ILPs in this chapter, there are, for each barge, at most N_j capacity constraints of the type of constraint (3.6), and one capacity constraint of type (3.4) and one of type (3.5). Hence, also for this chapter's problem, the number of fractional variables in the LP-relaxation is relatively small. We have seen in Chapter 2 that this relaxation (Step 1 of Algorithm 2.1) is close to the optimal solution. In this chapter, we have decided only to solve this relaxation because then the SAA method can use more realizations in the same amount of computation time. In other words, in all solutions methods, we drop the integrality constraints for the X^e-variables, X^i-variables, and Z-variables.

In Section 3.6.1, we describe three different terminal types that are used in the experiments. A terminal type reflects the distribution for the number of moves at a deep-sea terminal. In Section 3.6.2, small instances are solved to investigate the convergence rate of the SAA method to the optimal solution and the quality of the solution produced by the other methods. Large instances are solved in Section 3.6.3. In this section, we focus on the scalability of the SAA method and compare the outcome of the best parameters for the SAA with the SP-based method and the methods discussed in Section 3.5.3. Finally, in Section 3.6.4, we look at the quality of the methods if the number of moves is correlated between barges.

3.6.1 Terminal types

In practice, the congestion is not the same at each deep-sea terminal. Moreover, the way the terminals deal with congestion also differs per terminal. Therefore, we consider three different types of terminals: *predictable*, *unpredictable* and *open-closed* terminals. For each of these terminal types we give a discrete probability distribution for the number of moves that corresponds to the desired behavior.

At a predictable terminal, as the name suggests, the available number of

moves for each barge does not vary much. This terminal might not be congested, or it is conservative in the number of moves it allocates to barges, and with that, it can always have a rather constant number of moves. We model the predictable terminal with a Poisson distribution with expectation λ. On the other hand, at an unpredictable terminal, the number of moves is sometimes minimal and sometimes very large. Based on a limited amount of data for the moves at container terminals, we conjecture that some terminals are of this kind. We model these terminals with a geometric distribution with parameter p and the support $k = 0, 1, \ldots$.

Finally, we consider the open-closed terminals, which are either closed or they are open. If the terminal is closed, not a single container can be loaded or unloaded, but in case the terminal is open, then its moves are somewhat predictable. We model the open-closed terminals with a bimodal distribution that takes with probability $\frac{1}{3}$ the value 0 and with probability $\frac{2}{3}$, the value is distributed according to a Poisson distribution with expectation μ. The difference between an unpredictable terminal and an open-closed terminal is that the first one is accepting barges also if the terminal has very limited time available for loading and unloading. On the other hand, the open-closed terminal accepts a barge visit only if it has enough time to handle many containers.

To investigate the consequences of the level of variability for the number of moves, all distributions will have the same expectation, namely $\mathbb{E}[\Phi]$. For the three different distributions, the variance of the number of moves can be expressed in terms of this $\mathbb{E}[\Phi]$. The variance of the Poisson distribution is $\mathbb{E}[\Phi]$, for the bimodal distribution it is $\frac{1}{2}\left(\mathbb{E}[\Phi]\right)^2 + \mathbb{E}[\Phi]$ and the variance of the geometric distribution is $\left(\mathbb{E}[\Phi]\right)^2 + \mathbb{E}[\Phi]$. Thus, the variance of the geometric distribution is about twice the variance of the bimodal distribution.

All these three distributions are discrete because not a fractional number of containers can be loaded. Nevertheless, the methods described in Section 3.5 also apply to continuous probability distributions. Furthermore, we assume that, for all terminals and barges, the moves have the same distribution. However, the three solution methods described in Section 3.5 can all be applied to an instance with different distributions for every barge and terminal combination.

3.6.2 Small instances

First, we solve ten relatively small instances to investigate the convergence of the SAA solution to the optimal solution and to evaluate the quality of the other methods. The small instances consist of 50 export and 50 import containers that can be transported on two barges with a capacity of 50 TEU. The size of every container is 1 or 2 TEU, and the barge transportation costs for an import or export container are uniformly distributed between 25 and 100. In contrast,

Terminal type		N					
		10	50	100	250	500	1,000
Predictable	Objective function	9,701	9,467	9,427	9,344	9,339	9,310
	Optimality gap (%)	3.6	1.4	0.9	0.5	0.3	0.3
Unpredictable	Objective function	12,705	12,235	12,103	12,044	11,993	11,977
	Optimality gap (%)	9.5	2.3	1.0	0.4	0.1	0.0
Open-closed	Objective function	12,846	12,343	12,210	12,100	12,040	11,923
	Optimality gap (%)	12.8	3.2	2.2	1.1	0.4	0.2

Table 3.3 Average objective function and optimality gap of the SAA solution for the small instances with increasing numbers of scenarios.

the costs of transporting a container per truck are uniformly distributed between 150 and 200. The costs for a recourse action for an export container are 300. The containers are distributed over five different deep-sea terminals, and each barge is allowed to visit three of them. At each terminal, the expected number of moves is 10 for all terminal types.

The number of scenarios in the training set, denoted by N, is varied to evaluate the consequence of the size of the training set. We have used twenty different runs of the SAA method, i.e., $M = 20$, and we have used $N' = 5,000$; in other words, the validation set consists of 5,000 samples. Out of the twenty different solutions, we select the solution with the lowest objective function for the validation set. This solution is a negatively biased estimate for the value of the SAA solution. Thus, another validation set of 5,000 samples is used to obtain an unbiased estimate for the objective function.

In Table 3.3, the average of this objective function and the optimality gap for the SAA solution are given for the three different terminal types and different numbers of scenarios in the training set (N). The optimality gap in this table is calculated by dividing the outcome of Equation (3.27) by the lower bound of the optimal solution $\bar{t}^M_{N'}$. We use for this optimality gap and all remaining optimality gaps $\alpha = 0.025$. For each combination of N and terminal type, the average over the ten instances' optimality gaps is given.

The most important conclusion from Table 3.3 is that for $N = 1,000$, the average optimality gap for all three terminal types is small, namely 0.3% for the predictable terminals, 0.2% for the open-closed terminals and 0.0% for the unpredictable terminals. Hence, if 1,000 scenarios are used, the SAA method can find a solution that is (close to) optimal.

Furthermore, it can be seen that for $N = 10$, the predictable terminal has the smallest optimality gap. This observation can be explained by the fact that the Poisson distribution has the smallest variance. Hence, the scenarios in the training and validation are more similar than for the unpredictable and open-closed terminals. For those two terminal types, a sharp decrease in the optimality gap is seen for an increasing size of the training set.

Terminal type	$\gamma^i_{0k} - \gamma^i_{jk}$ 50		100		150	
Predictable	9,944	(139)	9,946	(138)	10,199	(133)
Unpredictable	12,630	(126)	12,684	(100)	12,824	(87)
Open-closed	12,757	(92)	12,773	(133)	12,780	(96)

Table 3.4 Average objective function and its standard deviation for the 10 small instances for the SP-based method using different settings for $\gamma^i_{0k} - \gamma^i_{jk}$.

It becomes clear from Table 3.3 that the predictable terminal has significantly lower costs than the two other terminal types. In other words, the inland container terminal would greatly benefit from the deep-sea terminals having a more reliable number of moves. On the other hand, the difference in the objective between the unpredictable and the open-closed terminal types is not that large.

Finally, for all three terminal types, the objective function for $N = 1,000$ is about 6% better than the solution found for $N = 10$. The running time for the training phase of the SAA increases if N grows. Nevertheless, for the small instances, even for $N = 1,000$, the training phase was able to finish within ten minutes. The validation phase's running time only depends on M and N' and is independent of N. If $M = 20$ and $N' = 5,000$, the validation phase's running time is about twelve minutes.

Now that we have shown that the solution of the SAA method converges to the optimal solution, it is also interesting to look into the quality of the solutions for the SP-based method and the other three methods. For the SP-based method, three different types of parameters need to be defined, namely γ^i_{jk}, γ^i_{0k}, and δ^i_{jk}. For setting the value of δ^i_{jk}, we count how many import containers are at terminal k available for transportation by barge j. After that, we divide this value by the average number of times a barge visits a terminal. In these small instances, the two barges visit both three terminals. Since there are in total five terminals, the average number of times a barge visits a terminal is $\frac{6}{5}$.

It is important to realize that in Algorithm 3.1, only the difference between γ^i_{jk} and γ^i_{0k} is used. We have decided to perform some numerical experiments to decide upon the best parameter setting for $\gamma^i_{0k} - \gamma^i_{jk}$. In Table 3.4, the average objective function and, between brackets, its standard deviation are given for all three terminal types, and $\gamma^i_{0k} - \gamma^i_{jk}$ is 50, 100, and 150. We see that for all three terminal types, the best results are obtained if $\gamma^i_{0k} - \gamma^i_{jk}$ is set to 50. So in the remaining of this section, we use these parameter settings for the SP-based methods. For the risk-averse trimmed mean method, we have to decide on the value for the parameter ρ. In Toktas et al. (2006), it is shown that the best solutions are obtained if $\rho = 0.8$, so we also use this value.

Terminal type		SAA	EXP	RAT	CPE	SP
Predictable	Objective function	9,310	9,963	10,221	9,983	9,944
	Standard deviation	239	140	216	159	145
	Gap SAA	-	7.1%	9.8%	7.3%	6.9%
Unpredictable	Objective function	11,977	13,051	12,761	13,051	12,630
	Standard deviation	219	182	137	140	140
	GAP SAA	-	9.0%	6.6%	9.0%	5.5%
Open-closed	Objective function	11,923	12,814	12,859	13,009	12,757
	Standard deviation	322	107	144	93	96
	Gap SAA	-	7.6%	7.9%	9.2%	7.1%

Table 3.5 Average objective function and its standard deviation for the 10 small instances for the five solution methods and for the three different terminal types.

In Table 3.5, the objective function for the expectational method (EXP), the risk-averse trimmed mean method (RAT), the CPE method and the SP-based method are compared with the SAA solution. For all methods, we give the average objective function over the ten instances, and the standard deviation of the objective function is given between brackets. The SAA solution in this table is the solution obtained by using $N = 1,000$, and we have seen above that this solution is close to the optimal solution.

The first conclusion to draw from Table 3.5 is that the SP-based method is the method that produces the best solutions, and the worst solutions are from the CPE method. Moreover, the difference between the objective functions for the expectational method and the SAA method is significant, and thus taking the uncertainty for the number of moves is beneficial. The risk-averse trimmed mean is too conservative for the predictable terminals, and it is better to choose the expectation. In contrast, for the unpredictable terminals, the variance of the number of moves is higher, and the risk-averse trimmed mean is performing better than the expectation.

For the open-closed terminals, the difference between the objective function of the expectational method and the SAA solution is with 7.6% only slightly larger than the difference between the objective function of the SP-based method and the SAA solution (7.1%). The fact that these two methods produce solutions with a somewhat similar objective function is mainly a coincidence because the solutions themselves are quite different. The expectational method ships more export containers per barge and fewer export containers per truck than the SP-based method. Given the specific parameters that we have used, the extra recourse costs incurred by the expectational method are comparable with the costs it saves by using fewer trucks.

It should be noted that the standard deviations of the objective functions are substantial, so the conclusion drawn above should be made with some reservations. Moreover, it is remarkable to see that the SAA method has the

largest standard deviation from all methods. For three instances, the SAA method finds solutions that result in a much lower objective function than for the other instances. For these three instances, the other methods do not find these good solutions, and thus the objective function of these methods has a lower standard deviation.

The quality of the SP-based method is comparable to the SAA method for $N = 10$. Nevertheless, the running time of the SP-based method is negligible compared to the SAA method. Only Algorithm 3.1 and a single ILP have to be solved to obtain a solution for the SP-based method, which is all done in about a second, whereas for the SAA method with $N = 10$ the training phase takes three to five seconds and the validation phase twelve minutes.

3.6.3 Large instances

For the small instances, we have shown that the SAA method converges to the optimal solution and that the SP-based method is the best heuristic method. In this section, we investigate the performance of the five methods for larger instances. We consider ten instances that consist of 750 import containers, 750 export containers, and four barges with a capacity of 250 TEU. Similar to the small instances, the containers have a size 1 or 2 TEU and the characteristics of the costs are the same as for the small instances. On top of that, we have added the condition that with probability $\frac{1}{4}$, a container cannot be transported on a barge. With this condition, we model the situation in which containers are not available for transportation on a certain barge because they have not arrived at the deep-sea port yet or have to be at the customer earlier than the arrival of the barge at the inland terminal. The containers are transshipped via ten deep-sea terminals, and each barge is only allowed to visit five of them. The expectation of the number of moves at a terminal is set to 75.

For the small instances, the SAA method's running time is small enough to solve the problem almost to optimality within a reasonable time. Nevertheless, for large instances, the running times become too big for large values of M, N, and N'. In Table 3.6, the running times of the training phase are given when $M = 1$ and for different values of N. For predictable terminals, the SAA method takes much longer than for the two other terminal types. Moreover, the running time for the predictable and unpredictable terminals when $N = 50$ is about twenty-five times larger than for $N = 10$. The running time for the open-closed terminals when $N = 50$ is even about forty times larger than for $N = 10$. For the predictable and open-closed terminals, the running time is about five times larger for $N = 100$ than for $N = 50$.

Since we want to compute a solution in three to four hours, we have to decide not to use $N = 100$ for the predictable terminals. We believe that the

	N		
Terminal type	10	50	100
Predictable	212	5,858	-
Unpredictable	20	534	2,810
Open-closed	7	294	1,454

Table 3.6 Average running time in seconds of the training phase of the SAA method for the large instance, for the three terminal types and different numbers of scenarios.

running time for the predictable terminals is much larger because the variety in the number of moves at a terminal is much lower. Thus, the scenarios are more similar. Consequently, the value of the solutions for visiting different terminals are close to each other. As a result, the branch-and-bound method in the ILP solver needs longer to find the optimal solution.

The running time of the validation phase does not vary much for the different terminal types and is linear in N'. Solving the underlying ILP for a single scenario takes about 0.06 seconds. Hence, if we denote the running time from Table 3.6 by r, the total running time for the SAA method with parameters is M, N and N' is $Mr+0.06MN'$ seconds. We use this formula to create different parameter settings, for which we expect the SAA method to be solved within three to four hours.

For the unpredictable and open-closed terminal types, we have created six different parameter settings. The average value and standard deviation of the objective functions, the optimality gaps, and the running times for these six different parameter settings are given in Table 3.7. Only four parameter settings for predictable terminals are considered because the running time for this terminal type is much larger. The average value and standard deviation of the objective functions, the optimality gaps, and the running times for the predictable terminal type are given in Table 3.8. Similar to the small instances, we use a set consisting of 5,000 scenarios to calculate the objective function in Tables 3.7 and 3.8.

Based on the results from Table 3.7, the parameters $M = 20$, $N = 50$, and $N' = 1,000$ gives the best results for the unpredictable and the open-closed terminal types. For both types of terminals, the objective function for $N = 50$ is substantially lower than for $N = 10$. Increasing the number of SAA runs only decreases the objective function slightly. The difference between $N = 50$ and $N = 100$ is not as big as between $N = 10$ and $N = 50$. In Table 3.3, we have already seen that for the small instances the biggest improvement was also made by N going from 10 to 50. Moreover, for $N = 100$ only four runs of the SAA algorithm could be performed.

At first it might be surprising that the optimality gap for both terminals for $N = 100$, $M = 4$, $N' = 5,000$ is smaller than for $N = 50$, $M = 20$, $N' = 1,000$,

	N	10			50		100
Terminal type	$(M;N')$	(10;5,000)	(20;5,000)	(50;3,500)	(10;5,000)	(20;1,000)	(4;5,000)
Unpredictable	Objective function	169,933	169,465	169,070	166,846	166,181	166,777
	Optimality gap (%)	14.3	12.6	12.3	3.9	4.1	2.7
	Running time (s)	3,293	6,280	10,700	8,380	11,392	15,232
Open-closed	Objective function	191,129	190,908	190,486	187,894	186,898	187,093
	Optimality gap (%)	18.2	16.1	17.7	6.4	5.2	4.2
	Running time (s)	2,971	6,934	9,954	5,708	7,315	10,262

Table 3.7 The average objective function for the ten large instances for the unpredictable and open-closed terminals and for different parameter settings of the SAA method.

	N	10			50
Terminal type	$(M;N')$	(10;5,000)	(20;5,000)	(50;1,500)	(2;5,000)
Predictable	Objective function	104,997	104,979	104,953	104,972
	Optimality gap(%)	0.4	0.3	0.2	0.2
	Running time (s.)	5,032	10,068	15,237	12,290

Table 3.8 The average objective function for the ten large instances for the predictable terminal type and different parameter settings of the SAA method.

but that the value of the objective function is larger. Nevertheless, this can be explained by the fact that the lower bound ($\bar{t}^M_{N'}$) for larger N is stronger. If we would use the lower bound for $N = 100$ for $N = 50$, the optimality gap for the latter will be smaller than for the former.

The best parameters for the predictable terminals are, according to Table 3.8, $N = 10$, $M = 50$, and $N' = 1,500$. The objective function for $N = 50$, $M = 2$, and $N = 5,000$ is only slightly worse. If more runs of the SAA method had been possible, the objective function would probably have been better for $N = 50$. Compared with the unpredictable and open-closed terminal types, a single run of the SAA method's training phase has a longer running time. However, we see in Table 3.8 that the solution produced by the SAA method for the predictable terminals has an optimality gap of only 0.2%, which is much smaller than the optimality gaps for the unpredictable and open-closed terminals.

In Table 3.9, the consequences of different values for $\gamma^i_{0k} - \gamma^i_{jk}$ for the SP-based method are investigated. For the large instances, the best value for the difference between the two gamma values is 150, compared to the small instances for which 50 gave the best results. However, it should be noted that for all three terminal types, the difference between the three different settings for the gamma value is extremely small.

In Table 3.10, the SAA solutions are compared, in a similar fashion as Table 3.5, to the solutions of the expectational method, the risk-averse trimmed mean method, the CPE method, and the SP-based method. Also for the large instances, the SP-based method has, on average, the smallest gap with the SAA solution. However, in contrast to the small instances, the SP-based method is

Terminal type	$\gamma^i_{0k} - \gamma^i_{jk}$ 50		100		150	
Predictable	105,134	(3,914)	105,114	(3,931)	105,099	(3,926)
Unpredictable	167,506	(4,221)	167,165	(4,067)	166,819	(3,993)
Open-closed	188,566	(3,811)	188,596	(3,771)	188,566	(3,811)

Table 3.9 Average objective function and its standard deviation for the 10 large instances for the SP-based method using different settings for $\gamma^i_{0k} - \gamma^i_{jk}$.

Terminal type		SAA	EXP	RAT	CPE	SP
Predictable	Objective function	104,953	105,203	116,210	105,212	105,099
	Standard deviation	4,136	4,112	3,853	4,093	3,926
	Gap SAA	-	0.2%	10.7%	0.2%	0.1%
Unpredictable	Objective function	166,181	176,177	168,362	178,326	166,819
	Standard deviation	3,997	3,658	3,828	3,614	3,993
	Gap SAA	-	6.0%	1.3%	7.3%	0.4%
Open-closed	Objective function	186,898	190,064	187,040	191,731	188,566
	Standard deviation	3,918	3,717	3,949	4,159	3,811
	Gap SAA	-	1.7%	0.1%	2.6%	0.9%

Table 3.10 Average objective function over ten large instances for the five solution methods and for the three different terminal types.

not for all three terminal types the best: for the open-closed terminal, the risk-averse trimmed mean method performs better. Nevertheless, the risk-averse trimmed mean method gives for the predictable terminal type a solution that is 10% worse than the SAA method. Hence, the SP-based method is more robust for different terminal types than the risk-averse trimmed mean method.

For the predictable terminal types, we see that the expectational method, the CPE method, and the SP-based method all produce solutions that are close to the best SAA solution that is found. Therefore, we may conclude that it might not be that beneficial to include the stochasticity of the number of moves into account: only using the expectation already produces excellent results for the predictable terminals. That is mainly due to the fact that if the moves are Poisson distributed, then it will hardly happen that the number of moves is exceeding the number of export containers loaded on the barge and thus few recourse costs have to be paid. On the other hand, for the unpredictable and open-closed terminal types, the expectational method results in much recourse costs. The risk-averse trimmed mean method gives lower costs because fewer export containers are loaded using this method.

For unpredictable and open-closed terminals, one should keep in mind that the SAA solutions still had an optimality gap of a few percentages. Hence, for the methods given in Table 3.10, the difference with the optimal solution is likely to be bigger than the reported difference with the SAA solution. Another thing to keep in mind is that the SP-based method's running time is only a couple of

seconds. Hence, the SP-based method is an excellent scalable alternative for the SAA method.

A final observation is that, similar to the small examples, predictable terminals give by far the lowest costs. However, although for the small instances the unpredictable and open-closed terminals had almost the same value for the objective function, for the large instances, the objective function for the unpredictable terminals is much lower than the open-closed terminals. A possible explanation for this could be that the expectation for the number of moves for the small and large instances differ. For the open-closed terminals, the number of moves is either zero, or Poisson distributed. Since the expectation for the number of moves for the small instances is lower than for the large instances, the Poisson distribution for the small instances also has a lower expectation. Consequently, the difference between being open or closed is less for small instances than for large instances. Although the variance for the unpredictable terminals is higher than for the open-closed terminals, the realizations are more evenly distributed over the support of the probability distributions. Hence, it is possible to have a better trade-off between recourse and transportation costs.

3.6.4 Correlated instances

In this section, we investigate the performances of the five methods when the number of moves is correlated. The instances are the same as used for the small instance in Section 3.6.2, but the difference is that there is a positive correlation for the number of moves at terminal k between the two different barges. Let Ψ_{1k} and Ψ_{2k} be the correlated number of moves at terminal k for these two barges. Given the two uncorrelated random variables Φ_{1k} and Φ_{2k} and a value $\alpha \in [0, 1]$, the variables Ψ_{1k} and Ψ_{2k} are defined as follows:

$$\begin{aligned} \Psi_{1k} &= \Phi_{1k} && k = 1, \ldots, l \\ \Psi_{2k} &= \lceil \alpha \Phi_{1k} \rceil + \lfloor (1-\alpha)\Phi_{2k} \rfloor && k = 1, \ldots, l. \end{aligned}$$

The value for α is a parameter to determine the amount of correlation: if α is zero, then Ψ_{1k} and Ψ_{2k} are uncorrelated and if $\alpha = 1$, then the variables Ψ_{1k} and Ψ_{2k} always have the same value. We assume that Φ_{1k} and Φ_{2k} are drawn from the same distribution. Consequently, the variables Ψ_{1k} and Ψ_{2k} are uniquely determined by a value for α and a distribution for Φ_{jk} for $j = 1, 2$. Moreover, the expectation of all four random variables Φ_{1k}, Φ_{2k}, Ψ_{1k}, and Ψ_{2k} is the same. It is trivially to see that this statement is true for the first three random variables and for the expectation of Ψ_{2k} we have

$$\begin{aligned} \mathbb{E}[\Psi_{2k}] &= \mathbb{E}[\lceil \alpha \Phi_{1k} \rceil + \lfloor (1-\alpha)\Phi_{2k} \rfloor] \\ &= \mathbb{E}[\lceil \alpha \Phi_{1k} \rceil] + \mathbb{E}[\Phi_{2k}] - \mathbb{E}[\lceil \alpha \Phi_{2k} \rceil] \\ &= \mathbb{E}[\Phi_{2k}]. \end{aligned}$$

The distribution of Ψ_{1k} is the same as used for Φ_{1k}, but Ψ_{2k} follows a different distribution. To derive the distribution for Ψ_{2k} we first need the distributions for $\lceil \alpha\Phi_{1k} \rceil$ and $\lfloor (1-\alpha)\Phi_{2k} \rfloor$. The probability density function for $\lceil \alpha\Phi_{1k} \rceil$ is as follows:

$$\begin{aligned}\mathbb{P}\left(\lceil \alpha\Phi_{1k} \rceil = \phi\right) &= \mathbb{P}\left(\phi - 1 < \alpha\Phi_{1k} \leq \phi\right) \\ &= \mathbb{P}\left(\Phi_{1k} \leq \frac{\phi}{\alpha}\right) - \mathbb{P}\left(\Phi_{1k} \leq \frac{\phi - 1}{\alpha}\right) \\ &= F\left(\frac{\phi}{\alpha}\right) - F\left(\frac{\phi - 1}{\alpha}\right).\end{aligned}$$

Furthermore, the probability density function for $\lfloor (1-\alpha)\Phi_{2k} \rfloor$ equals

$$\begin{aligned}\mathbb{P}\left(\lfloor (1-\alpha)\Phi_{2k} \rfloor = \phi\right) &= \mathbb{P}\left(\phi \leq (1-\alpha)\Phi_{2k} < \phi + 1\right) \\ &= \mathbb{P}\left(\Phi_{2k} < \frac{\phi + 1}{1-\alpha}\right) - \mathbb{P}\left(\Phi_{2k} < \frac{\phi}{1-\alpha}\right) \\ &= F\left(\left\lceil \frac{\phi + 1}{1-\alpha} - 1 \right\rceil\right) - F\left(\left\lceil \frac{\phi}{1-\alpha} - 1 \right\rceil\right) \\ &= F\left(\left\lceil \frac{\phi + \alpha}{1-\alpha} \right\rceil\right) - F\left(\left\lceil \frac{\phi + \alpha - 1}{1-\alpha} \right\rceil\right).\end{aligned}$$

The variable Ψ_{2k} is a convolution of $\lceil \alpha\Phi_{1k} \rceil$ and $\lfloor (1-\alpha)\Phi_{2k} \rfloor$ and thus the probability density function for Ψ_{2k} is given by

$$\begin{aligned}\mathbb{P}\left(\Psi_{2k} = \psi\right) &= \sum_{\phi=0}^{\psi} \mathbb{P}\left(\lceil \alpha\Phi_{1k} \rceil = \phi\right) \mathbb{P}\left(\lfloor (1-\alpha)\Phi_{2k} \rfloor = \psi - \phi\right) \\ &= \sum_{\phi=0}^{\psi} \left(F\left(\frac{\phi}{\alpha}\right) - F\left(\frac{\phi - 1}{\alpha}\right)\right) \\ &\quad \left(F\left(\left\lceil \frac{\psi - \phi + \alpha}{1-\alpha} \right\rceil\right) - F\left(\left\lceil \frac{\psi - \phi + \alpha - 1}{1-\alpha} \right\rceil\right)\right).\end{aligned}$$

Knowing the probability distribution for Ψ_{1k} and Ψ_{2k}, the expectational method, the risk-averse trimmed mean method, and the SP-based method can be applied to the correlated instances. Furthermore, it is also possible to generate correlated samples, and thus the SAA and CPE methods can also be used for the correlated instance.

In Table 3.11, the results for the five different solution methods for the predictable and unpredictable terminals and $\alpha = 0.25, 0.5$, and 0.75 are given. We have chosen not to use the open-closed distribution because the realizations do longer follow the idea of being either 0 or having a high value in the correlated distribution. The main conclusion for the correlated samples is that the

Terminal type	α		SAA	EXP	RAT	CPE	SP
Predictable	0.25	Objective function	9,753	9,968	10,199	10,012	9,913
		Standard deviation	253	213	138	159	201
		Gap SAA	-	4.6%	2.2%	2.7%	1.6%
	0.50	Objective function	9,764	9,960	10,166	9,996	9,940
		Standard deviation	282	136	197	150	142
		Gap SAA	-	2.0%	4.1%	2.4%	1.8%
	0.75	Objective function	9,698	9,964	10,193	9,959	9,909
		Standard deviation	231	142	215	133	203
		Gap SAA	-	2.7%	5.1%	2.7%	2.2%
Unpredictable	0.25	Objective function	12,432	13,049	12,756	12,976	12,735
		Standard deviation	172	173	134	126	95
		Gap SAA	-	5.0%	2.6%	4.4%	2.4%
	0.50	Objective function	12,402	13,106	12,801	13,137	12,785
		Standard deviation	172	150	116	184	124
		Gap SAA	-	5.7%	3.2%	5.9%	3.1%
	0.75	Objective function	12,492	13,078	12,776	12,914	12,757
		Standard deviation	173	176	126	102	88
		Gap SAA	-	4.7%	2.3%	3.4%	2.1%

Table 3.11 Average objective function over the 10 correlated instances for the five solution methods and the predictable and unpredictable terminal types.

SP-based method is also the method that is the closest to the SAA method. Nevertheless, for the unpredictable terminals, the risk-averse trimmed mean method is performing almost as good. Furthermore, the value of α does not have a big influence on the methods' performance.

The only difference between the small instances and the correlated instances is the distribution of the moves. If we compare the results of Table 3.11 with the results for the small instances in Table 3.5, we see that the value of the objective for the solution of the SAA method is slightly larger for the correlated instances than for the small instances. However, for the other methods, the objective function value for the correlated instances is about the same as for the small instances. A possible explanation could be that in the SAA method, it is not explicitly defined that the visits for a terminal by the two barges are correlated. The method can only learn that from the data, which is also the case for the CPE method. However, the other three methods can use the correlated distribution function.

3.7 Conclusion

In this chapter, we have studied the question: how to ship containers from and towards congested deep-sea terminals? In this problem, the number of containers that is allowed to be loaded and unloaded at a deep-sea terminal is unknown when the export containers are loaded on the barges. We have modeled this problem as a two-stage stochastic program with recourse. We have presented an SAA method that can solve small instances almost to optimality.

Nevertheless, the SAA method is not scalable, and thus, for larger instances, it takes too long to produce almost optimal solutions.

In practice, fast solutions are required, so we have also developed a fast heuristic method. The idea behind this method is that stochastic programming can be used to find the optimal solution for a simplified problem. The characteristics of this optimal solution to the simplified problem are used in the SP-based method that we have presented to solve the original problem. We have compared the results of this SP-based method with three methods for general stochastic assignment problems: the expectational method, the risk-averse trimmed mean method, and the comparative performance evaluation method. We have tested these methods for three different terminal types: predictable, unpredictable, and open-closed terminals. Moreover, we have also used three different types of instances: instances with a small number of containers and an uncorrelated number of moves, instances with a large number of containers and an uncorrelated number of moves, and instances with a small number of containers and a correlated number of moves.

The SAA method produces for small instances almost optimal solutions. For the larger instances and certain terminal types, the solution that is produced by the SAA method is about 2 to 4% from the optimal solution. The SP-based method performs almost always better than the other three methods. For the large instances, it can compute in a couple of seconds a solution that is less than 1% worse than the SAA method's solution, which requires a couple of hours of computing time. We have also seen that the performance of the five methods is not much different if the number of moves at a terminal is correlated between all the barges. All in all, if the planning is allowed to take a few hours, the SAA method is the best method to use, but the SP-based method is shown to be a good alternative for a faster solution. Concerning the different terminal types, we conclude that predictable terminals result in the lowest cost. For the small instances, the difference between the unpredictable and open-closed terminals are not that large. However, for the large instances, we conclude that the inland terminal has lower costs if the terminal is unpredictable than if the terminal is of the type open-closed.

The SAA method was not able to solve the large instances to optimality. We have used a simple implementation of the SAA method. In further research, it could be investigated if a more advanced decomposition to solve the SAA method will result in faster solutions for the SAA method. Currently, the SAA method takes up to a couple of hours to solve the problem, whereas the SP-based method only uses a couple of seconds. However, in practice, one might be willing to wait a couple of minutes for a good solution. Hence, a direction of further research could be to find another method in which the running time and the solution quality are in between the SAA method and the SP-based method.

In the current formulation of the problem, we have made some simplifying assumptions which could also be relaxed in further research. Including train transport as a mode of transportation that is cheaper than truck transportation, but more expensive barge transportation could be an interesting option. Moreover, including the route a barge has to sail to visit the terminals is also an option. In that case, one also has to make sure that the capacity of the barge is not violating between two deep-sea terminals. Moreover, solving the resulting ILP formulation is expected to take longer. Finally, one could also include that the number of moves for the first barge is revealed before the export containers for the other barges has to be loaded. The resulting problem would be a multi-stage stochastic problem.

4

Minimum cost paths with stochastic travel times and overbooking

4.1 Introduction

In this chapter, we shift our focus from problems in which many containers are transported to a problem with only a single container. The previous two chapters determined the best day and mode of transportation for a container, such that the total costs of transporting an entire set of containers was minimized. Such a situation is only possible if a shipper is willing to be flexible in the way a container is transported. It is likely that by minimizing the total costs, a part of the containers is not transported in the cheapest possible way. Therefore, the inland terminal, or another company that makes the consolidated planning, has to charge a fixed price independent of the way a container is transported. However, sometimes a shipper would like to decide for a specific container how it is transported.

In this chapter, we study a problem faced by a shipper that ships specialized cargo on a global scale. These specialized types of cargo could be pharmaceuticals. For instance, medicines need to be transported in a specific temperature range d. The goods are often shipped in cooling containers to control the temperature, but these containers only cool for a limited amount of time. When the shipment arrives after this period, it might not be at the right temperature. For these types of shipments, the shipper wants to specify beforehand the entire route and the conditions under which the goods are transported. Although the research for this chapter originated from this context, the models and algorithms presented in this chapter generalize also to the context of hinterland container transportation.

Two main factors influencing the route choice of a shipment are the *costs* and *duration*. Usually, one is looking for the cheapest option to transport freight such that the arrival on at the destination is on time. However, the travel time is

This chapter is based on B.G. Zweers and R.D. van der Mei. Minimum costs paths in intermodal transportation networks with stochastic travel times and overbookings. *Submitted*, 2020.

to a large extent stochastic. A vehicle might be delayed or a leg of the trip could be overbooked. Therefore, it is impossible to guarantee that a shipment will arrive on time. Possible options to deal with the travel time's stochasticity are to enforce the expected arrival time of the shipment to be before its deadline, or to put a penalty on late arrival and minimize the expected costs. These methods might work fine if a shipper transports many loads, but they are troublesome if only incidentally goods are transported. In these types of problems, it is more suitable to consider the *probability of on-time arrival*.

Since the shipper does not operate its own fleet, he or she has to obey the predefined schedule of the carrier. This schedule results in large variations in the duration of a shipment (Ziliaskopoulos and Wardell, 2000). Usually, there is some time in the schedule between consecutive departures. Hence, if there is a delay in one point of the transportation chain and a departure is missed, then the next departure is often much later. As a result, the planned departure of the next leg might also be missed, in which the delay could further propagate through the network. In practice, it could also happen that a leg is overbooked. In that case, the shipment can at best be transported in the next departure. The overbooking probability usually increases if one is behind the planned schedule, in which case the delay only gets worse.

In this freight network, we are interested in constructing Pareto-optimal solutions in which the probability of arrival before the deadline is compared with the costs of a route. With such a Pareto-front, the shipper can choose how much risk he or she is willing to accept to transport goods at a specific price. The number of routes grows exponentially with the number of nodes in a network and there do not exist efficient optimal algorithms. Hence, for larger-sized networks, we need to consider heuristics to find the most cost-efficient route given a certain *acceptance probability for risk*.

The contribution of this chapter is three-fold. First, we model a problem faced by a company delivering an online tool for shippers as a shortest path problem with stochastic travel times and overbooking probabilities. The goal of this problem is to find the cheapest path for which the probability of arrival before a deadline is above a certain threshold. Second, we give an optimal algorithm based on dynamic programming. Finally, we propose a heuristic in which the stochastic travel times are replaced by deterministic values that are a function of the risk one is willing to take. These deterministic values are used in an ILP formulation.

The remainder of this chapter is organized as follows. In Section 4.2, a description of the relevant literature is given. After that, we describe in Section 4.3 in detail the problem that we are solving. This problem formulation is translated into a mathematical model in Section 4.4. In Section 4.5, the

optimal algorithm and heuristic are presented to solve the model described in Section 4.4. The quality of this heuristic is investigated in Section 4.6. We conclude this chapter in Section 4.7.

4.2 Literature review

In this section, we focus first on routing problems in multimodal networks. However, these problems are mainly deterministic. Since our problem is stochastic, we also discuss the relevant literature concerning stochastic shortest paths. Finally, we review some works regarding risk measures that will be used in our heuristic.

According to Chang (2008), routing problems in intermodal transport networks have three important characteristics. First, they deal with multiple objectives, such as travel time and costs. Second, the schedules and delivery times must be included to avoid a mismatch in practice. Finally, the calculation of costs is complicated because it might be dependent on the weight or volume transported. In this work, we only focus on the first two points. Ziliaskopoulos and Wardell (2000) add that one should also account for delays at switching locations. We implicitly do that by including overbookings.

The multiple objectives can be included as a weighted sum in the objective function, or it can be decided to put one or more of the targets in the constraint. In Cho et al. (2012), the latter is done by constructing Pareto-optimal solutions regarding the travel time and transportation costs. They solve a weighted constrained shortest path problem using a dynamic programming formulation. In Gromicho et al. (2011), a problem faced by a logistic service provider is studied in which the goal is to find the k-cheapest routes that have to be found given time restrictions. This problem is also modeled as a recourse constraint shortest path and is solved using a two-stage variant of Dijkstra's algorithm.

A weighted sum of the travel time and the transportation costs is minimized in Chang (2008). He enforces time-windows for the moment a path should arrive in a node and solve the resulting problem using a Lagrangean relaxation. In Yang et al. (2011), a goal programming approach is used to minimize the weighted sum of transportation cost, transit time, and transit variability of an intermodal route. This problem is entirely deterministic because the transit variability is assumed to be a given constant. In multi-objective problems, the travel time is also sometimes assumed to be time-dependent. A multi-criteria time-dependent shortest path problem is studied in Androutsopoulos and Zografos (2009). In this problem, the scheduled departure time of an arc is fixed, and for every node, there is a strict time window for which the route should visit that node. In Ziliaskopoulos and Wardell (2000), a deterministic time-dependent shortest path problem is studied with a delay at the nodes. In

Chang et al. (2007), this problem is extended by not only minimizing the travel time but also the costs that are associated with that path.

In most classical shortest path problems with stochastic travel times, the goal is to find the path with the shortest expected duration (see, e.g., Hall (1986), Fu and Rilett (1998), and Miller-Hooks and Mahmassani (2000)). However, there is also some literature on stochastic shortest path problems in which there is an arrival deadline. One concept that is applied in this context is *stochastic dominance*. A distribution stochastically dominates another distribution if, for every possible value in its domain, its cumulative distribution function is at least as large as that of the other distribution. Zhang and Homen-de-Mello (2017) and Nie et al. (2012) study a problem in which the goal is to find a path that minimizes the earliness and lateness and that stochastically dominates a benchmark path. In Nie et al. (2012), this problem is solved using a dynamic programming formulation, and in Zhang and Homen-de-Mello (2017) a Sample Average Approximation method is proposed that can solve larger instances. In Chang et al. (2005), an algorithm is presented to find non-dominated paths with multiple time-dependent stochastic attributes.

General linear programming formulations in which the stochastic constraints have specific properties have also been studied. For instance, Cheng and Lisser (2012) consider a linear program with multiple stochastic constraints in which all random variables are normally distributed. Under this assumption, the problem can be approximated by a second-order conic program. Another example is the work of Luedtke et al. (2010) in which an integer linear program formulation is proposed for the situation in which only the right-hand side of the inequality is random and has a finite distribution.

In Häme and Hakula (2013), a problem in a public transit network is studied in which the goal is to find a path that arrives on time with maximum probability. This problem is solved to optimality using a Markov decision process. The main difference between this problem and our problem is that the one in Häme and Hakula (2013) can be dynamically adjusted. In a situation where a person is traveling, this is realistic, but it is often not applicable to freight transportation.

We conclude this section by reviewing the literature on *risk measures*. By a risk measure, we mean a function that maps a stochastic variable to a real value. A risk measure can be used to compare multiple stochastic variables. In Cominetti and Torrico (2016), multiple risk measures for shortest paths are considered. As they point out, most risk measures do not meet the *additive consistency property*. This property states that if, under a particular risk measure, one stochastic variable is preferred to another stochastic variable, then if to both these variables the same independent random variable is added, the preference relation should not change. They state that the only risk measure

that has this property is the *entropic risk measure*.

An entropic risk measure that has been used before in shortest path problems is the *certainty equivalent under exponential disutility* (Jaillet et al., 2016; Zhang and Tang, 2018). In Zhang and Tang (2018), this measure is applied in the context of a public transit network; their goal is to find a route that minimizes the certainty equivalent under exponential disutility, given that the arrival is before a deadline. Jaillet et al. (2016) study general routing problems for which they put constraints on the certainty equivalent under exponential disutility. They show how this framework can be applied in the case of distributionally robust optimization. As a special case, they solve a shortest path in which the least risky route with respect to the certainty equivalent under exponential disutility has to be found such that the arrival is before the deadline.

Markowitz (1952) proposes the *expectation-variance* (EV) risk measure. In this risk measure, the weighted sum of the expectation and variance of a random variable is taken. The more weight that is given to variance, the more risk-averse the outcome is. A disadvantage of this method is that it is not monotone, which means that it could be the case that one stochastic variable is almost surely dominated by another, but that the EV risk measure prefers the latter (Cominetti and Torrico, 2016). In Hutson and Shier (2009), a more general function of the expectation and the variance is applied to a shortest path problem. The function they minimize is a sum of a convex function of the mean and a concave function of the variance.

4.3 Problem formulation

We study a problem of a shipper who needs to ship goods through an intermodal network. The shipper is a small company that does not have its own fleet and needs to use the scheduled transport of carriers. Possible transportation modes are airplanes, ships, trains, and trucks. The shipper has a given deadline at which the shipment should be at its destination. This deadline could be imposed by the customer that requires its shipment to arrive before a certain moment, but it could also be caused by the packaging material that is used. For instance, some packaging that is used to cool temperature sensitive material stops working after a certain amount of time. A complete route is booked at the carrier, so there is no option to change it during the transshipment. The shipper is looking for the route of the shipment that has minimal costs but also satisfies that the shipment arrives on time with a certain probability.

We assume that for each leg of the transportation network the costs are known. The total costs of a route are simply the sum over all the legs it contains. Furthermore, we assume that we know for each leg the distribution of its duration. On top of that, these stochastic variables are independent of

each other. Nevertheless, there is some implicit dependency within a route. Each leg has a planned departure time associated with it, which is the first transportation possibility for that leg. Moreover, the next departure moments are also given. Consequently, if a shipment arrives after its planned departure at a hub, it can only leave that hub at the next departure time at the earliest. This later departure could have the effect that the shipment also has to wait for a more extended period at the next hub.

As the shipper is only shipping a single or a few containers through the network, we are not concerned with the capacity of a transportation link. The carrier decides if there is enough capacity left in each leg of the transportation chain to transport the shipment. However, as no-shows occur frequently, air carriers tend to overbook a flight. So it might be the case that a shipment cannot be transported on the planned time but that it will be shipped at a later moment.

We assume that for each leg in the transportation network, we know the probability of how many departures a shipment has to miss after its arrival. This probability depends on the planned departure. If the shipment arrives before its scheduled departure at a hub, the expected number of missed departures is less than if the shipment arrives after its planned departure. Moreover, the probability could also depend on the *booking classes* offered by the carrier. Standard booking classes are the cheapest option, but premium booking classes have the advantage of having a higher priority when the leg is overbooked. So for the same physical route, there could be a more expensive option that has a larger probability of arriving on time.

In general, a shipper does not have a fixed value for the on-time arrival probability of the shipment. Consider a situation in which one route has a slightly larger chance of arriving too late at the destination than another route, but the costs of the former path are only a fraction of the latter. In this situation, most shippers will be inclined to take the riskier but cheaper route, but the shipper should make that decision. Therefore, the goal of this problem is to present a Pareto-front in which the probability of on-time arrival is compared with the costs of a route.

4.4 Mathematical model

In this section, we present a mathematical model for the problem described in Section 4.3. The notation that is used in this model is summarized in Table 4.1. We model the intermodal transportation network as a directed acyclic graph $\mathcal{G} := (\mathcal{V}, \mathcal{A})$ with node set $\mathcal{V}$ and arc set $\mathcal{A}$. Let n denote the number of nodes in a graph. The graph has one source node and a destination node. We will number the nodes in such a way that the source node gets number 1 and

n	Destination node
c_{ij}	Costs for arc (i,j)
d_{ij}	Scheduled departure for arc (i,j)
f_{ij}	Frequency of departures on arc (i,j)
T	Deadline for arrival at node n
P	Path going from node 1 to node n
$F_P(\cdot)$	Cumulative distribution function of the arrival time at node n
Φ^0_{ij}	Stochastic variable for the travel time on arc (i,j)
Φ^1_{ij}	Stochastic variable for the number of departures missed before arc (i,j) is traversed if arrival at node i is before d_{ij}
Φ^2_{ij}	Stochastic variable for the number of departures missed before arc (i,j) is traversed if arrival at node i is after d_{ij}
$\underline{\Phi}$	Lower limit of domain of stochastic variable Φ
$\bar{\Phi}$	Upper limit of domain of stochastic variable Φ
β	Risk acceptance parameter
$\mathcal{V}$	Set of all nodes
$\mathcal{A}$	Set of all arcs
$\mathcal{P}$	Set of all paths from node 1 to node n

Table 4.1 Overview of the notation used in Chapter 4.

destination node gets number n. Furthermore, since the graph is acyclic it is possible to number the nodes such that there are no arcs between (i,j) if $j < i$. It could be possible that there are multiple nodes that correspond to a single hub. For instance, if the overbooking probability depends on the bookings class, one needs to make a node for every combination of hub and bookings class.

Let $\mathcal{P}$ be the set of all possible paths between node 1 and node n and we denote a single path by $P \in \mathcal{P}$. An arc (i,j) has associated costs of c_{ij}, and the total costs of all arcs in path P is denoted by $c(P)$. Moreover, an arc (i,j) has a stochastic variable Φ^0_{ij} for the time needed to traverse that edge and a scheduled departure time d_{ij}. For the sake of simplicity, we assume that there is a fixed time interval between the departures after d_{ij}. We call that interval between consecutive departure the *frequency* of an edge and denote it by f_{ij}. Note that our approach would also work if the departure times do not follow a specific pattern but are given beforehand. We assume that all departure times and frequencies are integers. This is realistic from a practical perspective because these times usually have a certain precision, for instance, the planned departure of a flight is usually only given with a precision of five minutes. Moreover, this assumption will be useful in computing the on-time arrival probability, as we will see in Section 4.5.1.

If the shipment arrives at node i before the scheduled departure time d_{ij}

of arc (i, j), then the number of departures it has to miss because there is no capacity left is the stochastic variable Φ_{ij}^1. However, if the shipment arrives after the scheduled departure d_{ij} at node i, then random variable Φ_{ij}^2 denotes the number of departures that is missed. In practice the expectation of Φ_{ij}^2 is larger than the expectation of Φ_{ij}^1. The cumulative distribution function for the arrival time at node n using path P is given by $F_P(\cdot)$. Furthermore, we are given a deadline $T > 0$ and a *risk acceptance threshold* $\beta = [0, 1]$. The risk acceptance threshold is the probability for which we accept a late arrival. The larger the value of β the more risk one is willing to accept. The goal is to find a path with minimal cost such that the probability that the arrival at node n is after T is less than or equal to β. In other words, the problem we need to solve is the following:

$$\min_{P \in \mathcal{P}} \quad c(P) \tag{4.1}$$

subject to

$$1 - F_P(T) \leq \beta \qquad\qquad \beta \in [0, 1]. \tag{4.2}$$

A Pareto-front can be constructed by varying the values of β. The larger the value of β, the cheaper the path will be. So if we start with a value for β equal to 1, the cheapest feasible route is found. Let us denote that path by P^*. After that, we calculate for path P^* the probability that it will arrive at node n after time T. In other words, we find the value for β for which constraint (4.2) holds with equality for path P^*. If that value of β is found, we update the value of β in constraint (4.2) such that it is just slightly lower. Consequently, P^* is no longer feasible and a new path is found with higher costs and lower probability of arriving too late. We repeat the entire procedure until there is no feasible solution anymore.

Running example

We use a simple network to illustrate the problem and solution methods. This network consists of four nodes and is given in Figure 4.1. On each arc, two sets of three numbers are given. The first set consists of the parameters for the departure. The first value represents the planned departure time, and the second one is the frequency of the departures on that arc. Finally, the third number, p_{ij}, is the overbooking probability on that arc. For simplicity, we assume that this is the same whether the shipment arrives before or after the planned deadline. Moreover, it also does not change if more departures have already be missed. Hence, the number of missed departures Φ_{ij}^1 is geometrically distributed with parameter $1-p_{ij}$ and 0 is in the domain of this random variable. In other words, the probability of missing k departures on arc (i, j) is given by

$$\mathbb{P}\left(\Phi_{ij}^1 = k\right) = p_{ij}^k(1 - p_{ij}) \qquad\qquad k = 0, 1, \ldots$$

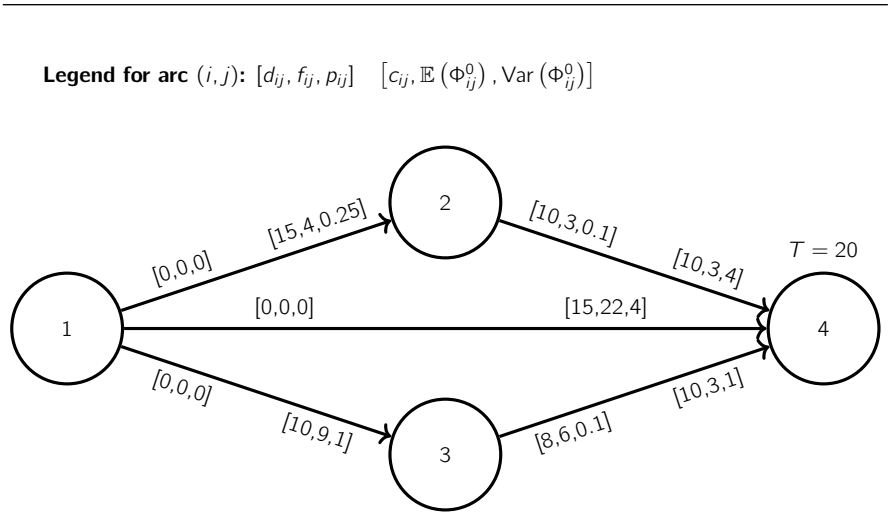


Figure 4.1 Toy example of a network to illustrate the model and the solution methods.

All departures from node 1 leave at time 0 and have no possibility of overbooking, thus the shipment will always leave that node at time 0. The second set of three parameters are concerned with the actual transportation on that arc. The first parameter represents the costs of that leg, the second the expected travel time, and the third value corresponds with the variation of the transportation time. All transportation time will follow the gamma distribution. Finally, the deadline of arrival at node 4 is 20.

4.5 Solution method

In this section, we discuss solution methods to solve the problem presented in Section 4.4. We first explain, in Section 4.5.1, how we compute the arrival distribution at the destination in the intermodal network. This method can be used in an optimal algorithm which is described in Section 4.5.2. However, the running time of this method grows, in the worst case, exponentially in the number of nodes in a network. Therefore, also a heuristic method is described in Section 4.5.3. The high-level idea of this heuristic is to replace the stochastic variable by deterministic risk measures.

4.5.1 Computing arrival and departure distributions

Consider a path in which l nodes are visited, and denote this path by $P = (p_1, p_2, \ldots, p_l)$. The first step in solving the problem (4.1)-(4.2) is to compute $F_P(T)$. Although evaluating whether a sum of random variables is less than a certain value is in general intractable (Khachiyan, 1989), we can exploit the fact

that we made the assumption that all departures are integral values. Hence, the distribution of the departures can be seen as a discrete distribution. However, as the travel time is continuous, the arrival distribution at the next node will also be continuous. Nevertheless, this continuous distribution can be assumed to be discrete as well because, as the departures only occur at discrete moments, we can round up all fractional arrivals to the nearest integer.

Let us denote the probability mass function of the arrival time and the departure at node p_j by, respectively $g_j^A(\cdot)$ and $g_j^D(\cdot)$. We assume that the shipment is available at time 0 at node 1, so $g_1^A(0) = 1$. Then the arrival and departure distributions for the others nodes can be computed by the following three equations:

$$g_j^D\left(d_{j,j+1}\right) = \sum_{x=0}^{d_{j,j+1}} g_j^A(x)\mathbb{P}\left(\Phi_{j-1,j}^1 = 0\right) \qquad j = 1, \ldots, l-1$$

$$\begin{aligned} g_j^D\left(d_{j,j+1} + kf_{j,j+1}\right) = &\sum_{x=0}^{d_{j,j+1}} g_j^A(x)\mathbb{P}\left(\Phi_{j-1,j}^1 = k\right) \\ &+ \sum_{i=0}^{k-1} \sum_{x=d_{j,j+1}+if_{j,j+1}+1}^{d_{j,j+1}+(i+1)f_{j,j+1}} g_j^A(x)\mathbb{P}\left(\Phi_{j-1,j}^2 = i\right) \\ &j = 1, \ldots, l-1 \qquad k = 1, 2, \ldots \end{aligned}$$

$$\begin{aligned} g_{j+1}^A(x) = \sum_{k=0}^{\infty} \Bigg(& g^D(d_{j,j+1} + kf_{j,j+1}) \\ & \mathbb{P}\left(x - d_{j,j+1} + kf_{j,j+1} - 1 < \Phi_{j,j+1}^0 \leq x - d_{j,j+1} + kf_{j,j+1}\right) \Bigg) \\ & j = 1, 2, \ldots, l-1 \qquad x = 0, 1, \ldots. \end{aligned}$$

In the first equation, the probability of departing at the planned departure from a node is calculated. For this to happen, the arrival at that node should be before the planned departure and there should be no overbooking. The probability of a departure at the other time epoch is calculated in the second equation. This probability consist of two summations. The first summation is the sum of the probability of arriving before the deadline but having to miss a departure. The second summation represents the probability of arriving after the deadline times the probability of missing the correct number of flights.

In the third equation, the probability of arriving at time x in node $j+1$ is calculated. This is a convolution of all possible departures at node j and the

time it takes to traverse arc $(j, j+1)$ such that the arrival at node $j+1$ is between $x-1$ and x. All the convolutions required in these three equations can be efficiently computed by standard Fast Fourier Transform algorithms. The probability mass function $g_n^A(x)$ can be used to compute the desired cumulative distribution function $F_P(T)$.

4.5.2 Optimal algorithm

The problem (4.1)-(4.2) is a variant of the *Resource Constrained Shortest Path Problem* (RCSPP), (see, e.g., (Pugliese and Guerriero, 2013)). For this problem, two types of exact algorithms have been developed: Dynamic Programming (DP) and Lagrangian Relaxation (Lozano and Medaglia, 2013). As the Lagrangian subproblem is still hard to solve for our problem, we have to decide to develop a DP algorithm.

In the DP algorithm, we iteratively go through all the nodes and keep track of all paths entering a node. In node i, we can discard a path P_1 if there exists a path P_2 with lower costs and that *stochastically dominates* P_1. By the latter we mean that the probability of arriving at node i before a given time is for path P_2 always greater than or equal to the probability for path P_1. To formalize this concept, let $G_{P,i}^A(\cdot)$ be the cumulative distribution function of the arrival time in node i in path P. So path P_2 stochastically dominates P_1 in node i if

$$G_{P_2,i}^A(x) \geq G_{P_1,i}^A(x) \qquad \forall x \geq 0. \tag{4.3}$$

The specifications of our problem make that it is not necessary to consider all values of x in Equation (4.3). It is only possible to leave node i at a specific number of departure times, namely $D(i) := \cup_{j:(i,j)\in\mathcal{A}} \cup_{k=0}^{\infty} \{d_{ij} + kf_{ij}\}$. Every arrival in node i between two consecutive departure times can be treated as the same. Hence, we can replace the condition of Equation (4.3) by

$$G_{P_2,i}^A(x) \geq G_{P_1,i}^A(x) \qquad \forall x \in D(i).$$

If path P_1 is more expensive than P_2 and is stochastically dominated by P_2, then we know that path P_1 will never be an optimal path for any acceptance threshold β. Hence, we can discard path P_1. However, the number of paths that need to be stored can be large, especially if the set $D(i)$ is large. Therefore, we also propose a heuristic in the next section.

Running example continued

The network given in Figure 4.1 has three possible routes from node 1 to node 4, namely a direct path, a route via node 2, and one via node 3. The direct path is the cheapest option with costs 15, the route via node 3 has costs 20, and the most expensive option is to ship via node 2 which has costs 25. In this

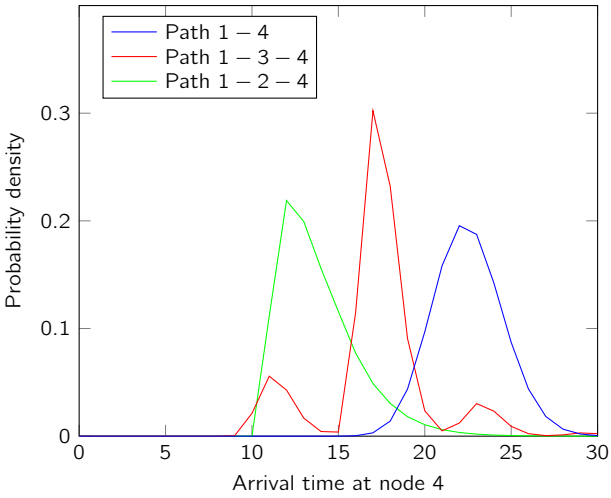


Figure 4.2 Probability density function of the arrival time at node 4 for the three different paths in the network from Figure 4.1.

example, the routes with lower costs also have a lower probability of arriving on time. The probability that the direct route arrives before 20 at node 4 is about 0.16, for the route via node 3 this probability is approximately 0.91, and finally, for the route via node 2 it is about 0.99.

In Figure 4.2, the distributions of the arrival time at node 4 for the different paths are plotted. The path 1-3-4 clearly has three different peaks that correspond to different planned departures. The first peak corresponds with the departure from node 3 at time 8, this peak is small because the probability of arriving before 8 at node 3 is not so large. The second peak resembles all departures from node 3 at time 14 and this is peak is the largest because the probability that the shipment arrives between time 8 and 14 at node 3 is large and the probability of overbooking is only 0.1. The third and smallest peak corresponds with the situation of an overbooking.

The fact that the distribution of path 1-2-4 has only one peak is caused by the fact that the frequency of the departures on arc (2,4) is 3 and thus the time between different departures is relatively short. Moreover, the travel time on that arc also has a larger variance. In general, the more nodes are in a path the more peaks are in the distribution of the arrival time at the final node.

If the three paths in this network would have been subpaths entering node 4 and there would have been an arc leaving node 4 with departure time 20, then it would not be possible to discard any of the three paths because the cheaper the path, the smaller the probability of arriving before 20 at node 4.

4.5.3 Risk measure heuristic

In this section, we develop a heuristic for the problem presented in Section 4.4. In this heuristic, all stochastic variables are replaced by a deterministic risk measure. As we would like to give multiple solutions for different levels of risk, we will define the level of risk one is willing to take by the *risk tolerance factor* α. The larger the value of α the more risk one is willing to accept. A risk measure for stochastic variable Φ and risk tolerance factor α is denoted by $\rho_\alpha(\Phi)$ and should be a decreasing function of α. We will use two different functions: the expectation-variance (EV) function and the certainty equivalent under exponentially disutility. In the remainder of this article, we will refer to the latter as the *certainty equivalent* (CE).

Expectation-variance method

The expectation-variance function for stochastic variable Φ on the domain $[\underline{\Phi}, \bar{\Phi}]$ and risk tolerance factor α is defined as (Markowitz, 1952):

$$E_\alpha(\Phi) := \max\left\{\underline{\Phi}, \min\{\mathbb{E}(\Phi) - \alpha \mathrm{Var}(\Phi), \bar{\Phi}\}\right\} \qquad \alpha \in (-\infty, \infty).$$

The idea behind the EV method is that for positive values of α a fraction of the variance is subtracted from the expectation, and a fraction of the variance is added to the expectation for positive values of α. As we replace Φ by $E_\alpha(\Phi)$, the value $E_\alpha(\Phi)$ is restricted to values that are between the lower and upper limit of Φ. The benefit of $E_\alpha(\Phi)$ is that it is easy to compute and has a relatively clear interpretation. Moreover, it can take any value in support of Φ and thus it can also be used for a risk-tolerant shipper.

Certainty equivalent method

For a random variable Φ and risk tolerance factor α the certainty equivalent is given as follows (Jaillet et al., 2016):

$$C_\alpha(\Phi) := \begin{cases} \alpha \ln \mathbb{E}\left(e^{\frac{\Phi}{\alpha}}\right) & \alpha > 0 \\ \lim_{\gamma \to 0} C_\gamma(\Phi) & \alpha = 0. \end{cases}$$

Using moment generating functions, $C_\alpha(\Phi)$ can be easily computed for a given distribution. For instance, $C_\alpha(\Phi) = \mu + \frac{\sigma^2}{2\alpha}$ if Φ is normally distributed with mean μ and standard deviation σ or $C_\alpha(\Phi) = \alpha \ln\left(\left(1 - \frac{\theta}{\alpha}\right)^{-k}\right)$ for $\alpha > \theta$ if Φ is gamma distributed with mean $k\theta$ and standard deviation $\sqrt{k}\theta$. The function $C_\alpha(\Phi)$ converges to the mean of Φ if α goes to infinity, and it converges to the upper limit of its domain if α goes to zero (Jaillet et al., 2016). A possible downside of this approach is that the value for $C_\alpha(\Phi)$ can thus never be below its mean, so it does not work very well for a risk-tolerant shipper. An advantage

of the certainty equivalent is that it grows exponentially for decreasing α, and thus, it converges fast to the upper limit of the stochastic variable. Therefore, it is a suitable risk measure for a risk-averse shipper.

Integer Linear Program Formulation

If each stochastic variable Φ is replaced by a risk measure $\rho_\alpha(\Phi)$, the problem becomes a deterministic optimization problem. For the moment, let us assume that the value of α is fixed, then the resulting problem can be formulated by the following ILP:

$$\min \quad \sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij} \tag{4.4}$$

subject to:

$$\sum_{j:(x_{1j})\in\mathcal{A}} x_{1j} = 1 \tag{4.5}$$

$$\sum_{i:(i,j)\in\mathcal{A}} x_{ij} = \sum_{k:(j,k)\in\mathcal{A}} x_{jk} \qquad j = 2,\dots n-1 \tag{4.6}$$

$$\sum_{i:(i,n)\in\mathcal{A}} x_{in} = 1 \tag{4.7}$$

$$a_i \le d_{ij} + (1-y_{ij})M \qquad \forall (i,j)\in\mathcal{A} \tag{4.8}$$

$$a_i \le d_{ij} + f_{ij}z_{ij} \qquad \forall (i,j)\in\mathcal{A} \tag{4.9}$$

$$\begin{aligned} &d_{ij} + f_{ij}z_{ij} + f_{ij}\left(\rho_\alpha\left(\Phi^1_{ij}\right)y_{ij} + \rho_\alpha\left(\Phi^2_{ij}\right)(1-y_{ij})\right) \\ &+ \rho_\alpha\left(\Phi^0_{ij}\right) \le a_j + (1-x_{ij})M \qquad \forall (i,j)\in\mathcal{A} \end{aligned} \tag{4.10}$$

$$a_1 = 0 \tag{4.11}$$

$$a_n \le T \tag{4.12}$$

$$a_i \ge 0 \qquad i = 1,\dots,n \tag{4.13}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j)\in\mathcal{A} \tag{4.14}$$

$$y_{ij} \in \{0,1\} \qquad \forall (i,j)\in\mathcal{A} \tag{4.15}$$

$$z_{ij} \in \mathbb{N}_0 \qquad \forall (i,j)\in\mathcal{A}. \tag{4.16}$$

In this ILP, there are four types of decision variables. For each arc (i,j) we have a binary variable x_{ij} indicating whether that arc is traversed or not. Second, for every node i the decision variable a_i corresponds with the arrival moment at node i. If a path does not visit node i, the value a_i can take any value. Third, the binary decision variable y_{ij} is equal to 1 if the shipment arrives at node i before the deadline d_{ij}. Finally, the decision variable z_{ij} describes the number of intervals after the deadline d_{ij} the arrival at node i is. Consequently, $d_{ij} + f_{ij}z_{ij}$ is the first possible departure from node i to node j.

The objective function in (4.4) minimizes the costs of the selected path. Constraint (4.5) enforces that the path leaves node 1 and constraint (4.7) that it arrives in node n. In constraint (4.6), it is ensured that if a path enters a node other than 1 or n, it also has to leave that node. If the value M is sufficiently large in constraint (4.8), then it enforces y_{ij} to be 1 if $a_i > d_i$. The constraint (4.9) makes sure that the departure time $d_{ij} + f_{ij}z_{ij}$ from node i is after a_i, the arrival at node i. In constraint (4.10), it is enforced that if arc (i, j) is traversed in the path, then the arrival at node j is at least a_j. Again here, we will need that M is a sufficiently large constant. In this constraint, the term $d_{ij} + f_{ij}z_{ij}$ represents the first possible departure time on arc (i, j). Furthermore, the number of missed departures is given by $\rho_\alpha\left(\Phi_{ij}^1\right) y_{ij} + \rho_\alpha\left(\Phi_{ij}^2\right)(1 - y_{ij})$, which equals $\rho_\alpha\left(\Phi_{ij}^1\right)$ is $y_{ij} = 1$ and $\rho_\alpha\left(\Phi_{ij}^2\right)$ otherwise. Finally, the term $\rho_\alpha\left(\Phi_{ij}^0\right)$ represents the travel time on arc (i, j), which is independent of the actual departure time of that arc. The constraints (4.11) and (4.12) ensure, respectively that the arrival at node 1 equals 0 and the arrival at node n is before the deadline T.

In practice, one might need multiple stochastic constraints. For instance, in a situation in which the freight has to be temperature regulated, the time the freight spends in a port without the right equipment to control the temperature of the shipment could also be subject to a probabilistic constraint. These types of constraints can easily be added to the ILP above. In such a situation, one should consider carefully if also multiple values for α are needed to differentiate between the risk tolerance for different constraints.

We solve the ILP (4.4)-(4.16) using a standard ILP solver. However, as this problem is a special case of the RCSPP, it is NP-hard. Hence, for larger-sized instances, the computation time might be too long. Nevertheless, many exact approaches have been developed for the RCSPP and they can be applied if needed. We refer to Pugliese and Guerriero (2013) for a survey on these exact algorithms.

Iterative procedure

In the ILP (4.4)-(4.16), it was assumed that α was fixed. As the value α has no clear interpretation, it is impossible for a practitioner to set a value of α beforehand. However, we know that the larger the value of α the more risk is taken and the cheaper the solution will be. Therefore, the Pareto-front can be constructed in the following way: initialize $\alpha = M$ sufficiently large and $\mathcal{P} = \emptyset$. Solve for that value of α the ILP (4.4)-(4.16). Assume the path resulting path consists of l visited nodes, and let us denote this path by $P = (p_1, p_2, \ldots, p_l)$. Add this path to the set $\mathcal{P}$. After that, find the minimum value of α for which the sum of all certainty equivalents of path P is less than T. In other words, find the minimum value of α for which the solution is feasible.

Algorithm 4.1: Procedure to find the minimum α for which a given path P is feasible for ILP (4.4)-(4.16).

Input: Path $P = (p_1, p_2, \ldots, d_l)$
Initialize $s = 0$, $\underline{\alpha} = 0$ if $\rho_\alpha(\Phi) = C_\alpha(\Phi)$ and $\underline{\alpha} = -M$ if $\rho_\alpha(\Phi) = E_\alpha(\Phi)$, and $\bar{\alpha} = M$ with M being a sufficiently large number.
while $\bar{\alpha} - \underline{\alpha} > 0.0001$ *or* $s < T - 1$ **do**
  $\alpha = \frac{\bar{\alpha}-\underline{\alpha}}{2}$
  $s = \rho_\alpha\left(\Phi^0_{p_1 p_2}\right)$
  for $i = 2, \ldots, l$ **do**
    if $s \leq d_{p_i p_{i+1}}$ **then**
      $s = d_{p_i p_{i+1}} + \rho_\alpha\left(\Phi^0_{p_i p_{i+1}}\right) + \rho_\alpha\left(\Phi^1_{p_i p_{i+1}}\right)$
    else
      $s = d_{p_i p_{i+1}} + \left\lceil \frac{s - d_{p_i p_{i+1}}}{f_{p_i p_{i+1}}} \right\rceil f_{p_i p_{i+1}} + \rho_\alpha\left(\Phi^0_{p_i p_{i+1}}\right) + \rho_\alpha\left(\Phi^2_{p_i p_{i+1}}\right)$
    end
  end
  if $s \leq T$ **then**
    $\bar{\alpha} = \alpha$
  else
    $\underline{\alpha} = \alpha$
  end
end
Output: α

A procedure to find this value for α is given in Algorithm 4.1. The idea of Algorithm 4.1 is simple: for a given value of α, the arrival time of path P at the final node n (s) is calculated. If this arrival time is lower than T, it means that the value of α for which the path arrives exactly at time T is larger. Hence, the lower bound for α, denoted by $\underline{\alpha}$ needs to be updated. If s is larger than T, then the upper bound for $\bar{\alpha}$ is updated. The function of the arrival time is discontinuous in α because if a path arrives just after d_{ij} at node i then the departure time at node i will be $d_{ij} + f_{ij}$. That is why in the while-statement in Algorithm 4.1, also the condition $s < T - 1$ is included. By this we ensure that in these situations we return the larger value of α. If the value of α is found in the way described in Algorithm 4.1, we subtract a small value ε from it to obtain a new value for α for which the solution P is not longer feasible. We again solve the ILP (4.4)-(4.16) and repeat the procedure until the ILP (4.4)-(4.16) is infeasible.

Running example continued

We now solve our running example given in Figure 4.1 with the risk measure heuristic. For this example, the solutions produced by the expectation-variance and the certainty equivalent approach are different. Recall that for α sufficiently large, the CE of a random variable is arbitrary close to its expectation. As the expectation of the travel time on the arc from node 1 to node 4 is larger than

the deadline, the direct route is infeasible for the CE method. The bottom route, via node 3, is feasible for the certainty equivalent. The travel time on the first leg is 9, so the shipment is too late for the planned departure time of 8. The first departure is at 14 and then the expected number of flights being missed is the $\frac{p_{ij}}{1-p_{ij}} = \frac{1}{9}$. The expected travel time from node 3 to node 4 is 3, so the arrival time at node 4 equals $17\frac{1}{9}$. This route is feasible as long as α is roughly larger than 3.10. If α is larger than 1.44 the route via node 2 is feasible, so the heuristic using the certainty equivalent approach will also give that route as an option.

The expectation-variance measure of a random variable takes a value smaller than its expectation as long as α is positive. Hence, if this measure is used, the direct route is found as long as $\alpha > \frac{22-20}{4} = \frac{1}{2}$. If α is just smaller than a $\frac{1}{2}$, the values taken by the EV method are close to the expectation, so the route via node 3 is also found in a similar way as for the certainty equivalent method. However, for the lowest value of α for which this route is feasible, the route via node 2 is infeasible. The EV method underestimates the risk of the path via node 3. So the risk tolerance factor for which it is still feasible is rather low.

Concluding, both the certainty equivalent method and the expectation variance method only find two of the three Pareto-optimal routes. The most risk-tolerant route is not found by the certainty equivalent method and the expectation-variance method does not return the most risk-averse route. Hence, if we combine the solutions from these two methods we do find the entire optimal Pareto-front.

4.6 Numerical results

In this section, we use numerical experiments to investigate the quality of the different solution methods. We first describe in Section 4.6.1 how we generate random instances. Afterward, in Section 4.6.2, these instances are solved using the solution methods described in Section 4.5 and the results are presented.

4.6.1 Instance generation

The different solution methods will be compared on randomly generated networks. These networks consist of n nodes with random arcs. To ensure that the graph is connected, we create an arc between every node i and $i+1$. For the other arcs, we assume that there is arc between node i and all nodes in $\left\{i+1, \ldots, \min\left\{n, i+\frac{n}{2}\right\}\right\}$ with probability $\frac{1}{2}$. This way, there could be a path from 1 to n with only one stop, but most routes will visit multiple hubs. This represents the dynamics of an intermodal network in which there is usually at least one long-haul trip from one hub to another.

The costs of an arc (i, j) are randomly generated as follows: $c_{ij} = (j - i) *$ Uniform$(1, 4)$. This ensures that the expected costs of an arc are larger if the destination of an arc is closer the final destination n. The mean duration of an arc (i, j) is randomly uniform between 3 and 10, and thus, it is independent of i and j. The standard deviation is randomly uniform between 0.2 and 3. We assume that the travel times are gamma distributed for two reasons. First, it is a right-skewed distribution, which reflects the situation that delays cause the mean of the distribution to be larger than its median. Second, for the gamma distribution the moment generating function and thus the certainty equivalent is well-defined.

We assume that the number of departures missed because of overbooking is geometrically distributed. The parameter of this distribution is 0.9 if the shipment arrives before its planned departure at a node an 0.75 if it arrives after its planned departure. In the first case, this reflects the situation with 10% overbooking probability. The value 0 is included in the domain of the geometric distribution to make sure that it is possible to miss no departures. We include the fact that a shipment is always shipped with the fifth departure, so it cannot miss more than four departures. Hence, the probability distributions for Φ^1_{ij} and Φ^2_{ij} are as follows for every arc (i, j):

$$\mathbb{P}\left(\Phi^1_{ij} = k\right) = \begin{cases} 0.1^k 0.9 & k = 0, 1, 2, 3 \\ 0.1^4 & k = 4 \end{cases}$$

$$\mathbb{P}\left(\Phi^2_{ij} = k\right) = \begin{cases} 0.25^k 0.75 & k = 0, 1, 2, 3 \\ 0.25^4 & k = 4. \end{cases}$$

For the scheduled departure on an arc (i, j), we calculate the shortest path with respect to the mean duration to node i and add a discrete random uniform number between 3 and 13 to it. The frequency of an arc is also a discrete number that is randomly generated from the uniform distribution between 4 and 20. We assume that every arc leaving node 1 has planned departure time 0 and is never overbooked, so it is always possible to leave this node at that time. For the final deadline, the shortest path to the destination with respect to the mean duration of the arcs is calculated. We multiply this shortest path with a *deadline factor* (T_f). This deadline factor reflects the behavior of a shipper that if there does not exist a route that gives a desired on-time arrival probability that then the deadline is set later.

4.6.2 Comparison of solution methods

In this section, we compare the optimal solution with the heuristic method using 100 instances of 50 nodes that are randomly generated as described in

T_f	Method	β 0.5	0.25	0.1	0.05	0.025	0.01
2	Opt	87	45	1	0	0	0
	EV	49	26	1	0	0	0
	CE	9	30	1	0	0	0
	EV+CE	67	35	1	0	0	0
3	Opt	100	100	87	71	57	21
	EV	85	76	49	31	20	6
	CE	96	94	73	57	46	19
	EV+CE	97	96	77	60	48	19
4	Opt	100	100	100	100	100	85
	EV	92	87	79	68	60	33
	CE	95	93	90	86	84	66
	EV+CE	96	95	92	86	86	69
5	Opt	100	100	100	100	100	100
	EV	93	90	84	78	68	51
	CE	96	93	91	91	89	86
	EV+CE	97	95	93	93	91	87

Table 4.2 Number of instances for which for different solution methods and acceptance thresholds feasible paths are found.

Section 4.6.1. We have chosen for this network size because its a realistic size and the optimal algorithm can still solve it in a few minutes. Nevertheless, for larger instances the running time of the optimal algorithm becomes problematic. For example, for a network consisting of 250 nodes, the average running is about an hour. The heuristic can still solve the problem for this size of networks in a few seconds.

To compare the quality of the heuristic paths with the optimal, we compute the actual probability of arriving late at the destination for every heuristic path. If it turns out that a path that was conceived as less risky by the heuristic is actually riskier and more expensive than another heuristic path, then it is removed from the set of heuristic paths. As the running time of the heuristic is relatively short, it is also possible to compute the solution of both the certainty equivalent and expected variance heuristic and take the best solution of the two. So in this section we will compare the EV heuristic, the CE heuristic, and the combination of these two (EV+CE) with the optimal solution. We assess the quality of the heuristic at two levels. First, we check if for a certain value of β a feasible solution is found. A good heuristic should have a high probability of finding a path if there exists a solution. The second level is the cost of a route produced by the heuristic. If the solution heuristic returns a path, then its costs should ideally be close to the optimal costs.

In Table 4.2, for different values of β and T_f it is shown for how many instances a solution is found by the different solution methods. By increasing the value of β or T_f, we see that the number of feasible solutions increases.

		β					
T_f	Method	0.5	0.25	0.1	0.05	0.025	0.01
2	EV	4.4%	3.4%	0.0%	0.0%	0.0%	0.0%
	CE	15.6%	5.3%	0.0%	0.0%	0.0%	0.0%
	EV+CE	5.4%	2.5%	0.0%	0.0%	0.0%	0.0%
3	EV	2.1%	2.7%	5.6%	3.7%	1.4%	0.0%
	CE	10.0%	5.0%	2.7%	2.7%	2.2%	0.0%
	EV+CE	3.1%	2.1%	1.5%	2.2%	2.1%	0.0%
4	EV	1.0%	5.2%	9.6%	12.1%	11.7%	5.9%
	CE	4.2%	2.4%	1.6%	3.7%	3.5%	3.7%
	EV+CE	1.5%	0.7%	1.0%	1.8%	2.5%	3.3%
5	EV	0.8%	2.2%	4.3%	8.0%	9.4%	6.6%
	CE	2.5%	2.5%	2.4%	1.9%	0.9%	1.1%
	EV+CE	0.5%	2.0%	1.5%	1.2%	0.8%	0.7%

Table 4.3 Average percentage difference in costs for a method with the optimal solution if method has found a solution for that instance.

Moreover, it can be concluded that the number of instances for which a feasible path is found by the EV and CE heuristic on their own is for certain combinations of β and T_f much lower than the instances for which the optimal solution gives a solution. However, by combining the two heuristics the number of feasible paths is much higher. Hence, we can conclude that the two different risk measures produce sufficiently different solutions. It should also be noted that the CE heuristic returns for more instances a feasible path, which confirms the idea that it is more risk-averse than the EV heuristic.

In Table 4.3, the costs of the solutions returned by the three heuristics is compared with the optimal costs. To make a fair comparison, we condition the costs for the optimal solution for all three methods only on the instances for which that heuristic finds a feasible solution. Hence, the number of instances for which we compute the optimality gap is the number given in Table 4.2. This could lead to outcomes that at first sight might be unexpected. For instance, for $T_f = 2$ and $\beta = 0.5$ the optimality gap for the EV heuristic is smaller than the optimality gap for the combination of the EV and CE heuristic. The solution returned by the combination of the EV and CE heuristic is at least as good as the solution from the EV heuristic. However, the combination heuristic finds solutions to more instances and these instances are likely to be harder. Hence, it is correct that the optimality gap for the combination heuristic is larger than for the EV heuristic. Nevertheless, in general the solutions that are obtained by combining are much better than those of the single heuristics and have an optimality gap of about 2%.

As we concluded before, the solutions from the CE heuristic are more conservative than those of the EV heuristic and this effect can also be seen in Table 4.3. For $\beta = 0.5$, the solutions produced by the CE heuristic have a

	Opt		EV		CE		EV+CE	
T_f	Mean	Std	Mean	Std	Mean	Std	Mean	Std
2	2.0	1.2	0.9	0.9	0.2	0.4	0.9	0.9
3	4.5	2.1	2.4	1.5	2.6	1.4	2.9	1.5
4	6.2	2.4	3.1	1.4	2.6	1.2	4.1	1.7
5	6.1	2.5	3.0	1.3	2.9	1.3	3.9	1.6

Table 4.4 Total number of paths found by the different methods

much larger gap with the optimal solutions than the solutions from the EV heuristic. This is caused by the fact that the CE heuristic often only produces solutions that have a much lower acceptance threshold than 0.5. For instance, if there is a cheap route for which the probability of arriving too late is 0.4 it is more likely to be found by the EV heuristic than by the CE heuristic. Nevertheless, if β decreases, the relative quality of the solutions by the CE method improves compared with the EV heuristic.

Finally, we look at the number of solutions produced by the different methods. The objectives of finding a path that arrives on time with sufficient probability and that has minimal costs are not conflicting if only a single solution is returned. We again solved the hundred instances but now for every $\beta \in \{0.01, 0.02, \ldots, 0.98, 0.99\}$. In Table 4.4, we give the average number of paths found for an instance and its standard deviation. We see that the average number of optimal paths for an instance is about 4 to 6. The number of paths returned by the two heuristics is lower with a value between 2 and 3. However, again by combining these two heuristics, the number of paths that is found is increased to an average value between 3 and 4. For T_f equals two, the number of average paths that is found is lower because there are fewer feasible paths. Furthermore, as we noted before the CE heuristic is more conservative than the EV heuristic and thus, the number of paths returned by the CE heuristic is also lower than for the EV heuristic.

4.7 Conclusion

We studied an intermodal routing problem inspired by a company that offers shippers a tool to find the best route for their shipment. In deciding what the best route is, a trade-off must be made between the shipment costs and the probability of arriving before a deadline at the destination. A distinct feature of our model is that it includes two types of stochasticity. First of all, the travel time between two nodes is stochastic. Moreover, we also added the possibility of overbooking as this happens regularly in practice and has a significant influence on the arrival time of a shipment. All legs in our transportation network have a planned departure time. If the shipment arrives after the planned departure time at the origin of this leg, then the probability of overbooking increases.

We have shown how to calculate, for a given path, the probability of arriving on time at the destination. As it could be hard to find dominating paths in our model, an optimal algorithm might need too much computation time for practical use. Therefore, we have proposed a heuristic in which the stochastic variables are replaced by two different deterministic risk-measures. The resulting problem is an RCSPP that we solve using an ILP formulation. This heuristic produces solutions that are close to the optimal solution. Moreover, the heuristic can solve networks with 250 nodes in a few seconds, and the optimal algorithm needs an hour for these networks.

Nevertheless, for even larger networks, also the running time of the heuristic might become too long to use in practice because multiple ILPs need to be solved. So an interesting direction for further research would be to find a heuristic that can solve larger instances. One way to do this could be to use exact algorithms tailored for the resource constraint shortest path problem, instead of using the standard ILP implementation. Another way to improve the running time of our heuristic could be to use a heuristic to solve the RCSPP.

In our current model, the planned departure time for a leg is assumed to be given. An extension of this model would be to decide on the first possible departure. This departure should then be in a set of possible departure times. This model would better reflect reality, but it also adds extra complexity to the problem. If one decides to plan the first departure later, then the probability of arriving before that time increases, and thus the probability of overbooking decreases. However, there is no option to leave earlier than the first departure, so the probability of arriving on time at the destination could also decrease.

In the context of container hinterland transportation, it might be more natural to investigate multiple containers simultaneously. Hence, a direction for further research is to extend the problem to the transshipment of multiple containers. One could think of a model that is in line with those of Chapters 2 and 3. In that case, a complicating factor is that an arc's capacity should also be taken into account. Moreover, the risk acceptance threshold and deadline could differ per container. Hence, it is no longer possible to construct a two-dimensional Pareto-front, unless one finds a way to represent the risk of a transportation plan in a single value.

In hinterland transportation networks, the number of possible transportation options is not that large. Therefore, in this context, a heuristic might not be needed. However, an interesting feature would be to include the fact that one can decide which terminals to visit with a barge. In Chapter 2, a barge visit was penalized because it increases the probability of a delay. If the model of this chapter is combined with that of Chapter 2, it might be possible to replace the artificial penalty by a risk acceptance threshold.

5
Pre-processing moves in container yards

5.1 Introduction

In the previous chapters, we have focused on the best way to ship containers in a multimodal network. In these networks, container terminals play an essential role because there the containers are transshipped from one node to another. As a result, it is possible to choose the most efficient mode of transportation for every part of a container's trip. Nevertheless, the handling of a container at a terminal imposes extra costs and takes time. Therefore, the operations at a terminal are essential to consider when optimizing the costs and reliability of multimodal transportation.

After a container arrives at a terminal, it is temporarily stored in a container yard. In order to save space, the containers are stacked on top of each other. However, the handling equipment at a terminal can only access the top container, so ideally, each container is at the top of its stack when it needs to leave the terminal. Nevertheless, when a container arrives at the container terminal, its departure moment is often unknown. Consequently, it frequently occurs that when a container needs to leave the terminal, other containers are stacked on top of that target container. These blocking containers will then need to be relocated to other stacks. These moves are called *relocation moves* and need to be prevented as much as possible since they impose extra costs and delays,

At an inland terminal, all outbound import containers need to be delivered to their final destinations by truck. A terminal is interested in reducing the turnaround time of a truck because then a truck can serve as many trucks as possible on a day. A reduction of the turnaround time is also beneficial for a truck driver because he or she can make more trips on a day. One way to reduce the truck's turnaround time is to perform fewer relocation moves when

This chapter is partly based on B.G. Zweers, S. Bhulai, and R.D. van der Mei. Optimizing pre-processing and relocation moves in the stochastic container relocation problem. *European Journal of Operational Research*, 283:954–971, 2020a and B.G. Zweers, S. Bhulai, and R.D. van der Mei. Pre-processing a container yard under limited available time. *Computers & Operations Research*, 123-105045, 2020c.

a truck is waiting at the terminal. Although the container's departure time is usually unknown when a container arrives at a terminal, a few hours before containers are retrieved, it is known in which period that will be. At an inland terminal, the crane to handle containers is more often idle than at a deep-sea terminal because fewer containers are transshipped via an inland terminal. At the time the crane is idle, it can perform *pre-processing moves*. The idea of pre-processing moves is to move containers to reduce the number of future relocation moves.

We are first to introduce the concept of these pre-processing moves in container yards. Therefore, this chapter is an introduction to the pre-processing moves. In Section 5.2, we review the relevant literature. The stacking problem we are facing is described in Section 5.3. Finally, in Section 5.4, two mathematical models are presented to solve the problem described in Section 5.3. In Chapters 6 and 7, solution methods to solve these problems will be presented.

5.2 Literature review

There are two main problems in the literature concerning the stacking of outbound containers: (i) the *Container Relocation Problem* (CRP), sometimes also called blocks relocation problem, and (ii) the *Container Pre-Marshalling Problem* (CPMP). Since our problem can be seen as a combination of the CRP and the CPMP, we will discuss the relevant literature for the two problems below.

In the CRP, each container has a unique label indicating the pick-up order of the containers. Containers that are on top of a container that is retrieved need to be relocated to other stacks. The objective of the CRP is to use as few of these relocation moves as possible. The CRP was introduced by Kim and Hong (2006) and is proven to be NP-hard (Caserta et al., 2012). As a result of this hardness, the literature concerning the container relocation problem can be divided into two different categories. The first branch of the literature deals with finding optimal solutions for the CRP, and the second stream of research focuses on heuristics for the CRP. One way to solve the CRP to optimality is to use integer programming. Caserta et al. (2012) were the first to introduce such a formulation. However, that formulation needed hours to solve instances of twenty to thirty possible locations of containers. In Zehender et al. (2015), the formulation of Caserta et al. (2012) is improved, but it is still not able to solve realistic instances.

Heuristics have been proposed for the CRP to solve larger instances in a reasonable time. In Caserta et al. (2011), a heuristic for the CRP that is based on dynamic programming is presented. As the number of states examined is restricted, this heuristic runs fast, but a rule-based heuristic described in Caserta

et al. (2012) gives better solutions. Another heuristic in which a restricted number of states is considered is presented in Wu and Ting (2012). In this article, the beam search procedure is applied. Also, a few meta-heuristics have been developed for the CRP. In Jovanovic et al. (2019b), the CRP is solved using ant colony optimization. A genetic algorithm is used in Hussein and Petering (2012) to solve a CRP variant in which the energy consumption is minimized. The energy consumption depends on the weight of a container, and thus, heavy containers should not move too far. They use the genetic algorithm to find the best parameter settings of a constructive heuristic.

The *Stochastic Container Relocation Problem* (SCRP), which is a generalization of the CRP, was introduced by Zhao and Goodchild (2010). In the SCRP, the exact order in which containers are retrieved is not known anymore. It is only given in which time interval a container is retrieved, but multiple containers could have the same interval. The retrieval order of the containers in the same time interval is a uniform random permutation. Using a simple heuristic, Zhao and Goodchild (2010) conclude that the value of information is essential for this problem. In Ku and Arthanhari (2016), a more advanced heuristic called the Expected Reshuffling Index is introduced. The idea behind this heuristic is to calculate a score for every stack based on the expected number of reshuffles needed for that stack. A container is relocated to the stack with the smallest number of reshuffles.

Another heuristic for the SCRP, the so-called Expected Minmax (EM) heuristic, is proposed by Galle et al. (2018b). This heuristic first tries to place a container in a stack where it does not need to be relocated. If such a stack does not exist, a container is placed in a stack where its relocation move will be the latest. In Galle et al. (2018b), an optimal formulation for the SCRP is also given, which can solve small instances within a reasonable time. The solution of the EM heuristic is close to the optimal solution and outperforms the ERI heuristic of Ku and Arthanhari (2016).

In the *Unrestricted Blocks Relocation Problem* (UBRP), another variant of the CRP, each container is allowed to be relocated and not only a container that is blocking another container. The first mathematical formulation for the UBRP is given in Caserta et al. (2012). In Petering and Hussein (2013), an Integer Linear Program (ILP) formulation that uses fewer decision variables than the model in Caserta et al. (2012) is proposed. Moreover, the running time of the model of Petering and Hussein (2013) is faster. At the moment, the best exact algorithm for the UBRP can be found in Tanaka and Mizuno (2018), in which an exact branch-and-bound algorithm is presented.

Besides the ILP formulation, also a look-ahead heuristic is proposed in Petering and Hussein (2013). In Jin et al. (2015), a heuristic that constructs a

partial tree with possible layouts is constructed. The layout for which a greedy heuristic gives the lowest number of relocation moves is selected as the best solution. In Tricoire et al. (2018), a metaheuristic for the UBRP is presented in which a beam search is combined with constructive heuristics. Finally, Feillet et al. (2019) present a local search heuristic for the UBRP that is incorporated in a dynamic programming formulation.

In Ji et al. (2015), a variant of the CRP is introduced that also incorporates loading plans of ships. In this problem, a ship's stowage plan is given, and the loading sequence has to be decided to minimize the number of relocations in the container yard is minimized. Ji et al. (2015) use a genetic algorithm to find good solutions for this problem. In Jovanovic et al. (2019a), a GRASP heuristic is proposed for the same problem, and this heuristic gives significantly better solutions than the method of Ji et al. (2015).

Contrary to the CRP and its variants, in the CPMP, containers are moved before any container is retrieved. These moves are called *pre-marshalling moves*. The goal of the CPMP is to use as few pre-marshalling moves as possible to obtain a stacking of the containers in which no relocation moves are needed. One way to solve the CPMP is to use ILP models. In the first paper about the CPMP, Lee and Hsu (2007) model the CPMP as a multi-commodity flow network that they solve using an ILP formulation. In De Melo da Silva et al. (2018), an ILP formulation is presented that can solve both the CPMP as the CRP. In Parreño-Torres et al. (2019), eight different ILP formulations for the CPMP are given, which are currently the best mathematical models for the CPMP.

Tree-based methods are another way to solve the CPMP to optimality. The first tree-based method was an A^* algorithm introduced by Expósito-Izquierdo et al. (2012). An A^* algorithm is a variant of a branch-and-bound algorithm in which the expected costs of the nodes below the current node are estimated. In Tierney et al. (2017), an improved A^* algorithm is given in which they make use of the lower bounds for the CPMP of Bortfeld and Forster (2012). The current state-of-the-art algorithm for solving the CPMP to optimality is a branch-and-bound algorithm that is first presented by Tanaka and Tierney (2018) and later improved by Tanaka et al. (2019). This method can solve almost all real-sized instances within an hour.

The CPMP has never been proven to be NP-hard, but since none of the optimal algorithms produces a fast solution, also heuristics have been developed. A constructive heuristic is the Lowest Priority First Heuristic (LPFH) of Expósito-Izquierdo et al. (2012). The idea behind the LPFH is to place the containers with the largest time frames in the correct position first. This heuristic consists of different phases, and in Jovanovic et al. (2017), multiple

heuristics are applied to every single of these phases. Multiple different solutions are obtained using this approach, and the best is chosen, which results in better solutions than the heuristic of Expósito-Izquierdo et al. (2012). In Hottung and Tierney (2016), a genetic algorithm is used to find the best parameters of a constructive heuristic. In Hottung et al. (2020), a tree search heuristic for the CPMP is developed in which the branching decisions are made based on a model learned by a deep neural network. The results of this heuristic are better than the heuristic of Hottung and Tierney (2016) but at the expense of longer computation times.

To the best of our knowledge, there is only limited work dealing with uncertainty in the CPMP, namely Rendl and Prandtstetter (2013) and Tierney and Voß (2016). These works study the Robust Container Pre-Marshalling Problem in which for each container, an interval in which the containers could be retrieved is given. The goal is to place containers such that the container's latest departure time is always earlier than the earliest possible departure time of a container underneath it. The difference between the intervals in this problem and the SCRP is that here the interval width is container-dependent. In Rendl and Prandtstetter (2013), this problem is solved using constraint programming, and in Tierney and Voß (2016), an IDA* heuristic is introduced that outperforms the earlier work of Rendl and Prandtstetter (2013) both in computation time and solution quality.

In the papers described above, the objective has been to minimize the number of pre-marshalling or relocation moves. Nevertheless, one can also decide to minimize the time needed to perform these moves. In Voß and Schwarze (2019), it has been shown that if this objective is chosen, then the number of relocation moves is often also minimal. In case the time needed is in the objective function, it is also natural to allow movement of containers to multiple bays. The first paper that includes a time dimension in the objective function is Lee and Lee (2010). In this paper, the objective is a weighted sum of the relocation moves and the time needed to perform these relocation moves. Lee and Lee (2010) propose a three-phase heuristic to solve this problem, but even for small instances, the running time is too large to be used in practice.

A much faster heuristic that also has a higher solution quality is proposed in Lin et al. (2015). This heuristic calculates for each stack a weighted sum of the minimum time frame of that stack and the time needed to travel to that stack. The stack for which this index is the lowest is chosen as a destination stack for the container. In Da Silva Firmino et al. (2019), an optimal A^* algorithm and a GRASP heuristic are proposed to minimize the crane's working time. A GRASP heuristic is a meta-heuristic in which a greedy construction heuristic is combined with a local search heuristic. In Galle et al. (2018a), a model is considered in which only the retrieval order of the first set of containers is known. The

objective is to minimize a weighted sum of the time needed to retrieve these containers, and the remaining future number of relocation moves in the bay. In Casey and Kozan (2012), a model is studied in which the objective is to minimize the total time needed to perform all movements. They consider both incoming and outgoing containers and propose both constructive and meta-heuristics to solve the problem. However, they study a terminal that uses a straddle carrier to handle containers, which is a different type of equipment than used in the other papers mentioned.

5.3 Problem description

In order to formulate our problem, it is crucial to describe the operations at a container terminal first in Section 5.3.1. In Section 5.3.2, we list all assumptions that are made in our problem, and finally, in Section 5.3.3, the pre-processing phase is introduced.

5.3.1 Terminal operations

In terminals, containers are stored in a rectangular yard, as illustrated in Figure 5.1(a). One row of containers in such a yard is called a *bay*. The commonly used equipment to store and retrieve a container from a yard is a Rail Mounted Gantry Crane (RMGC), which is also given in Figure 5.1(a). Other vehicles, such as terminal tractors or trucks, are used to move the container to another position. This position could, for instance, be the ship or the customer. In both figures of Figure 5.1, such a vehicle is depicted.

In Figure 5.1(b), one sees one bay and an RMGC with a trolley attached to it. The trolley can be lowered to pick up a container from the yard, but only the top container of a stack can be picked up by the trolley. Afterward, the trolley is lifted again to move the container over other containers to the end of the bay to place it at the vehicle. As a result, the maximum number of containers in a stack is limited, because otherwise, the trolley cannot move a container from one side of the bay to the other. Moreover, the width of the RMGC also imposes a constraint on the maximum number of stacks in a bay. Consequently, the number of containers in a bay, called *slots*, is also limited. For example, the bay in Figure 5.1(b) can contain at most four stacks that have a maximum height of three and therefore has twelve slots.

The RMGC also moves the top container of a stack to another stack if it has to be relocated. In case a container is moved to a stack in another bay, the entire RMGC has to move. At the same time, only the trolley has to move if the container is moved to a stack in the same bay. The former is much more time-consuming, and on top of that, in some terminals, it is not even allowed to move the crane if it is carrying a container (Lee and Hsu, 2007). Therefore,

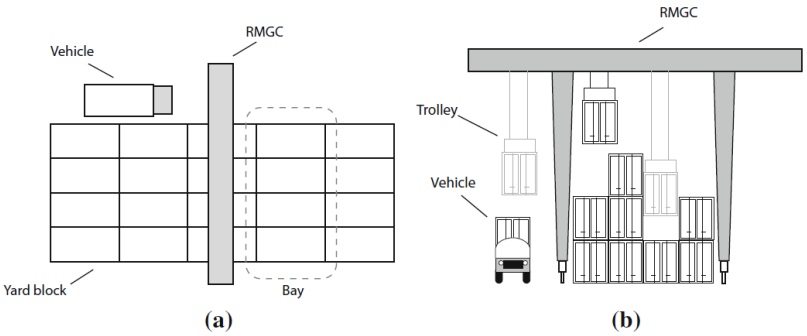


Figure 5.1 The layout of an RMGC and a container yard (Tierney et al., 2017).

it is not allowed to move a container from one bay to another bay, and thus, the problem is two-dimensional, similar to Figure 5.1(b).

5.3.2 Assumptions

In order to have a good balance between the current practice at container terminals and computational tractability, five assumptions are made.

Assumption 5.1. *All containers in a bay will leave the bay before any new containers arrive.*

Assumption 5.1 is made because it imposes a natural end of the period that needs to be considered. Moreover, most container terminals have specific areas in which only outbound or inbound containers are stored. Consequently, it often happens that no new containers arrive in a bay with outbound containers that will leave the terminal this day. A problem that arises when this assumption is not made is the *Dynamic Container Relocation Problem* (Akyüz and Lee, 2014).

Assumption 5.2. *A container may only be moved in the relocation phase if it blocks the container that needs to be retrieved.*

In Section 5.2, we have already seen that Assumption 5.2 is sometimes relaxed and that then the unrestricted version of the CRP is obtained. However, this assumption makes the relocation phase significantly easier, and it is also common practice in container terminals (Caserta et al., 2012). Under Assumption 5.2, the number of relocation moves performed when a single container is retrieved is the same as the number of containers that are blocking this target container. Whereas in the unrestricted version, there is no tight bound for the number of relocation moves. On top of that, with this assumption, checking

whether a container needs to be moved in the relocation phase is straightforward. If a container has only containers underneath it that are picked up later, it will never be moved in the relocation phase. We say that a container for which no relocation moves are needed is *well-placed* or *correctly-placed*. Contrary, a container that is not well-placed is called *badly-placed* or *poorly-placed*.

Assumption 5.3. *For each container, the time interval in which it is picked up is known, but the retrieval order inside an interval is a random uniform permutation.*

Assumption 5.3 distinguishes the CRP from the SCRP because, in the CRP, each container has its unique time interval in which it is picked up. This assumption reflects the situation in which terminals have a truck appointment system (Ku and Arthanhari, 2016). In such a system, the terminal has a fixed number of time intervals in which trucks can arrive to pick up a container. Multiple trucks can make an appointment to pick up a container in such a time interval. After that, the terminal knows which containers will leave the terminal in which interval, but it does not know anything about the order of the departures in such a time interval. As no information is available, it is most natural to assume that the retrieval order is a uniform permutation, since then each order is equally likely.

The moment the information of the retrieval order inside an interval becomes known to the terminal can vary and leads to two slightly different variants of the SCRP: the *online model* (Zhao and Goodchild, 2010; Ku and Arthanhari, 2016) and the *batch model* (Galle et al., 2018b). In the batch model, a container is only retrieved after all trucks have arrived at the terminal for that time interval. Consequently, if the first container in a time interval is retrieved, then the exact retrieval order for all containers in that time interval is known. Whereas in the online model, each container is retrieved immediately after a truck has arrived to pick it up. Therefore, no extra information about the containers that are retrieved later during that interval is known when it needs to be relocated.

The batch model is more suitable for large terminals in which the time intervals are short, and there is a significant amount of time between the arrival of a truck at the terminal and the moment it is served. In contrast, the online model better reflects smaller terminals in which trucks are served faster (Galle et al., 2018b). As the batch model implicitly assumes that the crane does not have any idle time, the online model is a variant in which pre-processing makes more sense. Motivated by this, we use the online model.

Assumption 5.3 is the only assumption that makes the problem stochastic, but the variability of the number of relocation moves is more than one might expect at first sight. For example, consider a stack that consists of two containers from the same time interval. If the top container is the first to be retrieved and

the bottom container the last, no relocation moves are needed. Whereas if the bottom container is the first to be retrieved, one relocation move is needed for the top container. Hence, with probability $\frac{1}{2}$ no relocation moves are needed and with probability $\frac{1}{2}$ one relocation move is needed for the containers in that interval.

On top of that, note that the fact that the retrieval order of containers in one specific interval is stochastic does not only influence the expected number of relocation moves needed to retrieve the containers from that interval. The relocation moves performed in one time interval influence the layout of the bay after that interval has ended. Because of Assumption5.2, it is only allowed to move containers that are on top of the target container. Therefore, the retrieval order of the containers inside an interval has a significant influence on how relocation moves are performed. Consequently, it might be that each retrieval order of an interval results in a different layout after the interval is finished. As a result, the first interval's retrieval order might influence the number of relocation moves needed for the last interval. Thus, the total number of relocation moves can vary across a wide range of values. Finally, Assumption 5.3 implies that computing the expected number of relocation moves for a given bay and relocation policy is computationally expensive. On the other hand, if a certain relocation policy for the deterministic CRP is used, then the number of relocation moves for a specific bay can be computed efficiently.

Assumption 5.4. *The time to move a container from one stack to another does not depend on the stack to which a container is moved.*

Assumption 5.4 is based on the fact that the time needed to pick up and release a container with a trolley is considerably larger than the time needed to move the trolley. As a result, the stack to which a container is relocated does not significantly influence the total relocation time of a container. In Section 5.2, we have seen a few papers in which Assumption 5.4 is not made, and the total objective is to minimize the weighted average of the number of relocation moves and the total working time of the crane. A consequence of Assumption 5.4 is that the order of the stacks is not relevant, and any permutation of the stacks can be treated as if it were the same instance. Another consequence of Assumption 5.4 is that only the number of moves is relevant in a solution.

Assumption 5.5. *It is physically possible to stack every container on every other container.*

Since containers have different sizes, Assumption 5.5 is not evident for general containers. A container of forty feet cannot be stacked on top of a single container of twenty feet. However, Assumption 5.5 is an assumption

that is not restricting from a practical perspective because most terminals locate containers with different sizes in different bays. Moreover, in case there would be certain containers on which we cannot stack a container, the pre-processing and relocation moves are easier: when we need to determine to which stack a container is moved, we could simply ignore the stacks in which we cannot stack the container.

5.3.3 Pre-processing phase

If the crane is idle, it can perform some pre-processing moves to position the containers so that fewer relocation moves are needed. At first sight, the pre-processing moves might look the same as the pre-marshalling moves in the CPMP. However, the main difference is that after all pre-marshalling moves are performed, no relocation moves are needed, and there might still be some relocation moves after the pre-processing phase has finished. Therefore, we can distinguish two phases in our problem: (i) the *pre-processing phase* and (ii) the *relocation phase*. The pre-processing phase ends when the first container is retrieved from the yard, and at that time, the relocation phase starts. The relocation phase is equivalent to the SCRP.

There are two main advantages of pre-processing moves over pre-marshalling moves. First of all, it could be that the crane's idle time is too short to perform all pre-marshalling moves. Second, positioning a container such that it does not need to be relocated might require many moves. In that case, the extra costs and time needed do not outweigh the reduction of a single relocation move. These two different advantages result in two different problems: (i) the *Stochastic Container Relocation Problem with Pre-Processing* (SCRPPP) and (ii) the *Stochastic Container Relocation Problem with Constrained Pre-Processing* (SCRPCPP).

In the SCRPPP, the pre-processing moves and relocation moves are jointly minimized, whereas, in the SCRPCPP, the time that is available for the pre-processing phase is limited. Hence, the SCRPPP is suitable if one wants to set costs on the moves, and if the crane's idle time is short, the SCRPCPP should be used. In Sections 5.4.1 and 5.4.2, we give a formal definition of, respectively, the SCRPPP and SCRPCPP.

Running example

Throughout Chapters 5, 6, and 7, an example will be used to illustrate the problems and their solution methods. This example is given in Figure 5.2 and the viewpoint in this figure is the same as in Figure 5.1(b). The bay in Figure 5.1 consists of five stacks that all have a maximum height of four containers. The numbers inside each container represent the interval in which the container is

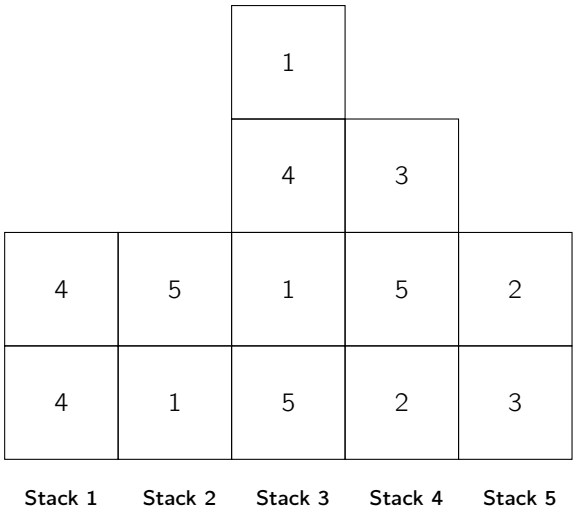


Figure 5.2 Example of a bay with five stacks that have a maximum height of four containers.

retrieved. Moreover, the stacks are numbered from left to right, so the leftmost stack is stack number 1, and the rightmost stack is stack number 5.

Although it is easy to check that a container is relocated at least once, computing the exact number of expected relocation moves for a container is difficult. As then, one also needs to be able to compute the bay at the moment that the container has to be relocated, and that, in turn, depends on the previous relocation moves. For example, the container with time frame 5 in stack 4 is relocated at least once, because the container underneath it is retrieved in time interval 2. However, it is hard to consider all potential bays at the moment that this container is relocated.

On the other hand, computing the exact expected number of relocation moves for the top container of the first stack is easy. The retrieval order of the containers in a time interval is a random uniform permutation. Thus, both the probability that the top container of stack 1 is retrieved before and after the bottom container is $\frac{1}{2}$. If the top has to be relocated, there are only containers with time frames 4 and 5 left in the bay. All other containers have already left the bay. Besides, the top container of stack 1, there are only two other containers with time frame 4. Hence, there is always an empty stack or a stack with only containers with time frame 5. Hence, if the top container of stack 1 is relocated, it will only be relocated once, and its expected number of relocation moves is $\frac{1}{2}$.

In the pre-processing phase, one could decide to move the container with

time frame 3 in stack 4 to the first stack. In the fourth stack, this container needs to be relocated at least once because that stack's bottom container is retrieved in time interval 2. However, in stack 1, the container with time frame 3 is positioned correctly. Hence, with that pre-processing move, the expected number of relocation moves is reduced by at least one.

5.4 Mathematical model

To describe the SCRPPP and SCRPCPP, we need to introduce some notation. All notation that is used in Chapters 5, 6, and 7 is summarized Table 5.1. Let us call the initial bay before the pre-processing phase B. We can denote the movement of the top container from stack o to stack d as the pair (o, d). If p is the number of pre-processing moves executed, the set of all pre-processing moves can be denoted by: $\mathcal{P} = \{p_1, p_2, \ldots, p_m\} = \{(o_1, d_1), (o_2, d_2), \ldots, (o_m, d_m)\}$. The bay that is obtained by the pre-processing moves $\mathcal{P}$ will be called $B(\mathcal{P})$.

Let n be the number of different retrieval orders in a bay and let $\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ be the set of those orders. We assume that the relocation phase's movements are made according to a certain relocation policy π that potentially takes the complete bay and all possible retrieval orders into consideration. Given a relocation policy π and a retrieval order σ_i, the number of relocation moves for a bay B is denoted by $R_\pi(B, \sigma_i)$. Hence, the expected number of relocation moves for a bay B is given by $\frac{1}{n}\sum_{i=1}^{n} R_\pi(B, \sigma_i)$.

Computing the exact expected number of relocation moves can be computationally intensive because the number n can be huge. Consider a bay with 28 containers that are partitioned into seven intervals. If every interval consists of four containers, the number of retrieval orders is $(4!)^7$, which is approximately 4.5 billion. Therefore, we will sometimes use a function to estimate the number of relocation moves. For a given bay B and policy π, we use $f(B, \pi)$ to refer to such an estimate.

The remainder of this section is organized as follows. In Section 5.4.1, the SCRPPP is defined as a mathematical optimization problem, and the SCRPCPP is defined in Section 5.4.2. Furthermore, we show in Section 5.4.3 that both the SCRPPP and the SCRPCPP are NP-hard. Finally, we derive in Section 5.4.4 bounds on the number of containers that can be in a bay to allow pre-processing moves.

5.4.1 SCRPPP

The goal of the SCRPPP is to find the pre-processing moves $\mathcal{P}$ and the policy π such that the weighted sum of the number of pre-processing moves and the expected number of relocation moves is minimized. The weight of pre-processing move is denoted by α, which is a value between 0 and 1. The weight for a re-

B	Specific layout of a bay
$\mathcal{S}$	Set of all stacks in a bay
$\mathcal{C}$	Set of containers in a bay
$\mathcal{P}$	Set of all pre-processing moves
$B(s_1, s_2)$	Layout of bay B after the top container of stack s_1 has moved to stack s_2
$B(\mathcal{P})$	Bay after the pre-processing moves in $\mathcal{P}$
H	Maximum height of a stack
C	Number of containers in a bay
b	Number of bays in a yard
Z	Largest time frame in a bay
$n(s)$	Number of containers in stack s
$l(s)$	Smallest time frame of stack s
$h(s)$	Largest time frame of stack s
$s(c)$	Stack of container c
$u(c)$	Smallest time frame of containers underneath c
$t(c)$	Time frame of container c
$q(c)$	Category of container c
$p(c)$	Position of a container c in stack $s(c)$ 1 is the lowest and H the highest position
$C(t, s)$	Container that is positioned in stack s at tier t
UB	Upper bound for the optimal solution
LB	Lower bound for the optimal solution
π	Relocation policy
$f(B, \pi)$	Estimation of expected number of relocation moves in bay B using policy π
p_j	The j^{th} pre-processing move
τ	Time needed to move a container within a bay from one stack to the other
t_{ij}	Time needed to move the crane from bay i to bay j
T	Time available for the pre-processing phase
ρ	Maximum number of pre-processing moves

Table 5.1 Notation used in Chapters 5, 6, and 7.

location moves is normalized to 1. The pre-processing moves are given a lower weight because they are performed at less busy periods. Formally, the goal of the SCRPPP is to find the following minimum:

$$\min_{\mathcal{P},\pi} \quad \alpha|\mathcal{P}| + \frac{1}{n}\sum_{i=1}^{n} R_\pi(B(\mathcal{P}), \sigma_i). \tag{5.1}$$

The part $\alpha|\mathcal{P}|$ in Equation (5.1) is equivalent to the costs for the pre-processing moves. By the pre-processing moves $\mathcal{P}$, the bay B is transformed into bay $B(\mathcal{P})$, so the expected number of relocation moves for the bay after pre-processing moves is calculated via $\frac{1}{n}\sum_{i=1}^{n} R_\pi(B(\mathcal{P}), \sigma_i)$.

At first sight, one might think that it is beneficial to perform many pre-processing moves such that no relocation moves are needed because pre-processing moves have a lower weight than relocation moves. However, re-

location moves have two advantages compared to pre-processing moves. The first advantage is that during the pre-processing phase, all original containers are still in the bay. In contrast, when a relocation move is executed, some containers have already left the bay. The second advantage of a relocation move is that when the container needs to be relocated, at least some part of the retrieval order inside an interval is known. On the contrary, when a pre-processing move is made, no information about the retrieval order inside an interval is known.

If α is close to 1, the two benefits of the relocation moves outperform the advantage of the pre-processing moves, which have a slightly smaller weight. Hence, no pre-processing moves will be performed, and the problem is similar to the SCRP. On the other hand, if α is close to 0, as many pre-processing moves as needed will be performed to prevent any relocation move, and the problem is equivalent to the CPMP. However, for more moderate values of α, it is hard to find the right balance between pre-processing moves and relocation moves.

Running example continued

Let us now consider a situation in which we perform four pre-processing moves to the bay in Figure 5.2, namely $\mathcal{P} = \{(4,1),(4,5),(4,1),(2,1)\}$. In Figure 5.3, the bays after all these four pre-processing moves are shown. The idea of the first three pre-processing moves is to empty the fourth stack such that in the fourth pre-processing move the container with time frame 5 from the second stack can be placed in the fourth stack. Note that this container with time interval 5 was not placed correctly in stack 2, but that it is well-placed in the fourth stack.

If we apply the optimal policy for the relocation moves to the bay in Figure 5.3(d), the expected number of relocation moves is 3. Hence, the objective function for the bay in Figure 5.2 to which we apply these four pre-processing moves and the optimal relocation policy is $4\alpha + 3$. If one would use the optimal policy for the relocation moves for the bay in Figure 5.2 without any pre-processing moves, the expected number of relocation moves for the bay in Figure 5.2 is $6\frac{1}{3}$. Hence, the moves P are beneficial as long as $\alpha \leq \frac{3\frac{1}{3}}{4} = \frac{5}{6}$.

5.4.2 SCRPCPP

In the SCRPCPP, it is assumed that there is a fixed time available for the pre-processing phase, which we denote by T. In Assumption 5.4, we have assumed that the time needed for a move is independent of the destination stack. Consequently, we can assume that there is a fixed amount of time needed to make a single pre-processing move, which we denote by τ. Using these two time units, there is a maximum number of pre-processing moves that

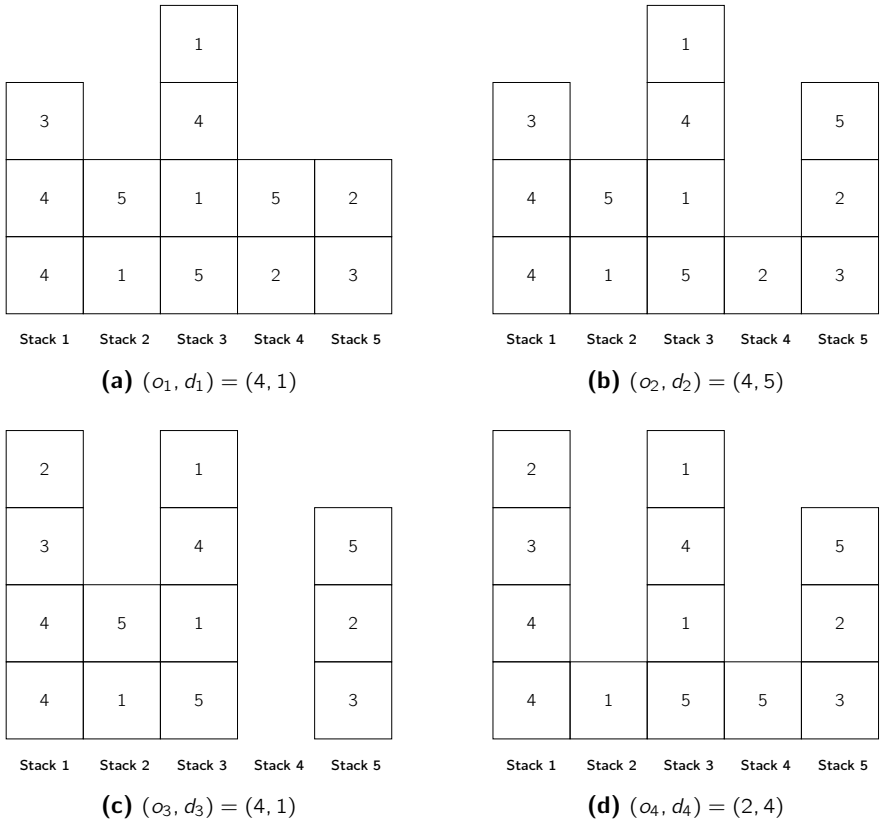


Figure 5.3 Four pre-processing moves for the bay of Figure 5.2.

can be performed, namely $\rho := \left\lfloor \frac{T}{\tau} \right\rfloor$. Note that if ρ is defined in this way, it is an integer. Using this notation, the SCRPCPP is given by the following mathematical optimization problem:

$$\min_{\mathcal{P},\pi} \quad \frac{1}{n} \sum_{i=1}^{n} R_\pi(B(\mathcal{P}), \sigma_i) \tag{5.2}$$

subject to

$$|\mathcal{P}| \leq \rho. \tag{5.3}$$

The objective function in (5.2) represents the expected number of relocation moves for bay B after the pre-processing moves in $\mathcal{P}$ if policy π is applied to the relocation phase. Furthermore, the constraint in (5.3) ensures that no more than ρ pre-processing moves are performed.

Running example continued

If only a single pre-processing move is allowed for the bay of Figure 5.2, the best pre-processing move is moving the top container of the fourth stack to the first stack. The top container of stack 4 is poorly-placed in that stack and will need to be relocated at least once. All containers in the first stack have a higher time frame than 3, so the container will not need to be relocated if placed in stack 1. The top container of stack 3 can also be correctly placed in stacks 1, 4, and 5. However, its expected number of relocation moves is only $\frac{1}{2}$ in the example bay of Figure 5.2, so less improvement is made by moving this container. Nevertheless, if two pre-processing moves are allowed, this container will be moved. If more than two pre-processing moves are allowed for the bay of Figure 5.2, it is more complicated to determine the best pre-processing moves.

5.4.3 Complexity

In Caserta et al. (2012), the CRP is proven to be NP-hard. In this proof, the decision problem of *Mutual Exclusion Scheduling* (MES) is reduced to an instance of the decision version of the CRP. Solving that instance is shown to be equivalent to a yes-instance of the MES by Caserta et al. (2012). That specific instance is also an instance for the SCRPPP and the SCRPCPP, and it is trivial to show that it can be solved in n pre-processing moves and no relocation moves if and only if it corresponds to a yes-instance of the MES. Hence, deciding if the objective function of the SCRPPP equals αn is equivalent to finding a yes-instances of the MES. Furthermore, if $\rho = n$ deciding if no relocation moves are needed for the SCRPCPP is also equivalent to a yes-instance of the MES. Therefore, both the SCRPPP and the SCRCPP are NP-hard.

5.4.4 Feasibility pre-processing moves

It could be that for a given bay, not all types of pre-processing moves are possible. Therefore, in this section, we develop bounds on the maximum number of containers in a bay such that pre-processing moves can be performed. It is straightforward to see that all slots in a bay are occupied, then it is impossible to move any container in the pre-processing phase. Moreover, if relocation moves are needed to retrieve the first container, then there are no empty slots for the containers that need to be relocated. Hence, it is likely that the SCRPPP and the SCRCPP have no feasible solution. In Lemma 5.1, we give a bound on the maximum number of containers that can be in a bay for the SCRPPP and SCRCPP to have a feasible solution.

Lemma 5.1. *If the number of containers C is bounded by $C \leq SH - (H - 1)$, then there exists a feasible solution for the SCRPPP and SCRPCPP.*

The proof of Lemma 5.1 follows directly from the condition in Caserta et al. (2012) that the number of containers in a bay does not exceed $SH - (H - 1)$ is sufficient for the feasibility of the CRP. The stochasticity of the SCRP does not influence this bound, and since not performing any pre-processing moves is always feasible, this bound also applies for the SCRPPP and SCRPCPP.

The bound given in Lemma 5.1 only considers the moves executed in the relocation phase. However, if there are $SH - (H - 1)$ containers in a bay, it might not be possible to move a container in the pre-processing phase such that it becomes well-placed. For example, consider a bay in which all stacks but one have reached their maximum height, and the other stack contains only a single container. In this bay, exactly $SH - (H - 1)$ containers are situated. If one of the completely filled stacks has a container with the largest time frame Z as its top container, an empty stack is needed before this container can be well-placed. The reason for that is that this container with time frame Z can only be well-placed at the bottom of a stack. Since there are only $H - 1$ free slots in the bay, it is impossible to get an empty stack and correctly place the container with time frame Z. However, if we have H available slots in a bay, it is possible to place any container to the bottom of a stack, as is shown in Lemma 5.2.

Lemma 5.2. *If there are H empty slots in a bay B with $S > 2$ stacks, it is possible to move every container to the bottom of each stack in the pre-processing phase.*

Proof. Let us denote the container that we would like to place at the bottom of a stack by c and the stack it is currently located by s. Furthermore, we name the stack to which we would like to move container c in the pre-processing phase s'. Without loss of generality, we can assume that container c is the top container of its stack. In the case that container c is not the top container, all containers on top of it could be placed in different stacks, and container c is the top container of its stack. For the remainder of the proof, we distinguish two cases: (i) stack $s \neq s'$, and (ii) $s = s'$.

Let us consider the first case and choose one stack $s'' \in \mathcal{S} \setminus (\{s\} \cup \{s'\})$. We make sure that its height becomes $H - 1$. If its current height is H, we move its top container to any stack in $\mathcal{S} \setminus \{s\}$. In case its current height is less than $H - 1$, we place containers from the stacks $\mathcal{S} \setminus \{s\}$ in stack s'' until it has reached the height $H - 1$. If there are not enough containers in the stacks $\mathcal{S} \setminus \{s\}$ to fill the stack s'' to the height $H - 1$, then all these stacks should be empty. Hence, container c can be placed at the bottom of stack s'. Therefore, we can assume that stack s'' has reached the height $H - 1$. After stack s'' has reached the height $H - 1$, we place container c in stack s''. The stack s''

has now reached its maximum height and in the stacks $\mathcal{S} \setminus \{s''\}$ are at least H available slots.

By definition, the stack s' contains $n(s')$ containers, which means that there are $H - n(s')$ free slots in that stack. Consequently, there are at least $H - (H - n(s')) = n(s')$ free slots in the stacks $\mathcal{S} \setminus (\{s'\} \cup \{s''\})$, thus all containers of stack s' can be placed in the stacks $\mathcal{S} \setminus \{\{s'\} \cup \{s''\})$. Afterward, stack s' is empty, and the container c that is on top of the stack s'' can be placed at the bottom of stack s'.

Secondly, we will consider the case that $s = s'$. Again a stack $s'' \in \mathcal{S} \setminus \{s\}$ is chosen to get a height of $H - 1$. If there are not enough containers in the stacks $\mathcal{S} \setminus (\{s\} \cup \{s''\})$ to fill stack s'' with $H-1$ containers, it means again that the stacks $\mathcal{S} \setminus (\{s\} \cup \{s''\})$ are empty. Therefore, we could place container c in stack s'' and all other containers from the stack s in the stacks $\mathcal{S} \setminus (\{s\} \cup \{s''\})$. Afterward, container c can be placed at the bottom of stack s. In case that stack s'' can be filled with $H-1$ containers, we place container c also in stack s''. In the stacks $\mathcal{S} \setminus (\{s\} \cup \{s'\})$ are now at least $H - n(s)$ free slots, thus all $n(s)$ containers from stack s can be placed in the stacks $\mathcal{S} \setminus (\{s\} \cup \{s'\})$. Finally, container c can be placed at the bottom of stack s. □

Since we have shown above that it might not be possible to move a container to the bottom of a stack if there are $H - 1$ free slots in a bay, we know that the bound given in Lemma 5.2 has to be tight. A consequence of Lemma 5.2 is that if the number of containers in a bay with more than two stacks is limited by $(S - 1)H$, then each container can become well-placed.

Corollary 5.1. *If a bay B with $S > 2$ stacks has at most $C \leq (S - 1)H$ containers, every container could be moved in the pre-processing phase such that it becomes well-placed.*

Proof. By Lemma 5.2, we know that it is always possible to place a container at the bottom of a stack if there are H free slots in a bay with more than two stacks. Since a container at the bottom of a stack is always well-placed, we know that each container can become well-placed if the number of free slots is at least H. If the number of containers in a bay is bounded from above by $(S - 1)H$, the number of free slots should be at least H. □

In Lemma 5.2, we have shown that there always exists a feasible solution to move a container from one stack s to the bottom of another stack s' if there are H free slots in the bay. However, under that condition, one might need to move containers from a third stack s''. In the heuristics we give in Chapters 6 and 7, we would like to move a container in the pre-processing phase from a stack s to a correct position in stack s' without moving containers from other stacks.

In Lemma 5.3 and Corollary 5.2, we prove that that is possible if there are $2(H-1)$ free slots.

Lemma 5.3. *If a bay B has $2(H-1)$ empty slots, each container in stack $s \in \mathcal{S}$ can be moved in the pre-processing phase to the bottom of stack $s' \in \mathcal{S}$ without moving containers from the stacks $\mathcal{S} \setminus (\{s\} \cup \{s'\})$.*

Proof. Let container c be the container that is moved in the pre-processing phase such that it is placed at the bottom of stack s'. As a result of Assumption 5.4, we can reshuffle the stacks in any order without adjusting the problem. Thus, if container c is already placed at the bottom of a stack, we can shuffle the stacks so that the container c is placed at the bottom of any other stack. Hence, without loss of generality, we assume that container c is not yet placed at the bottom of a stack.

We distinguish two cases: (i) the situation in which the origin and destination stacks s and s' are different, and (ii) the situation in which they are the same. In the first case, we need to have available slots for all containers in the destination stack s', which are at most H containers. Moreover, at most $H-2$ containers are placed on top of container c. Hence, at most $H-2$ slots should be available for containers from the stack s. Therefore, the total number of free slots needed to move container c from the origin stack to the destination stack is $H + H - 2 = 2(H-1)$.

In the second scenario, in which the origin and the destination stack are the same, the reasoning is slightly different. It is essential to realize that we need to store container c temporarily in another stack and that no other containers from the origin stack might be placed on top of container c because otherwise, it cannot be placed again in the origin stack. Hence, we need at least two stacks with available slots. Since container c should block the smallest number of empty slots as possible, it is placed on the stack that already consists of the most containers. Let us denote the stack in which container c is temporarily stored by s'' and let $p(c)$ be the position of container c in stack s''. After container c is placed in stack s'' we cannot use stack s'' anymore for the other containers. The number of available slots that cannot be used in stack s'' equals $H - p(c)$. Hence, according to the condition of this lemma, there should be at least $2(H-1) - (H - p(c)) = H + p(c) - 2$ slots available for the other containers in the origin stack. Since $p(c)$ is by definition at least 1, there are at least $H-1$ slots available for other containers, and they can thus be placed in other stacks than stack s''. Afterward, container c can be placed at the bottom of the origin stack. □

Note that in Lemma 5.3, we do not impose the condition that there are more than two stacks because this lemma also holds if the number of stacks is

one or two. If there is only one stack, all $2(H-1)$ free slots should be in that single stack. The inequality $2(H-1) \leq H$ only holds for the trivial bay in which $H = 1$ and there is only one container, or $H = 2$, and there are no containers. In case there are two stacks, there can be at most two containers in the bay if the number of free slots is at least $2(H-1)$. With two containers, both containers can be placed at the bottom of a stack, so the lemma also holds.

The bound given in Lemma 5.3 is tight. Consider a bay with three stacks, in which one has a height of H, the second has two containers, and the third stack contains only a single container. The number of free slots in this bay equals $H-2+H-1 = 2(H-1)+1$. To place the top container of the second stack in the third stack, the third stack needs to be empty. However, there is no place for the container of the third stack in the first stack. Furthermore, it is also impossible to move the top container of the second stack at the bottom of the second stack. We could move the top container of the second stack to the third stack, but then the second stacks' bottom container has no stack in which it can be placed.

The minimum number of available slots given by Lemma 5.3 can be used to get a bound on the maximum number of containers that can be in a bay such that every container can become well-placed in a stack without moving containers from a third stack. In Corollary 5.2, we derive a bound on this number of containers. The proof of this corollary is similar to the proof of Corollary 5.1.

Corollary 5.2. *If the number of containers C is bounded by $C \leq SH-2(H-1)$, then every container could be moved from stack $s \in \mathcal{S}$ in the pre-processing phase to stack $s' \in \mathcal{S}$ such that it is well-placed without moving containers from stacks $\mathcal{S} \setminus (\{s\} \cup \{s'\})$.*

6
Optimizing pre-processing and relocation moves

In this chapter, we describe solution methods to solve the SCRPPP, which is defined in Section 5.4.1. We give in Section 6.1 both a heuristic method and an optimal branch-and-bound algorithm. The solution quality and running time of these methods are investigated in Section 6.2. We end this chapter with a conclusion in Section 6.3.

6.1 Solution methods

In this section, we present both a heuristic method and an optimal branch-and-bound algorithm for the SCRPPP. As shown in Section 5.4.3, the SCRPPP is NP-hard. Thus we expect that for larger instances, the branch-and-bound algorithm cannot produce an optimal solution in a reasonable time. Therefore, the local search heuristic is needed to produce solutions for large-sized instances. Moreover, the branch-and-bound algorithm needs a good upper bound for the optimal solution in order to run efficiently, and the solution of the heuristic is a good upper bound for the optimal solution.

In Section 6.1.1, we will describe a local search heuristic to solve the SCRPPP. As a subroutine, this heuristic needs a fast method to get an estimate for the expected number of relocation moves for a given bay. In Section 6.1.2, we will describe the method that is used to get such an estimation. For the optimal branch-and-bound algorithm that is described in Section 6.1.4, we need a lower bound, which is derived in Section 6.1.3.

6.1.1 Local search heuristic

In this section, we will describe a local search method to solve the SCRPPP. Our main focus is on the pre-processing phase because in Galle et al. (2018b) it is shown that the Expected Minmax (EM) heuristic produces fast and good solutions for the SCRP. Therefore, we will use that heuristic for the relocation phase.

This chapter is based on B.G. Zweers, S. Bhulai, and R.D. van der Mei. Optimizing pre-processing and relocation moves in the stochastic container relocation problem. *European Journal of Operational Research*, 283:954–971, 2020a.

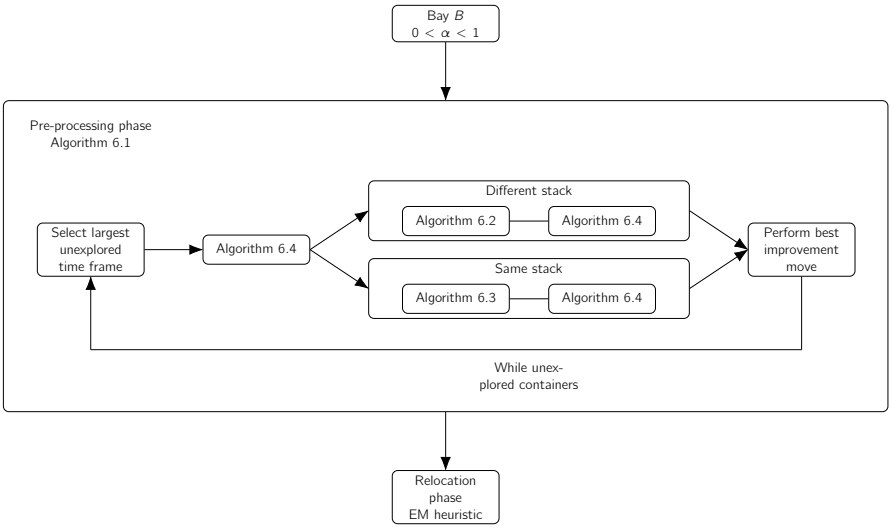


Figure 6.1 Structure of the local search heuristic for the SCRPPP.

General overview

The general idea of the local search heuristic is to check for every container that is poorly-placed if there is a stack to move that container to such that the container is correctly-placed and the objective function improves. We call this container the *investigated container*. Similar to the LPFH of Expósito-Izquierdo et al. (2012) for the CPMP, we start with investigating the movement of the highest time frame containers. These containers are the most difficult containers to place in a correct position because they can only be placed at the bottom of a stack. Let us call the stack to which we try to move the investigated container, the *destination stack*. We try every stack as possible destination stack, and if there is a stack that yields an improvement, then the investigated container is moved to the stack that gives the largest improvement. If, for none of the stacks, the investigated containers' movement gives an improvement in the objective function, then the next container is considered.

For the investigated container to be placed correctly in the destination stack, two types of containers need to be moved to other stacks. First of all, the investigated container might not be at the top of a stack, so the containers above the investigated container need to be moved to other stacks. Furthermore, to place the investigated container correctly in the destination stack, it might be needed to remove containers from that destination stack. Ideally, we would like to place both types of containers in a stack in which all containers are correctly-placed and in which itself is also correctly-placed. In that case, likely,

Algorithm 6.1: Local search heuristic for pre-processing phase of the SCRPPP.

Input: Bay B,0 $0 < \alpha < 1$, and relocation policy π.
Initialize pre-processing moves $\mathcal{P} = \emptyset$.
for $p = Z, Z-1, \ldots, 1$ **do**
  $A_p = \{c \in \mathcal{C} : t(c) = p\}$
  for $i = 1, \ldots, |A_p|$ **do**
    Select randomly a container c with time frame p that is not placed correctly,
    Use Algorithm 6.4 to get $f(B, \pi)$.
    for $s \in \mathcal{S}$ **do**
      Move container c to stack s according to Algorithms 6.2 or 6.3.
      Let $m(s)$ be the number of performed pre-processing moves and B_s the resulting bay.
      Use Algorithm 6.4 to estimate $f(B_s, \pi)$.
      $l(s) = f(B, \pi) - (\alpha m(s) + f(B_s, \pi))$.
    end
    if $\max_{s \in \mathcal{S}}\{l(s)\} > 0$ **then**
      Place container c in stack $s' = \arg\max_{s \in \mathcal{S}}\{l(s)\}$.
      Add the respectively pre-processing moves to $\mathcal{P}$.
      The resulting bay is the new bay B.
    end
    $A_p = A_p \setminus \{c\}$.
  end
end
Output: Set of pre-processing moves $\mathcal{P}$.

we do not need to move the container a second time. If such a stack does not exist, the container is placed in a stack where the minimum time frame is as low as possible. That decision is based on the fact that in this stack, the fewest containers can be correctly-placed.

Detailed description

We now give a more detailed description of the local search heuristic. The structure of the local search is given in Figure 6.1. Algorithm 6.1 is the main algorithm, and in this algorithm, it is decided which container is the investigated container and to which stack it should be moved. In Algorithms 6.2 and 6.3, it is decided how all containers should be moved such that the investigated container can be placed in the destination stack. The difference between Algorithms 6.2 and 6.3 is that Algorithm 6.2 is used if the destination stack is different from the stack in which the investigated container is located and Algorithm 6.3 if these two stacks are the same.

In Algorithm 6.1, the containers are considered in decreasing order of their time frame. If there are multiple containers with the same time frame, a random container is selected. We use Algorithm 6.4 to get an estimate for the number of relocation moves in a bay using relocation policy π ($f(B, \pi)$). We use Algorithm 6.2 and 6.3 to move the container to another stack. Afterward,

Algorithm 6.2: Pre-processing moves to move container c to a correct position in stack $s \neq s(c)$.

Input: Bay B with S stacks, container c, stack s, $0 < \alpha < 1$, $1 \leq \lambda_1 \leq S$ and $1 \leq \lambda_2 \leq S$.
$\mathcal{P} = \emptyset$.
$O_c = \{c' \in \mathcal{C} : s(c') = s(c) \wedge p(c') > p(c)\}$.
$M_c = \{c' \in \mathcal{C} : s(c') = s \wedge u(c') < t(c)\}$.
while $O_c \cup M_c \neq \emptyset$ **do**
  Let o and m be the top container of, respectively, the sets O_c and M_c.
  if $t(m) \geq t(o)$ **then**
    $c' = m$.
    $M_c = M_c \setminus \{c'\}$.
    $p_1 = s$.
  else
    $c' = o$.
    $O_c = O_c \setminus \{c'\}$.
    $p_1 = s(c)$.
  end
  if *There exists a stack* $s' \in \mathcal{S} \setminus (\{s(c)\} \cup \{s\})$ *for which* $n(s') < H$ *and* $I(s') > t(c)$ *and* $f(s', \pi) \leq \alpha$ **then**
    For all stacks $s' \in \mathcal{S} \setminus (\{s\} \cup \{s(c)\}$ with $n(s') < H$ and $f(s', \pi) \leq \alpha$, select randomly a stack s'' out of the at most λ_1 stacks with the smallest values for $I(s') > t$.
  else
    For all stacks $s' \in \mathcal{S} \setminus (\{s\} \cup \{s(c)\})$ with $n(s') < H$, select randomly a stack s'' out of the at most λ_2 stacks with the smallest values for $I(s')$.
  end
  Relocate container c' to stack s''.
  $\mathcal{P} = \mathcal{P} \cup \{(p_1, s'')\}$.
end
$\mathcal{P} = \mathcal{P} \cup \{(s(c), s)\}$.
Output: Pre-processing moves P and bay B.

Algorithm 6.4 is rerun to get an estimate for the number of relocation moves in the new bay. With that estimate, the improvement in the objective function ($I(s)$) can be calculated.

If the number of containers in a bay is less than or equal to $SH - 2(H-1)$, we know by Corollary 5.2 that every container can be placed in every stack using Algorithms 6.2 and 6.3. In case there are more than $SH - 2(H-1)$ containers in the bay, it might be infeasible to place a container in a certain stack. If a move is infeasible, the improvement is set to $-\infty$. If there is a stack that gives a strictly positive improvement, the container is moved to the stack that gives the best improvement.

In Algorithm 6.2, we need to move three different types of containers. The first type is container c itself that needs to move to its destination stack. However, it is only possible to access container c if we have moved the set of containers that are stacked on top of container c. We denote this second

set by O_c. Finally, container c needs to be positioned in stack s such that it does not need to be relocated. All containers that need to be removed such that container c can be placed correctly in stack s are in the set M_c. Before container c can be moved, the containers in O_c and M_c need to be moved to a different stack. From the containers in the sets O_c and M_c, only the top container can be relocated. In Algorithm 6.2, the container with the largest time frame of these two top containers is moved first. The container with the largest time frame is selected because if that container can be correctly-placed, the other container can be correctly-placed on top of it.

When a container is moved to a different stack, the algorithm checks first if there exists a stack that satisfies the following three conditions: the first condition is that the stack has an empty slot ($n(s') < H$). Secondly, the stack should only contain containers with a higher time interval than the container we are going to place in that stack $l(s') > t(c)$ because then the container does not need to be relocated. The final condition is that no pre-processing moves are performed for containers in that stack. This condition is equivalent to checking whether the expected number of relocation moves for that stack is fewer than or equal to α ($f(s', \pi) \leq \alpha$). A stack that satisfies these three conditions is referred to as a *correct stack*.

If there are multiple correct stacks, the stacks for which the smallest time frame is as small as possible are preferred because they are for fewer containers correct stacks. If there are fewer than λ_1 correct stacks, a random correct stack is selected. In case there are at least λ_1 correct stacks, we randomly pick one of the λ_1 stacks with the smallest minimum time frame. If there are no correct stacks for a container, we would like to place it in a stack with the minimum smallest time frame. This approach is similar to the LPFH, and its rationale is that a stack that has a container with a low time frame is less likely to be a correct stack for other containers. Hence, it is better to fill this stack with a container that has to be moved again than a stack with the potential to be a correct stack for more containers. We select randomly one of at most λ_2 stacks with the smallest minimum time frames.

Algorithm 6.3 is similar to Algorithm 6.2 when it comes to deciding the destination stack of a container. However, the sets of containers that need to be moved are different. Besides container c, there are again two different sets: S_1 and S_2. Set S_1 contains all containers placed on top of container c, and set S_2 consists of all containers placed below container c that need to be moved to another stack for container c to be placed in a correct position. Contrary to Algorithm 6.2, we do not need to determine which of the two top containers from the sets need to be moved first because the containers in set S_2 can only be moved when the containers from the set S_1 are moved to other stacks. After all containers in the set S_1 are moved to other stacks, container c needs to be

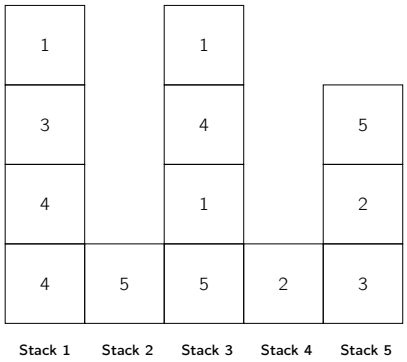
Algorithm 6.3: Pre-processing moves to move container c to a correct position in stack $s = s(c)$.

Input: Bay B with S stacks, container c, stack s, $0 < \alpha < 1$, $1 \leq \lambda_1 \leq S$ and $1 \leq \lambda_2 \leq S$.
$P = \emptyset$
$S_1 = \{c' \in \mathcal{C} : s(c') = s \wedge p(c') > p(c)\}$
$S_2 = \{c' \in \mathcal{C} : s(c') = s \wedge p(c') < p(c) \wedge u(c) < t(c)\}$
while $S_1 \neq \emptyset$ **do**
  Let c' be the top container of stack s.
  if *There exists a stack* $s' \in \mathcal{S} \setminus \{s(c)\}$ *for which* $n(s') < H$ *and* $l(s') > t(c')$ *and* $f(s') \leq \alpha$ **then**
    For all stacks $s' \in \mathcal{S} \setminus \{s\}$ with $n(s') < H$ and $f(B, s') \leq \alpha$, select randomly a stack s'' out of the at most λ_1 stacks with the smallest values for $l(s') > t(c')$.
  else
    For all stacks $s' \in \mathcal{S} \setminus s$ with $n(s') < H$, select randomly a stack s'' out of the the at most λ_2 stacks with the smallest values for $l(s')$.
  end
  Relocate container c' to stack s''.
  $S_1 = S_1 \setminus \{c'\}$.
  $P = P \cup \{(s, s'')\}$.
end
Move container c to the highest stack s''' with $n(s''') < H$ and update $s(c)$
$P = P \cup \{(s, s''')\}$.
while $S_2 \neq \emptyset$ **do**
  Let c' be the top container of stack s.
  if *There exists a stack* $s' \in \mathcal{S} \setminus (\{s(c)\} \cup \{s\})$ *for which* $l(s') > t(c')$ *and* $n(s') < H$ *and* $f(s') \leq \alpha$ **then**
    For all stacks $s' \in \mathcal{S} \setminus (\{s\} \cup \{s(c)\})$ with $n(s') < H$ and $f(B, s') \leq \alpha$, select randomly a stack s'' out of the at most λ_1 stacks with the smallest values for $l(s') > t(c')$.
  else
    For all stacks $s' \in \mathcal{S} \setminus (\{s\} \cup \{s(c)\})$ with $n(s') < H$, select randomly a stack s'' out of the at most λ_2 stacks with the smallest values for $l(s')$.
  end
  Relocate container c' to stack s''.
  $S_2 = S_2 \setminus \{c'\}$.
  $P = P \cup \{(s, s'')\}$.
end
$P = P \cup \{(s(c), s)\}$.

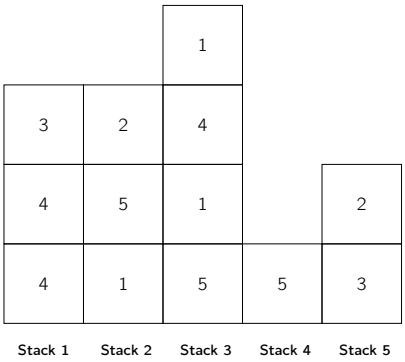
temporarily stored in another stack. The containers in set S_2 cannot be placed in that stack because then it is impossible to move container c back to stack s. As a consequence, in Algorithm 6.3 container c is placed in the highest stack because then the fewest slots for the containers in set S_2 are blocked.

Running example continued

We will illustrate Algorithms 6.1, 6.2, and 6.3 with the bay as given in Figure 5.2. Moreover, we set the value of α to $\alpha \geq \frac{1}{2}$, and the parameters λ_1 and λ_2 are



(a) Situation after the container with time frame 5 from the fourth stack of Figure 5.2 is placed in the second stack according to Algorithm 6.2.



(b) Situation after the container with time frame 5 from the fourth stack of Figure 5.2 is placed in the fourth stack according to Algorithm 6.3.

Figure 6.2 Two different outcomes after moving the container with time frame 5 in the fourth stack of the bay of Figure 5.2 in the pre-processing phase if $\alpha \geq \frac{1}{2}$.

both set to 2. Algorithm 6.1 first examines the containers with the largest time frame, in this case time frame 5. Let us say that the container with time frame 5 in stack 4 is the first container to be moved in the pre-processing phase. Furthermore, let us denote this container by c. We will first discuss how this container is placed in the second stack using Algorithm 6.2, and then, how it is placed according to Algorithm 6.3 in the fourth stack. Figure 6.2(a) shows the situation after the placement of container c in the second stack, and the bay after container c being moved to the fourth stack is depicted in Figure 6.2(b).

If container c is placed in stack 2, then set O_c is the container with time frame 3 above container c, and the set M_c is equal to the two containers located in the second stack. Since the time frame of container c is larger than 1, also the bottom container of the second stack needs to be removed. The order in which the containers from the sets O_c and M_c are moved to other stacks is: first the container with time frame 5 from M_c, second the container with time frame 3 from O_c and finally the container with time frame 1 from M_c. When the first container from M_c needs to be moved to another stack, there is no correct stack. Hence, it is placed in one of the $\lambda_2 = 2$ stacks with the smallest time frames and an empty slot. Since stack 3 has no empty slot, there are only two stacks possible, namely 1 and 5. Let us say that the container is placed in the fifth stack. After that, the container with time frame 3 from the set O_c is moved. The expected number of relocation moves for the first stack is $\frac{1}{2}$, and because $\alpha \geq \frac{1}{2}$, the first stack is a correct stack for this container. Moreover, it is the only correct stack for this container, so the container is always placed

in that stack. Finally, the container with time frame 1 from M_c has stacks 1 and 5 as correct stacks. Let us assume that it is placed in the first stack, then the situation after container c is placed in the second stack is as given in Figure 6.2(a).

When container c is placed in the fourth stack, the set S_1 is the container with time frame 3 above container c, and the container with time frame 2 constitutes on its own the set S_2. Similar to the situation in Figure 6.2(a), the first stack is the only correct stack for the container with time frame 3. Afterward, container c is also placed in the first stack because that is the only stack that has a height of 3. After that, no correct stack exists for the container with time frame 2 of set S_2. It could be placed in both stack 2 and 5 because stacks 1 and 3 have reached their maximum height. In Figure 6.2(b), we have assumed that the container is placed in the second stack, after which container c could be placed back to the fourth stack.

In both Figures 6.2(a) and 6.2(b) four pre-processing moves have been performed from the situation as shown in Figure 5.2. The final bays for both scenarios share that container c is placed at the bottom of a stack and thus is not relocated. However, to examine which situation is better, we need to know the expected number of relocation moves for the complete bay. In the next section, a method is discussed of how this number can be estimated.

6.1.2 Estimation number of relocation moves

For the local search heuristic described in Section 6.1.1, a fast estimation method for a bay's number of expected relocation moves is needed. It is important to note that in the heuristic, as sketched in Figure 6.1, the relocation phase will not be solved to optimality, but instead, the EM heuristic is used. Hence, we are not interested in estimating the optimal number of relocation moves, but the number of moves for the EM heuristic. One way to obtain the expected number of relocation moves for a bay is to simulate a set of retrieval orders and use the EM heuristic to compute the average number of relocation moves. However, in the local search heuristic of Algorithm 6.1, we need to compute the expected number of relocation for many different bays, so a faster method is preferred. Moreover, simulation will inherently result in stochastic outcomes. We prefer to have a deterministic estimation since that is consistent if the estimation method is called multiple times in Algorithm 6.1. Therefore, we use a rule-based estimation method that is given in Algorithm 6.4.

The main idea of Algorithm 6.4 is to estimate the number of relocation moves needed for one specific container. If a container has only containers with a higher time frame underneath it, it is evident that it will never be relocated. However, if a container needs to be relocated, it is hard to get the exact

Algorithm 6.4: Rule-based algorithm to estimate the expected number of relocation moves for a bay B.

Input: Bay B
for *All containers* c *in* $\mathcal{C}$ **do**
  if $u(c) > t(c)$ **then**
    $q(c) = 1$.
  else
    if $u(c) = t(c)$ **then**
      $q(c) = 2$.
    else
      if $u(c) > \min_{s \in \mathcal{S}}\{h(s)\}$ *or* $t(c) < \max_{s \in \mathcal{S}}\{l(s)\}$ **then**
        $q(c) = 3$.
      else
        $q(c) = 4$.
      end
    end
  end
end
Output: $\sum_{c \in \mathcal{C}} 1.4 \cdot \mathbb{1}_{q(c)=4} + \mathbb{1}_{q(c)=3} + 0.5 \cdot \mathbb{1}_{q(c)=2}$.

number of relocation moves that will be needed for that container. In order to get a reasonable estimation of the number of relocation moves, we divide the containers into four categories, numbered 1 to 4. Containers in category 1 do not need to be relocated. For the containers in categories 2, 3, and 4, we expect less than one, precisely one, and more than one relocation move, respectively. It is important to note that we use the word 'expect' here because it is hard to compute exactly how often a container needs to be relocated.

In Algorithm 6.4, the containers are divided into different categories, and these categories are used to get an estimation for the expected number of relocation moves. Algorithm 6.4 iterates through all containers in a bay and consists of three if-statements. In the first if-statement, it is checked if the smallest time frame of the containers underneath container c is higher than the time frame of container c. If this is the case, container c will not be relocated, and it is assigned to category 1.

Secondly, if the smallest time frame of the containers underneath c is the same as the time frame of container c, container c is assigned to category 2. In this situation, it is unsure if container c will need to be relocated. If it is to be retrieved before the container(s) with the same time frame in the same stack, it does not need to be relocated. However, it might also happen that container c is retrieved after a container from the same stack and the same interval. In that case, container c will have to be relocated to another stack. Nevertheless, at that time, all containers with a lower time frame have already left the bay. Thus, container c can, most likely, be placed in a stack where it does not have to be relocated again. All in all, the expected number of relocation moves for

containers in category 2 is likely less than one.

After the first two if-statements, all containers that are not yet assigned to a category have at least one container with a lower time frame underneath them. These containers will always be relocated at least once. In the third if-statement, it is checked whether there may be a *good stack* available for container c at the first time it will be relocated. A good stack is defined as a stack in which container c will not need to be relocated a second time. It is hard to determine precisely, in an efficient way, whether there is a good stack for a container when it is relocated. This difficulty lies in the fact that we do not know the bay's layout before the container under consideration is relocated. Hence, we use two rules of thumb to check whether it is likely that a good stack is available for container c at the time it needs to be relocated.

The first rule of thumb exploits the fact that container c will be relocated at time $u(c)$. If there is a stack for which all containers have a lower time frame than $u(c)$, all containers from that stack will already be retrieved at time $u(c)$. Therefore, that stack might be empty at time $u(c)$, and container c could be placed in that stack without any further relocation moves. The second rule of thumb checks whether there exists a stack for which all time frames are higher than the time frame of container c. In case there exists such a stack, it is likely that container c can be placed on that stack without any further relocation moves. For both rules of thumb, it might be that the good stack is not available anymore at the time the container needs to be relocated because another container is already placed in that stack. If one of the two rules is satisfied, container c belongs to category 3. Otherwise, it falls under category 4.

After all containers have been assigned to a category, the number of relocation moves for a bay is estimated using the formula

$$\sum_{c \in \mathcal{C}} 1.4 \cdot \mathbb{1}_{q(c)=4} + \mathbb{1}_{q(c)=3} + 0.5 \cdot \mathbb{1}_{q(c)=2}. \tag{6.1}$$

The coefficients for the categories in this sum are both based on logical reasoning and numerical experiments. Below we will explain how these coefficients are derived. We know that the containers in category 1 will never be relocated, so they get a weight of 0 in the sum. To find a good coefficient for the containers of categories 2, 3, and 4, we have used linear regression. The 'true' number of relocation moves is determined by simulating 1,000 retrieval orders for each of the 1,440 instances for the SCRP as created by Ku and Arthanhari (2016) and using the EM heuristic to solve them. This average number of relocation moves for an instance will act as an independent variable. For each instance, the number of containers in categories 2, 3, and 4 are explanatory variables. Hence, 1,440 independent variables could be explained by three dependent variables.

If linear regression is used, we get the coefficients 0.515 for the category 2 containers, 1.035 for category 3, 1.39 for category 4, and an R^2 of 0.968. If we round the coefficients for the categories 2, 3 and 4 to respectively 0.5, 1, and 1.4 , then the R^2 only slightly decreases to 0.966. Therefore, we choose these values in Equation (6.1) and Algorithm 6.4.

The mean absolute percentage difference between the outcome of the simulations and Algorithm 6.4, which is only 5.16%. It should be noted that this is based on solving the original SCRP instances of Ku and Arthanhari (2016). Nevertheless, it is easier to predict the expected number of relocation moves after some pre-processing moves have been performed. Since the more pre-processing moves are made, the more containers are placed such that they do not need to be relocated. These containers belong to category 1, which is the only category for which we can derive the exact expected number of relocation moves. Hence, the further the local search heuristic proceeds, the more accurate the prediction using Algorithm 6.4 will be.

Running example continued

To illustrate Algorithm 6.4, the containers as previously given in Figure 5.2 are assigned to categories in Figure 6.3. The green containers are all containers from the first category. All green containers have no containers at all underneath them or only containers with a strictly higher time frame. The containers in the second category are colored yellow. For these containers, the container with the smallest time frame underneath them has the same time frame as they have. The fourth stack's top container belongs to the third category and is thus orange because the second criterion of the third if-statement is valid. The maximum of the smallest time frame of the first stack is 4, which is strictly smaller than 3. Hence, we expect that this container can be relocated to the first stack and afterward does not need to be relocated again. The red containers are from category 4 because they do not satisfy any of the conditions in the if-statements.

If Algorithm 6.4 is run for the bay of Figure 6.3, we get an estimation of the total number of relocation moves that is equal to: $3 \cdot 1.4 + 1 + 2 \cdot 0.5 = 6.2$. Since this is a small example, we can calculate the expected number of relocation moves used by the EM exactly, which is $6\frac{1}{3}$. All in all, the prediction of Algorithm 6.4 is rather accurate for this bay.

After a method to estimate the number of relocation moves is developed, we can look again at the two bays in Figure 6.2 to see if these pre-processing moves improve the original bay. In the bay in Figure 6.2(a), there are two containers from category 2, namely the upper container with time frame 4 in the first stack and the upper container with time frame 1 in the third stack.

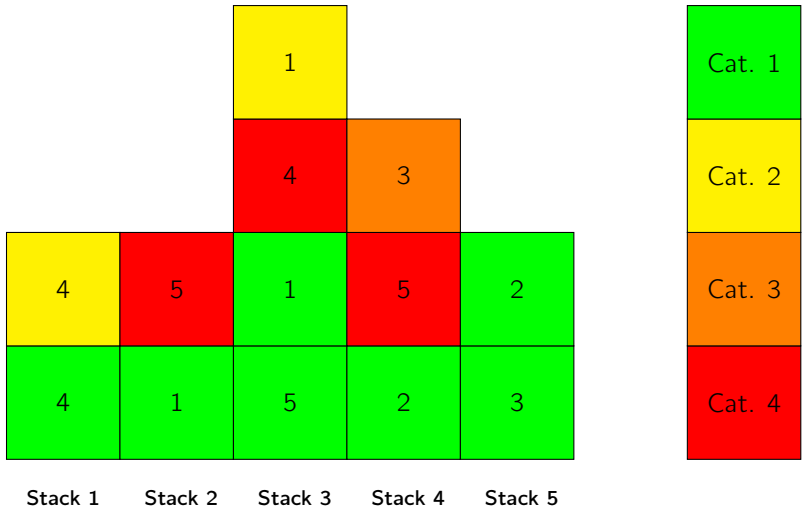


Figure 6.3 Categories of containers in the bay from Figure 5.2. The colors green, yellow, orange, and red correspond to category 1, 2, 3, and 4, respectively.

Furthermore, the container with time frame 4 in the third stack belongs to category 3. This container belongs to category 3 because in stack 2 are only containers with a larger time frame. Finally, the container with time frame 5 in stack 5 is in category 4 because no stack is expected to be empty if it has to be relocated, and no stacks contain only containers with a larger time frame. All in all, the number of relocation moves can be estimated by

$$1.4 + 1 + 2 \cdot 0.5 = 3.4.$$

In the bay of Figure 6.2(b), the same containers are in category 2 as in Figure 6.2(a). Besides that, the containers with time frame 4 in the third stack and time frame 2 in stack 2 are in category 3. Both of these containers would namely be well-placed if we would move them to stack 4. The container with time frame 5 in the second stack belongs to category 4. Therefore, estimated number of relocation moves for the bay of Figure 6.2(b) is

$$1 \cdot 1.4 + 2 \cdot 1 + 2 \cdot 0.5 = 4.4.$$

Recall that both the bays in Figures 6.2(a) and 6.2(b) were obtained after four pre-processing moves from Figure 5.2. Combined with the estimated number of pre-processing moves, the estimated objective of the bay in Figure 6.2(a) is $4\alpha + 3.4$ and for the bay in Figure 6.2(b) the estimated objective function is $4\alpha + 4.4$. The bay in Figure 6.2(a) is thus always preferred over the bay in Figure 6.2(b). Moreover, the four pre-processing moves that lead to the bay in

Figure 6.2(a) are only carried out in Algorithm 6.1 if $6.2 - (4\alpha + 3.4) > 0$, or stated otherwise if $\alpha < 0.7$.

6.1.3 Lower bound

The local search heuristic of Section 6.1.1 can be used as an upper bound for the branch-and-bound algorithm that is discussed in Section 6.1.4. However, besides a good upper bound, also a tight lower bound is needed. Multiple lower bounds are known for the (S)CRP, (see, e.g., Galle et al. (2018b) and Scholl et al. (2018)). One lower bound for the CRP is to count how many containers have a container with a lower time frame underneath them. These containers need to be relocated at least once, and all other containers will never be relocated. Hence, a lower bound for the deterministic CRP for bay B is

$$LB_{CRP}(B) = \sum_{c \in \mathcal{C}} \mathbb{1}_{t(c)<u(c)}.$$

The lower bound for the SCRP from Galle et al. (2018b) is similar but also takes into account the fact that two containers in a stack might have the same time frame. In this lower bound, the probability that a container is blocking another container is calculated. If for a container c holds that $t(c) < u(c)$, then it will never be relocated and if $t(c) > u(c)$ the probability of a relocation move is 1. If $t(c) = u(c)$, it is slightly more complex to calculate the desired probability. To derive that probability, let us define $m(c)$ as follows:

$$m(c) := |\{c' \in \mathcal{C} : s(c) = s(c') \wedge p(c) > p(c') \wedge u(c) = t(c) = t(c')\}|. \quad (6.2)$$

The number $m(c)$ represents the number of containers that are located underneath container c and that have the same time frame. The only possibility that container c does not have to be relocated, is when it is retrieved before any of the $m(c)$ containers. This scenario occurs with probability $\frac{1}{m(c)+1}$. Hence, the probability that container c does need to be relocated is $1 - \frac{1}{m(c)+1} = \frac{m(c)}{m(c)+1}$. Combining this expression with the lower bound for the CRP gives the lower bound for the SCRP of Galle et al. (2018b), $LB_{SCRP}(B)$, is formally given by

$$LB_{SCRP}(B) = \sum_{c \in \mathcal{C}} \mathbb{1}_{t(c)<u(c)} + \frac{m(c)}{m(c)+1}. \quad (6.3)$$

From this lower bound for the SCRP, we can derive a trivial lower bound for the SCRPPP. The container c needs to be moved at least $\mathbb{1}_{t(c)<u(c)} + \frac{m(c)}{m(c)+1}$ times. Ideally, this move is done in the pre-processing phase at a cost of α, thus a lower bound for the SCRPPP is $\alpha LB_{SCRP}(B)$.

One extra insight helps us get a tighter lower bound than $\alpha LB_{SCRP}(B)$. It is only possible to move a container in the pre-processing phase if all the

Algorithm 6.5: Lower bound for the SCRPPP.

Input: Bay B and $0 < \alpha < 1$.
for $c \in \mathcal{C}$ **do**
  if $t(c) > u(c)$ **then**
    $lb(c) = 1$.
  else
    if $t(c) = u(c)$ **then**
      $lb(c) = \frac{m(c)}{m(c)+1}$.
    else
      $lb(c) = 0$.
    end
  end
end
for $s \in \mathcal{S}$ **do**
  for $t = 0 : n(s) - 1$ **do**
    $g_1(t) := \sum_{c:s(c)=s \wedge p(c) \geq n(s)-t} lb(c)$.
    $g_2(t) := (n(s) - t + 1)\alpha$.
    if $g_2(t) < g_1(t)$ **then**
      for $c \in \mathcal{C} : s(c) = s \wedge p(c) \geq n(s) - t$ **do**
        $lb(c) = \alpha$.
      end
    end
  end
end
Output: $LB = \sum_{c \in \mathcal{C}} lb(c)$.

containers on top of it are also moved in the pre-processing phase. If only a single container c is considered, the lower bound for the number of moves for that container c is given by

$$\min \left\{ \mathbb{1}_{t(c)<u(c)} + \frac{m(c)}{m(c)+1}, \alpha \left(n(s(c)) - p(c) + 1 \right) \right\}. \tag{6.4}$$

The first part of the minimum is the expected number of relocation moves if the container is not moved in the pre-processing phase and is the same as in Equation (6.3). The second part of the minimum is the cost associated with moving the container in the pre-processing phase. The height of the stack in which container c is positioned is given by $n(s(c))$. As $p(c)$ indicates the position of container c in stack $s(c)$, in total $n(s(c)) - p(c) + 1$ containers are moved if container c is moved in the pre-processing phase. It is not possible to sum the expression in Equation (6.4) over all containers because if there are multiple containers in a stack for which $\alpha(n(s(c)) - p(c) + 1) \leq \mathbb{1}_{t(c)<u(c)} + \frac{m(c)}{m(c)+1}$ the pre-processing move of the top container of a stack is counted twice. Nevertheless, if a stack is considered from the top to the bottom, Equation (6.4) can be used to obtain a lower bound for an entire bay. In Algorithm 6.5, it is shown in detail how this lower bound is constructed.

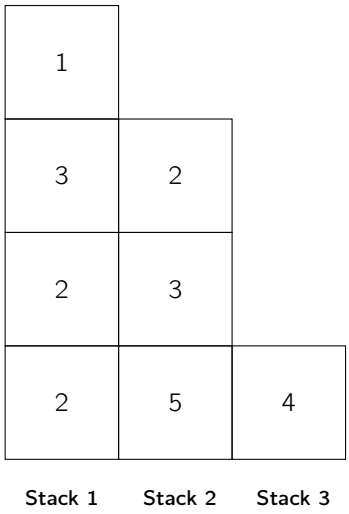


Figure 6.4 Example of a bay layout to illustrate the lower bound.

Algorithm 6.5 consists of two main for-loops. In the first for-loop, for every container, the probability of relocating is calculated in the same way as in Equation (6.3). In the second for-loop, for every container, the current lower bound for that container and the containers on top of it is calculated. If the sum of these lower bounds is larger than α times the number of containers under consideration, pre-processing all these containers is beneficial. Hence, for each of these containers, the lower bound is set to α. Otherwise, it is not beneficial to perform any pre-processing moves, and the lower bound remains the same.

The bay in Figure 6.4 will be used to illustrate the lower bound for the SCRPPP. In this bay, all but two containers have only containers with a larger time frame underneath them. The two containers with a positive number of relocation moves are the containers with time frame 3 in the first stack and the container with time frame 2 in stack 1 located on top of the other container with time frame 2. The probability that the container with time frame 3 is relocated is 1, and the probability that the container with time frame 2 is relocated is $\frac{1}{2}$. Therefore, the lower bound of Equation (6.3) for this bay is $1\frac{1}{2}$, so the trivial lower bound for the SCRPPP would be $1\frac{1}{2}\alpha$.

To move any container from the first stack, the top container of that stack also needs to be relocated. Thus, if we would like to move the container with time frame 3 from that stack in the pre-processing phase, at least two pre-processing moves are needed. Since these two pre-processing moves are only resulting in one relocation moves less, it is only beneficial if $\alpha \leq \frac{1}{2}$. So the

lower bound produced by Algorithm 6.5 for the bay from Figure 6.4 is 3α if $\alpha \leq \frac{1}{2}$ and $1\frac{1}{2}$ if $\alpha \geq \frac{1}{2}$.

We will close this section with Lemma 6.1 that shows that the lower bound that is given by Algorithm 6.5 is tight.

Lemma 6.1. *The lower bound for the optimal solution of the SCRPPP given by Algorithm 6.5 is tight.*

Proof. The example in Figure 6.4 can be used to prove this lemma. As stated previously, the lower bound for $0 < \alpha \leq \frac{1}{2}$ for this instance is 3α and for $\frac{1}{2} \leq \alpha < 1$ it is $1\frac{1}{2}$. It can be easily verified that the expected number of relocation moves for the bay in Figure 6.4 is $1\frac{1}{2}$, thus if one decides to perform no pre-processing moves, then the objective function for the SCRPPP is $1\frac{1}{2}$.

If pre-processing moves are performed in an optimal solution, it is to position the badly-placed containers. It is easy to see that the optimal way to position the container with time interval 3 in stack 1 in the pre-processing phase is by the pre-processing moves $\mathcal{P} = \{(1,2),(1,3)\}$. After these two pre-processing moves, the objective function is $2\alpha + \frac{1}{2}$. In a third pre-processing move $(1,3)$, it is also possible to position the top container with time frame 2 in a correct position. Hence, then the objective function will be 3α. These three pre-processing moves are optimal if $\alpha \leq \frac{1}{2}$. Hence, for any value of α, the lower bound from Algorithm 6.5 is tight. □

6.1.4 Branch-and-bound algorithm

In this section, a branch-and-bound algorithm is presented to solve the SCRPPP to optimality. Similar to the local search heuristic in Section 6.1.1, we focus on the pre-processing phase. Since the relocation phase is equivalent to the SCRP, we use the Pruning-Best-First-Search (PBFS) algorithm of Galle et al. (2018b) to solve the relocation phase to optimality. However, as the SCRP is NP-hard, this algorithm does not run in polynomial time. The idea of the branch-and-bound algorithm is to investigate all possible pre-processing moves in the most efficient way. Lemma 6.2 shows that an upper bound on the optimal solution can be used to derive a bound on the maximum number of pre-processing moves in an optimal solution.

Lemma 6.2. *The maximum number of pre-processing moves in the optimal solution is bounded from above by* $\left\lfloor \frac{UB}{\alpha} \right\rfloor$.

This lemma follows directly from the fact that performing p pre-processing moves yields an objective function of at least $p\alpha$. Let us denote the maximum number of pre-processing moves in the optimal solution by $d := \left\lfloor \frac{UB}{\alpha} \right\rfloor$. On top of that, we could perform at most $S(S-1)$ different pre-processing moves for

Algorithm 6.6: Branch-and-bound algorithm for the SCRPPP.

Input: Bay B, relocation policy π, and $0 < \alpha < 1$
Let B' be the bay after the pre-processing moves of Algorithm 6.1 with $\lambda_1 = \lambda_2 = 1$ using p pre-processing moves. $UB := p\alpha + f(B', \pi)$
$d := \left\lfloor \frac{UB}{\alpha} \right\rfloor$
$LB :=$ the lower bound from Algorithm 6.5.
$SOL := UB$
$Q := (B, d, LB)$
$I := \emptyset$
while $LB < SOL$ *and* $Q \neq \emptyset$ **do**
  Find the triplet $(B', d', LB') \in Q$ with the smallest LB'. If there are multiple bays, choose the bay with the smallest value of d'. In case there are still multiple bays left, choose the bay that was the earliest added to Q.
  Compute $f(B', \pi)$.
  $Q = Q \setminus \{(B', d', LB')\}$ and $I = I \cup \{(B', d', LB')\}$
  if $f(B', \pi) + (d - d')\alpha < SOL$ **then**
    $SOL = f(B', \pi) + (d - d')\alpha$
  end
  if $d' > 0$ **then**
    for $s_1, s_2 \in S$ *and* $s_1 \neq s_2$, $n(s_1) > 0$ *and* $n(s_2) < H$ **do**
      Compute the lower bound LB'' for bay $B'' = B'(s_1, s_2)$ using Algorithm 6.5.
      if $LB'' + (d - (d' - 1)) \cdot \alpha < SOL$ *and* $\{(B, d, LB) \in Q \cup I : B = B'' \wedge d \geq d' - 1\} = \emptyset$. **then**
        $Q = Q \cup \{(B'', d' - 1, B'')\}$
      end
    end
  end
end
Output: SOL

a given bay because, for every stack $s \in \mathcal{S}$, we could place its top container in every other stack. Therefore, in total there are at most $\sum_{i=0}^{d}(S(S-1))^i$ different solutions for the pre-processing phase of the SCRPPP. Even for small instances with 5 stacks, the number of possible solutions of the pre-processing phase is for $d = 7$ more than a billion. Fortunately, using the branch-and-bound algorithm in Algorithm 6.6, it is possible to reduce the number of solutions of the pre-processing phase for which we need to find the optimal number of relocation moves.

In Algorithm 6.6, the branch-and-bound algorithm to solve the SCRPPP to optimality is given. The first step of this algorithm is to compute an upper bound for the optimal solution. Firstly, we use the local search heuristic of Algorithm 6.1 with $\lambda_1 = \lambda_2 = 1$ for the pre-processing phase. The heuristic solution is a feasible solution for the SCRPPP and, thus, an upper bound for the optimal solution. Afterward, we estimate the expected number of relocation moves for the resulting bay B', denoted by $f(B', \pi)$. For Algorithm 6.6 to

give the optimal solution, both the relocation policy should be optimal, and the estimation of the number of relocation moves should be exact. Furthermore, Algorithm 6.5 is used to compute a lower bound for the optimal solution. In case the lower and upper bounds are the same, we have proven that the pre-processing moves of Algorithm 6.1 with $\lambda_1 = \lambda_2 = 1$ result in an optimal solution. Otherwise, we start by exploring all possible pre-processing moves in the while-loop in Algorithm 6.6.

In this while-loop, we have a set of candidate solutions Q and a set I of all solutions for which the optimal solution has already been computed. The while-loop ends either if we have found a set of pre-processing moves for which the objective function equals the lower bound of the initial bay or if the set Q is empty. In the while-loop, we select the element in Q for which we have the smallest lower bound. In the case of a tie, we select the solution for which the most pre-processing moves have been performed. If there are still multiple solutions, we select the element that was the first to be added to Q. We prefer an element whose lower bound is smaller because this solution is the solution that is the most likely to have the lowest objective function.

For the element in Q that we have selected, the objective function is computed. In case the objective function is lower than the best-known solution found so far, SOL is updated. Afterward, if the remaining number of pre-processing moves in the current solution is larger than zero, we investigate all possible pre-processing moves for the current solution. For every possible resulting bay, we firstly compute its lower bound. In the case this lower bound is smaller than OPT we check if the same bay with fewer pre-processing moves is not already in either Q or I. If the element is not already in Q or I, it needs to be investigated and added to Q. In checking whether a bay is already in Q or I, we make use of the fact that a specific bay is considered the same as a bay in which the same stacks are placed in a different order.

6.2 Numerical results

In this section, we will use numerical experiments to check the quality of the local search heuristic and the branch-and-bound algorithm presented in Section 6.1. Both these methods will be evaluated according to their solution quality and running time. We will use the SCRP instances introduced by Ku and Arthanhari (2016) as our problem instances for the SCRPPP. This set of instances consists of 1,440 instances with the number of stacks ranging from 5 to 10. Furthermore, the maximum stack height ranges from 3 to 6. For half of the instances, the number of containers is half of the available slots, and the other half of the instances have a fill rate of $\frac{2}{3}$. For each specific combination of the number of stacks, maximum stack height, and fill rate, there are 30

instances. All these instances satisfy the condition of Corollary 5.2, which is that the number of containers is smaller than $SH - 2(H - 1)$. Therefore, the local search heuristic can be applied to all instances without checking whether a move is feasible.

The remainder of this section is organized as follows. First, we will investigate in Section 6.2.1 the maximum size of an instance that can be solved to optimality by the branch-and-bound algorithm. Second, in Section 6.2.2, we will compare the results of the local search heuristic with the optimal solution. Finally, in Section 6.2.3, the solutions for the SCRPPP will be compared with the solutions for the SCRP and CPMP to see the benefits of pre-processing moves.

6.2.1 Optimal solution

To test the efficiency of the branch-and-bound algorithm of Section 6.1.4, we solve the instances of Ku and Arthanhari (2016) for different parameter settings. As stated before, the minimum number of stacks for these instances is 5, and the maximum is 10. We investigate the optimal solution for both this minimum and maximum number of stacks. For each combination of numbers of stacks, height of the stack and fill rate, 30 instances are solved for the following values of α: $\alpha = 0.25$, $\alpha = 0.5$, and $\alpha = 0.75$. For every single instance and value of α the running time is set to at most one hour.

Table 6.1 shows how many of the 30 instances were solved to optimality within the hour. Furthermore, the average running time of all instances that were solved to optimality is given. As one can see, for the instances with five stacks and a maximum stack height of three, all instances were solved and on average it took at most a few seconds. However, for instances with a maximum stack height of six, almost none of the instances could be solved to optimality within an hour. The running time of the branch-and-bound algorithm increases extremely fast as the stack size increases. It can be concluded from Table 6.1 that the branch-and-bound algorithm can solve small instances for the SCRPPP but that for larger instances, the running time is too large. However, it should also be noted that there is a significant fluctuation in the running time for different instances of the same size. For example, nineteen of the instance with $H = 4$, $S = 10$, $\alpha = 0.75$, and fill rate of 67% are not solved to optimality within an hour. Nevertheless, the eleven instances that are solved to optimality within an hour have an average running time of only 152.6 seconds.

The number of stacks has a much smaller influence on the running time than the maximum height of a stack. The bays with $S = 10$ and $T = 3$ and the bays with $S = 5$ and $T = 6$ have the same number of containers. However, the former can be solved much faster than the latter. This observation can be explained by the fact that the number of moves in a bay with lower stacks is

H	S	α	0.25		0.5		0.75	
		Fill rate	50%	67%	50%	67%	50%	67%
3	5	Solved	30/30	30/30	30/30	30/30	30/30	30/30
		Time (s)	0.03	3.1	0.03	0.9	0.02	0.5
	10	Solved	30/30	27/30	30/30	29/30	30/30	30/30
		Time (s)	1.6	265.1	1.0	90.3	0.1	3.3
4	5	Solved	30/30	28/30	30/30	30/30	30/30	30/30
		Time (s)	1.6	426.1	0.6	94.3	0.2	0.5
	10	Solved	22/30	3/30	25/30	7/30	29/30	11/30
		Time (s)	644.0	1702.2	106.7	828.8	182.8	152.6
5	5	Solved	27/30	4/30	29/30	4/30	30/30	13/30
		Time (s)	462.4	1174.4	82.9	495.5	45.4	552.2
	10	Solved	3/30	0/30	7/30	1/30	16/30	4/30
		Time (s)	1707.9	-	1593.8	518.9	303.5	2112.0
6	5	Solved	9/30	0/30	14/30	0/30	20/30	1/30
		Time (s)	1068.9	-	722.4	-	271.4	81.9
	10	Solved	0/30	0/30	2/30	0/30	6/30	0/30
		Time (s)	-	-	73.9	-	917.2	-

Table 6.1 Number of instances solved to optimality using Algorithm 6.6 and their running times.

smaller than in a bay with higher stacks. Hence, the lower and the upper bound for the SCRPPP are stronger if the number of stacks is lower. In case the value of the upper bound is larger, the value for d is also higher. Since the number of possible pre-processing moves is given by $\sum_{i=0}^{d} (S(S-1))^i$, the value d has a more significant impact on the size of the solution space than the value S.

Similar reasoning applies to instances with a fill rate of 67%. These instances need more moves than instances with the same number of stacks and maximum height, but a fill rate of 50%. Therefore, the upper bound and the value of d is larger for instances with a fill rate of 67%. As a result, the running time for instances with a higher fill rate is larger. For some values of S, H, and α the running time that is given in Table 6.1 is sometimes lower for instances with a fill rate of 67% than for 50%, see for instance $S = 10$, $H = 6$, and $\alpha = 0.75$. However, for these instances, fewer instances with a 67% fill rate are solved within an hour than instances with a 50% fill rate.

Furthermore, from Table 6.1, it can be concluded that the larger the value of α, the faster the branch-and-bound algorithm runs. An explanation for this result is that the maximum number of pre-processing moves that can be used in the optimal solution is bounded by $\lfloor \frac{UB}{\alpha} \rfloor$. The larger the value of α, the fewer pre-processing moves could be performed, and thus the fewer solutions need to be investigated in Algorithm 6.6. Finally, it should also be noted that the larger the value of α, the smaller the difference between the instances with a fill rate of 50% and 67%.

6.2.2 Local search heuristic

In the previous section, we have seen that the branch-and-bound algorithm can find the optimal solution for small instances but cannot solve larger instances within a reasonable time. This observation should not be surprising because we have shown that the SCRPPP is NP-hard. In this section, we will investigate whether the quality of the local search heuristic is close to the optimal solution for small instances and whether the heuristic running time is acceptable for larger instances.

The quality of the pre-processing moves that are performed in Algorithm 6.1 is compared with the optimal pre-processing moves in Algorithm 6.6. In order to make sure that we only look at the effect of the pre-processing moves, the relocation phase is solved, for both scenarios, to optimality according to the PBFS. We use two different variants of the local search heuristic. In the first variant, $\lambda_1 = \lambda_2 = 3$ is used and the second method uses $\lambda_1 = \lambda_2 = 1$. If $\lambda_1 = \lambda_2 = 3$, the local search heuristic is a randomized algorithm, and we refer to it as *Randomized Pre-Processing* (RPP). Since it is a randomized algorithm, the algorithm is run 150 times or stops after 100 runs without an improvement. This set-up is the same as the set-up of the LPFH to make a fair comparison possible in Section 6.2.3. After each of these runs, the optimal expected number of relocation moves for the bay after pre-processing is calculated. If Algorithm 6.1 uses $\lambda_1 = \lambda_2 = 1$ as parameters, it is a deterministic algorithm and we refer to it as *Deterministic Pre-Processing* (DPP). This deterministic algorithm is obviously only run once.

In Table 6.2, we give the objective function for the three methods described above for instances with five and ten stacks and a maximum height of three. The value of α is set to 0.25, 0.5 and 0.75. The first thing to note is that the RPP is performing better than the DPP, but that the difference between them is quite small. Another observation is that the smaller the value of α, the larger the difference between the heuristic values and the optimal solution. This makes sense because for larger values of α fewer pre-processing moves are performed in all solutions, and the relocation moves contribute for a large share to the objective function. Furthermore, the difference between the optimal and heuristic solution is larger for instances with ten stacks than five stacks. The possible number of pre-processing moves is larger for instances with ten stacks than five stacks, so it makes sense that the heuristic is performing worse for these.

Although the gap between the heuristic and the optimal solution differs per parameter setting, the average gap between the optimal branch-and-bound algorithm and the RPP+PBFS and DPP+PBFS over all instances reported in Table 6.2 is, respectively, 4.6% and 5.8%. It is important to realize that

H	S	α	0.25		0.5		0.75	
		Fill rate	50%	67%	50%	67%	50%	67%
3	5	B&B	0.642	1.250	1.222	2.328	1.503	2.836
		RPP+PBFS	0.725	1.356	1.222	2.439	1.508	2.916
		DPP+PBFS	0.750	1.434	1.222	2.469	1.511	2.942
	10	B&B	1.000	1.867	1.922	3.534	2.581	4.606
		RPP+PBFS	1.192	2.083	2.172	3.622	2.592	4.631
		DPP+PBFS	1.200	2.175	2.172	3.656	2.592	4.647

Table 6.2 Objective function under the optimal relocation policy for the bay obtained after pre-processing according to the branch-and-bound algorithm (B&B), the local search heuristic for $\lambda_1 = \lambda_2 = 1$ (DPP) and the local search heuristic with $\lambda_1 = \lambda_2 = 3$ (RPP).

the objective function for the SCRPPP is discontinuous. Consequently, if the heuristic does not find the optimal solution, its solution will likely be substantially worse than the optimal solution. For instance, if the pre-processing phase of the heuristic results in the same bay as the optimal solution but takes one pre-processing move extra, then the difference between the heuristic and optimal solution is already α.

After it is shown that the value of the objective function of the DPP and RPP is close to optimal for small instances, we will solve larger instances with the DPP and RPP. In Galle et al. (2018b), it is shown that the PBFS is not able to solve larger instances of the SCRP to optimality within an hour. Hence, we decided to use the EM heuristic for the relocation phase. We have chosen to use 1,000 simulations of the retrieval order to get the average number of relocation moves.

In Table 6.3, the value of the objective function for both the DPP+EM and RPP+EM is given. Furthermore, the average running time of the pre-processing phase and the relocation phase for the RPP+EM per instance is given. The deterministic DPP+EM is run only once. Therefore, the pre-processing and relocation phase running times are an order of 100 smaller than for the RPP+EM. As the total average running time of the DPP+EM is always less than a second for every instance, we have chosen to only report the running times of the RPP in combination with EM heuristic in Table 6.3. Contrary to Tables 6.1 and 6.2, not only the results of the instances with five and ten stacks are given, but all instances with stacks five up to ten are used.

The total running time of the RPP+EM heuristic increases when the maximum stack size and α increase. Notably, the time needed for the EM heuristic in the relocation phase is more substantial when $\alpha = 0.75$. For this value of α, fewer pre-processing moves are performed, and thus the relocation phase is harder. Although the running times of the RPP+EM are higher than for the DPP+EM, the objective for the RPP+EM is also smaller than for the DPP+EM.

H		α	0.25		0.5		0.75	
		Fill rate	50%	67%	50%	67%	50%	67%
3	DPP+EM	Objective	0.98	1.79	1.72	3.15	2.08	3.93
	RPP+EM	Objective	0.96	1.72	1.70	3.11	2.08	3.89
		Time pre-processing (s)	0.2	0.4	0.1	0.4	0.1	0.4
		Time relocation (s)	0.2	0.2	0.1	0.7	1.0	4.2
4	DPP+EM	Objective	1.86	3.45	3.25	5.88	4.07	7.47
	RPP+EM	Objective	1.80	3.11	3.22	5.66	4.04	7.32
		Time pre-processing (s)	0.5	0.7	0.4	1.0	0.4	1.2
		Time relocation (s)	0.3	0.8	0.4	4.0	3.4	19.7
5	DPP+EM	Objective	3.14	5.65	5.46	9.81	6.83	12.55
	RPP+EM	Objective	2.88	4.93	5.29	9.16	6.74	11.88
		Time pre-processing (s)	0.9	1.6	0.9	1.9	1.1	2.7
		Time relocation (s)	0.8	1.8	3.6	10.2	14.3	42.5
6	DPP+EM	Objective	4.39	8.69	7.65	14.86	9.70	18.66
	RPP+EM	Objective	3.93	7.03	7.17	13.00	9.41	17.20
		Time pre-processing (s)	1.5	2.5	1.4	3.7	1.7	4.2
		Time relocation (s)	1.9	4.1	4.8	18.9	25.7	67.3

Table 6.3 Objective function for the DPP+EM and RPP+EM and the average running time per instance for the RPP+EM.

For $\alpha = 0.25$, the difference between the objective function for the two methods is the largest. This is caused by the fact that for $\alpha = 0.25$, more containers are moved in the pre-processing phase, and for $\alpha = 0.75$, more containers are moved in the relocation phase. Furthermore, the more containers are in the bay, the more one can gain by performing different pre-processing moves, and thus the percentage difference between the RPP+EM and DPP+EM is bigger for larger values of H and a fill rate of 67%.

6.2.3 SCRP and CPMP

In this section, we will compare the newly proposed local search heuristic with two existing heuristics for the CPMP and SCRP. The LPFH is a heuristic for the CPMP and can be used for the pre-processing phase. The LPFH stops the moment no relocation moves are left. It is also possible to perform no relocation moves, after which we could apply the EM heuristic for the SCRP for the relocation phase. The results of these two methods are compared with the RPP+EM method in Table 6.4.

The EM method does not use any pre-processing moves, and thus the value of the objective function is independent of α. The LPFH does perform pre-processing moves, but the number of moves is not influenced by α because it always tries to find the fewest moves such that no relocation moves are needed. Hence, the value of the objective function is just a linear function of α. We see in Table 6.4 that for α is 0.25 and 0.5, the LPFH results in a lower value of the objective function than the EM heuristic. On the other hand, for $\alpha = 0.75$,

	α	0.25		0.5		0.75	
H	Fill rate	50%	67%	50%	67%	50%	67%
3	RPP+EM	0.96	1.72	1.70	3.11	2.08	3.89
	EM	2.47	4.34	2.47	4.34	2.47	4.34
	LPFH	0.84	1.62	1.68	3.24	2.52	4.86
4	RPP+EM	1.80	3.11	3.22	5.66	4.04	7.32
	EM	4.63	8.12	4.63	8.12	4.63	8.12
	LPFH	1.63	3.16	3.26	6.33	4.89	9.49
5	RPP+EM	2.88	4.93	5.29	9.16	6.74	12.55
	EM	7.48	13.00	7.48	13.00	7.48	13.00
	LPFH	2.84	5.48	5.67	10.95	8.51	16.43
6	RPP+EM	3.93	7.03	7.17	13.00	9.41	17.20
	EM	10.32	18.43	10.32	18.43	10.32	18.43
	LPFH	4.03	9.55	8.07	19.10	12.10	28.65

Table 6.4 Objective function for the RPP+EM, the EM and the LPFH.

the EM heuristic gives a better solution.

The RPP+EM outperforms both the EM and LPFH for almost all instances. On average, the objective function for RPP+EM is about 9% lower than that of the minimum of the EM and LPFH. Only for the instances with a small number of containers and $\alpha = 0.25$, the LPFH is slightly better. For $\alpha = 0.25$, it is beneficial to move a container in four pre-processing moves to a correct position. Hence, it is almost always possible for smaller instances to place all containers in the correct position. Since the LPFH is tailored for placing the containers in the correct position in the fewest moves, it is logical that it outperforms the RPP+EM.

Furthermore, we see in Table 6.4 that the difference between the RPP+EM and the EM and LPFH is bigger for large than for small instances. This difference can be explained by the fact that the more containers are in a bay, the more difficult it is to place a container in the correct position in the pre-processing phase. Moreover, if a container is moved in the relocation phase, then for larger instances, it has more containers on top of it. Hence, for larger instances, more pre-processing and relocation moves are needed, and the more can be gained by balancing those two types of moves. Similarly, the trade-off between pre-processing and relocation moves is more critical if $\alpha = 0.5$. Thus the difference between the RPP+EM and the EM and LPFH is also more significant for this value of α. For instance, for $\alpha = 0.5$, $H = 6$, and the fill rate of 67%, the improvement is even more than 40%.

6.3 Conclusion

In this chapter, we have presented the first solution methods for the SCRPPP. We have developed an optimal branch-and-bound algorithm to solve this prob-

lem. Since we have also proven that this problem is NP-hard, we have developed a local search heuristic to solve larger instances. That heuristic makes use of a newly developed estimation method for the number of relocation moves. For small instances, the heuristic gives close to optimal solutions. Furthermore, for larger instances and moderate values of α, a significant improvement is made compared to only moving containers in the pre-processing or relocation phase.

The optimal branch-and-bound algorithm that we propose could be improved if sharper upper and lower bounds are derived. If the upper bound is smaller, then the search tree could be pruned earlier. Furthermore, if the lower bound is larger, then the optimal number of relocation moves would need to be calculated for fewer bays. In the lower bound that we use, the moves needed to remove containers from a stack to place another container in that stack are not taken into account. It is not straightforward how one would incorporate these moves because if one stack is empty, it might be that many containers benefit from that and could be placed in that stack. Nevertheless, taking these moves into account would improve the lower bound.

The local search heuristic that we have proposed might improve if a better estimation method for the number of relocation moves is used. The quality of this estimation method is essential in our local search algorithm because it is used in deciding whether the bay after pre-processing moves has improved. The rule-based method is effective but simple. It might be that other estimation methods will result in a better prediction and improve the local search heuristic performance. If the local search heuristic is improved, a better upper bound for the branch-and-bound algorithm is also available.

In this chapter, we have seen that the SCRPPP generalizes the well-studied SCRP and CPMP. With the value of α, one has certain control over the length of the pre-processing phase. If $\alpha = 0.5$, then at most two pre-processing moves are performed to have one relocation move less. If there is enough time to make the pre-processing moves, then the SCRPPP can be a good problem to balance the extra pre-processing moves with the reduction in relocation moves. However, it might be that the crane is idle for a limited amount of time. In this case, it might be better to use the SCRPCPP, which solution methods are discussed in the next chapter.

7
Limited number of pre-processing moves

In this chapter, we solve the SCRPCPP, defined in Section 5.4.2. In Section 7.1, two heuristic and a branch-and-bound algorithm are developed. It is quite natural to extend the SCRPCPP to multiple bays, so we do that in Section 7.2. We use, in Section 7.3, numerical experiments to compare the solution methods and to see what can be gained by investigating multiple bays simultaneously. Finally, we conclude this chapter in Section 7.4.

7.1 Solution methods

In this section, we discuss three solution methods to solve the SCRPCPP. Similar to the SCRPPP, we only develop methods for the pre-processing phase because a good heuristic and optimal formulation for the relocation phase have already been given in Galle et al. (2018b). As we have shown in Section 5.4.3, the SCRPCPP is NP-hard, thus we first give a heuristic for the pre-processing phase in Section 7.1.1. In this heuristic, we need again the method of Section 6.1.2 to estimate the number of relocation moves for a given bay.

After we have developed a heuristic, we derive in Section 7.1.2 a lower bound for the SCRPCPP. This lower bound is used in a branch-and-bound method that is presented in Section 7.1.3. In this branch-and-bound algorithm, the number of expected relocation moves for a bay needs to be calculated. If the optimal expected number of relocation moves is calculated in each node of the branch-and-bound tree, then the branch-and-bound algorithm produces the optimal solution for the SCRPCPP. However, the number of relocation moves for a bay can also be estimated in a sub-optimal way using the rule-based method and that gives us another heuristic for the SCRPCPP.

7.1.1 Top Correct heuristic

In this section, a heuristic to find good pre-processing moves is described. We call this heuristic the *Top Correct* (TC) *heuristic*. The general idea of the

This chapter is based on B.G. Zweers, S. Bhulai, and R.D. van der Mei. Pre-processing a container yard under limited available time. *Computers & Operations Research*, 123-105045, 2020c.

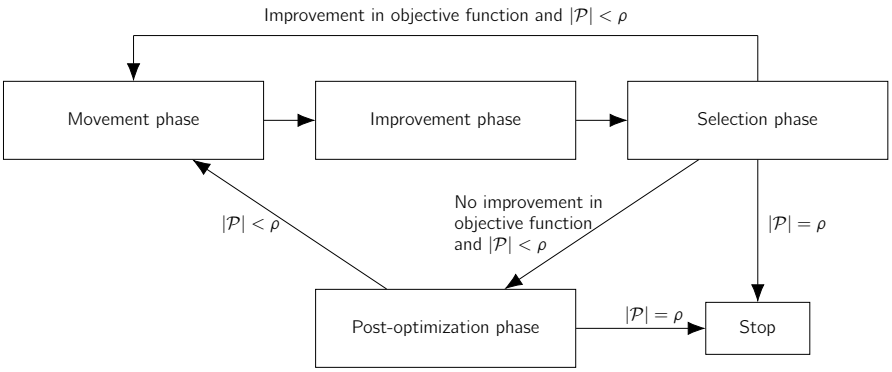


Figure 7.1 Overview of the TC heuristic.

TC heuristic is that we try for each combination of two stacks, to move the top container of one stack, called the *origin* stack, to the correct position in another stack, called the *destination* stack. It is also possible to place a container in a correct position in the same stack as it is currently located. In this case, the origin and destination stack are the same stack. After that, we calculate for each combination of origin and destination stacks the difference in the number of expected relocation moves and choose the pair that yields the largest decrease in the expected number of relocation moves without exceeding the maximum number of pre-processing moves.

In total, the TC heuristic consists of four different phases that we call (i) the *movement*, (ii) the *improvement*, (iii) the *selection*, and (iv) the *post-optimization* phase. In the first three stages, two different decisions can be made, thus the heuristic has $2^3 = 8$ different variants. Below, the four different phases are described. In Figure 7.1, an overview of the TC heuristic is given.

Phase 1: Movement

In the movement phase, the origin stack's top container is moved into a correct position in the destination stack. To place the container correctly in the destination stack, it might be necessary to move containers from the destination stack to other stacks before the top container of the origin stack can be moved to the destination stack. We refer to these moves as the *cleaning moves*. In cleaning moves, containers are ideally moved to stacks in which they are correctly-placed.

Out of the stacks in which a container is correctly-placed, the stacks with the fewest poorly-placed containers are selected. We prefer stacks in which few containers are poorly-placed because if a container is placed in a stack in which it is correctly-placed, while other containers are not correctly-placed, it

might still need to be moved a second time in the pre-processing phase. If there are multiple stacks with the same number of poorly-placed containers and in which the container that is moved will be correctly-placed, then the container is moved to stack in which the minimum time frame is the lowest. If it is not possible to move a container in the cleaning phase to a stack in which it will not need to be relocated a second time, then it is moved to a stack in which it is moved as late as possible in the relocation phase. This last step is equivalent to the procedure applied in the EM heuristic for the SCRP (Galle et al., 2018b).

We have defined a container to be correctly-placed if it only has containers with a lower time frame underneath it. However, if for a specific container, the minimum time frame of all containers underneath it is the *same* as its own time frame, then the probability that it does not need to be relocated is positive but smaller than one. We call this type of container *semi-correct*. In case the top container of the origin stack is allowed to be semi-correctly-placed in the destination stack, it could be that fewer cleaning moves are necessary, and still, the number of relocation moves is reduced. Hence, the two different variants for the movement phase are whether we require the top container of the origin stack to be semi-correct or correct in the destination stack.

Phase 2: Improvement

It could be that after the movement phase also other containers can be correctly-placed. For instance, when the container that is correctly-placed in the movement phase has a large time frame, many containers can be stacked upon that container. So after the movement phase, the TC heuristic tries to move poorly-placed containers to stacks in which they are correctly-placed. This phase is called the *improvement phase*, and the moves performed in this phase are called *improving moves*. Improving moves are equivalent to the "excellent moves" in Hottung and Tierney (2016).

The containers with the largest time frames are first moved in this phase because if these containers are correctly-placed in a stack, containers with a lower time frame can be correctly-placed on top of them. At first sight, it might seem wise always to perform the improving moves. However, in some scenarios, it is better not to perform improving moves and use the remaining pre-processing moves differently. Hence, the two variants of the improvement phase are: *performing improving* and *not performing improving* moves.

Phase 3: Selection

The number of pre-processing moves in the movement and improvement phase can be calculated for each combination of origin and destination stacks. There are in total S^2 possible combinations, but we only consider combinations of stacks for which fewer pre-processing moves were needed than the available

pre-processing moves. For all these combinations, the remaining number of relocation moves are calculated using the rule-based estimation method presented in Section 6.1.2. Hence, we can estimate how much the objective function changes if the top container of the origin stack is placed correctly in the destination stack for each combination of origin and destination stack. However, even when for all feasible combinations of origin and destination stacks, the number of expected relocation moves can be estimated, then it is still unclear which combination to select. It might be that one combination yields a strong decrease in the objective function and uses many pre-processing moves, while another combination only uses a few pre-processing moves, but has a smaller improvement.

In the TC heuristic selection phase, we distinguish two different selection methods: the *greedy* and the *ratio* selection method. In the greedy method, the combination of origin and destination stack that results in the largest decrease in the objective function is selected. On the other hand, in the ratio method, the improvement in the objective function is divided by the number of pre-processing moves needed to obtain the improvement per pre-processing move. As a result, the ratio selection method generally selects a solution in which a moderate improvement is obtained in a relatively small number of pre-processing moves. In contrast, in the greedy selection method, a solution is chosen that produces the biggest improvement in a large number of pre-processing moves. Although the improvement made by the ratio section method is lower than by the greedy selection method, it might be that many improving moves are possible for the bay that is selected. In that case, it is beneficial that a rather large number of pre-processing moves is still available for upcoming iterations of the improvement phase.

In both selection methods, there may be multiple combinations of origin and destination stack that result in the same improvement of the objective function. In that case, we select the leftmost origin stack and, after that, the leftmost destination stack. If no pre-processing move results in a decrease in the expected number of relocation moves, the TC heuristic continues to the post-optimizing phase. Otherwise, it starts again in the movement phase.

Phase 4: Post-optimization

The three phases described above all have three two different options and thus there are in total eight different variants possible. Each of these eight variants terminates the moment that there is no feasible combination of origin and destination stacks that result in an improvement of the objective function. However, it could be that there are still some pre-processing moves available that are unused. The idea of the post-optimizing phase is to use these moves to change the bay without changing the estimate of the objective function.

After the bay has changed, we again try the first three phases to see if an improvement can be made. It could be the case that a new top container can be easily placed correctly and with that improve the objective function.

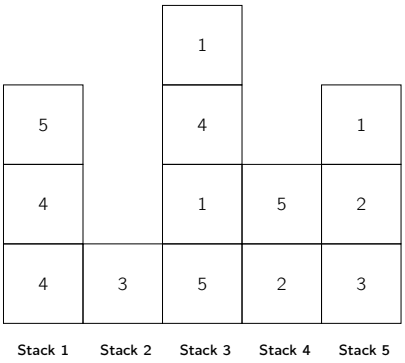
In the post-optimization phase, we use the concept of *Good-Good* (GG) moves introduced by Tanaka and Tierney (2018) in the context of the CPMP. In a GG move, a well-placed container is moved to a stack in which it is still well-placed. It is likely that a GG move does not change the number of relocation moves in a bay. A GG move can be beneficial if after that GG move new improving moves are possible. The idea behind a GG move is to make the most improving moves possible. Hence, the container that is moved in a GG move is the container for which the difference between its time frame and the container underneath it is maximized. The destination stack is chosen such that the difference between the time frame of the top container and the time frame of the container that is moved is minimized. We apply a single GG move and then again the first three phases to improve the resulting bay. If no improvement is possible, another GG move is applied until the moment that either all pre-processing moves are performed or no GG moves are available.

Running example continued

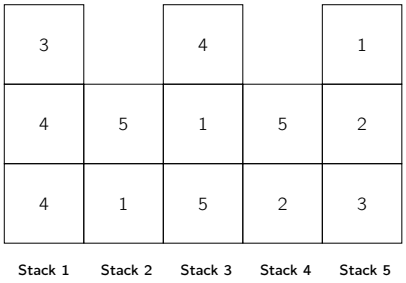
Let us apply the TC heuristic to the bay of our running example in Figure 5.2. In total, the TC heuristic produces eight solutions for this bay, but if the maximum number of pre-processing moves is three, then there are only two different solutions which are shown in Figure 7.2. All four variants in which the greedy selection method is used in the selection phase gives the bay in Figure 7.2(a), whereas the bay in Figure 7.2(b) is obtained using the ratio selection method.

Placing the top container of the fourth stack of the bay of Figure 5.2 at a correct position in the second stack needs three moves, and then the bay in Figure 7.2(a) is obtained. In this case, the containers in stack 2 need to be moved to a different stack. These moves are the cleaning moves. It is infeasible to position these containers in the third stack because that stack already has the maximum number of containers. Moreover, these containers can neither be positioned in stack 4 because we would like to move the fourth stack's top container. Hence, stacks 1 and 5 are the stacks to which they could be moved.

The first container to be moved from stack 2 is the container with time frame 5. This container cannot be correctly placed in any of the two stacks, and thus, it is positioned in the stack in stack 1 because, in this stack, it will be relocated in time interval 4, which is later than interval 2 of the fifth stack. Afterward, the container with time frame 1 can be positioned correctly in both stacks 1 and 5. Since stack 1 has two poorly-placed containers, and stack 5 does not contain any poorly-placed containers, it is moved to stack 5. In the



(a) Bay of Figure 5.2 after performing three pre-processing moves using greedy selection.



(b) Bay of Figure 5.2 after performing three pre-processing moves using ratio selection.

Figure 7.2 Two possible outcomes of the TC heuristic after three pre-processing moves for the bay of Figure 5.2.

final third move, the top container from stack 4 is moved to the second stack.

The estimate of the number of relocation moves for the initial bay of Figure 5.2 using the rule-based method is 6.2. For the bay of Figure 7.2(a), the estimate for the number of relocation moves is 4.8. This number is the lowest that can be obtained by moving the top container of a stack in a correct position while using at most three pre-processing moves. Hence, that is why the greedy method selects this bay as the best bay.

In the first move of the ratio method, the top container of the fourth stack of the initial bay of Figure 5.2 is placed in the first stack. The estimated number of relocation moves for the bay obtained after this bay is 5.2, and thus the improvement per pre-processing move is exactly one. To compare that with the bay in Figure 7.2(a), the improvement per move in that bay is $\frac{6.2-4.8}{3} \approx 0.47$.

After the first pre-processing move, only two pre-processing moves are left. The only top container that can be correctly positioned using one or two pre-processing moves is the top container of the middle stack. The improvement in the objective function is 0.5 if placed in stack 1, 4, or 5. Since stack 1 is the most left stack of these three stacks, the container is placed in that stack. Afterward, there is no top container that can be placed correctly in a stack with improving the objective function. Hence, the post-optimizing phase is entered.

The top container of the first stack is now the container with time frame 1 that has been moved in the second pre-processing move. This container is the only container for which we can apply a GG move. This container is still

correctly positioned if it is moved to the fourth or fifth stack. The fourth stack's top container has time frame 5, while the time frame of the fifth stack is only 2. Therefore, the container with time frame 1 is moved to the fifth stack, because then the difference with the top container is minimized. All in all, the bay of Figure 7.2(b) is obtained with the greedy selection method.

The optimal number of expected relocation moves for the bay in Figure 7.2(a) is $5\frac{1}{6}$, and for the bay in Figure 7.2(b), it is 5. Hence, for the bay of Figure 5.2, if three pre-processing moves are allowed, any variant of the TC heuristic that uses the ratio selection method produces a better solution than the variants that use the greedy selection method. However, for other bays or different maximum numbers of pre-processing moves, the greedy selection method might produce better pre-processing moves than the ratio selection method.

7.1.2 Lower bound

In this section, we present a lower bound for the SCRPCPP. For ease of explanation, we first describe the lower bound in the scenario in which each container has a unique time interval, i.e., the deterministic scenario. The lower bound for the stochastic setting is based on the lower bound for this deterministic setting, and it is described later.

Deterministic CRP

In the CRP, it is easy to check whether a container needs to be relocated or not. If a container has a container with a smaller time frame underneath it, it is relocated at least once, and otherwise, it is never relocated. Let us denote the lower bound for the number of relocation moves for container c by $lb(c)$, then $lb(c) = 1$ if $t(c) > u(c)$ and $lb(c) = 0$ otherwise. Hence, the lower bound for the number of relocation moves can be expressed as $\sum_{c \in \mathcal{C}} lb(c)$. So a lower bound for the number of relocation moves for the bay without any pre-processing moves can easily be calculated. The pre-processing moves are used to reduce the number of relocation moves, but each pre-processing move can reduce the lower bound for the relocation moves by at most one. Hence, a trivial lower bound for the CRP with pre-processing moves is $\max\{0, \sum_{c \in \mathcal{C}} lb(c) - \rho\}$.

However, it has to be possible for that lower bound to place each poorly-placed container in a correct position with only a single pre-processing move. There are two reasons why that might not be possible for a bay. The first reason is that there could be no stack with only containers with a higher time frame than the poorly-placed container. Secondly, it could be that the poorly-placed container has a well-placed container on top of it. In this case, the well-placed container needs to be moved before the poorly-placed container

can be accessed.

In case not all poorly-placed containers can become well-placed with a single pre-processing move, it might be possible to strengthen the lower bound. We divide the poorly-placed containers into two groups: the *one-move* and *multiple-moves* containers. The one-move containers can be moved to a correct position in a single pre-processing move, and the multiple-moves containers need more than one pre-processing move to be placed correctly. If the number of one-move containers is less than or equal to the number of pre-processing moves, then the lower bound for the CRP with pre-processing moves remains $\max\{0, \sum_{c\in\mathcal{C}} lb(c) - \rho\}$. Otherwise, at least one of the pre-processing moves is needed for a multiple-moves container, and thus the lower bound is equal to $\max\{0, \sum_{c\in\mathcal{C}} lb(c) - \rho + 1\}$. The improved lower bound is inspired by the lower bound of the CPMP given in Tanaka and Tierney (2018).

To check which containers are one-move containers, the bay needs to be investigated stack on stack level. If the top container of a stack is poorly-placed and can be placed correctly in another stack, it is a one-move container, and the container underneath it can be investigated. Otherwise, the top container is a multiple-moves container, and with that, all poorly-placed containers in that stack are multiple-moves containers. If the top container is a one-move container, then for the container underneath it, we can check if there is a stack in which it will be correctly-placed. If that is the case, that container is also a one-move container, and that stack's next container is considered. Otherwise, the container is a multiple-moves container, and the next stack is considered.

It is important to realize that no multiple-moves containers will become a one-move container after a one-move container has been moved in the pre-processing phase to a stack in which it is correctly-placed. If a one-move container has been moved to a stack in which it is correctly-placed, then the container with the minimum time frame of that stack is the one-move container that has just been moved. Hence, the minimum time frame of that stack has increased. Furthermore, the one-move container was poorly-placed in its origin stack, and thus the minimum time frame of that stack has to be lower than the time frame of the one-move container. Consequently, after a movement of a one-move container, the layout has not improved for any multiple-moves container, and no multiple-moves container will ever become a one-move container.

Running example continued

The bay of Figure 5.2 has been slightly adjusted to illustrate the lower bound for the deterministic case. In Figure 7.3, the containers are positioned in the same way as in Figure 5.2, but each container has a unique time interval. The

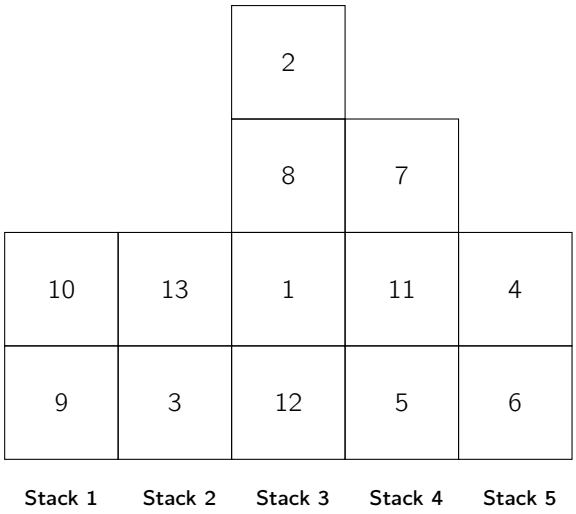


Figure 7.3 Deterministic variant of the bay in Figure 5.2 in which every container has a unique time interval.

order in which the containers in Figure 7.3 are retrieved is a possible retrieval order for the containers in Figure 5.2. Six containers need to be relocated at least once in the bay of Figure 7.3, namely the containers with time interval 2, 7, 8, 10, 11, and 13. Hence, the lower bound for the number of relocation moves for the bay in Figure 7.3 is six.

Out of these six containers, the containers with time frames 2, 7, and 8 are one-move containers, and the other containers are multiple-moves containers. When placed in stack 1, the container with time interval 7 does not need to be relocated anymore. Furthermore, the containers with time frame 2 can be correctly positioned in every other stack, and if that container is moved, the container with time frame 8 is examined. This container can be placed in a correct position in stack 1. It is important to realize that, in this specific example, the order in which the containers are moved in the pre-processing phase is crucial. If container 7 is moved to the first stack before container 8, then container 8 cannot be correctly positioned in that stack.

If the number of pre-processing moves is three or less, then the three one-move containers can be moved, and the lower bound equals six minus the number of pre-processing moves. For instance, the lower bound for two pre-processing moves equals four. If there are more than three pre-processing moves, then there is at least one multiple-moves container that needs to be moved to reduce the number of relocation moves further. Therefore, the lower bound equals six minus the number of pre-processing moves plus one. For

instance, the lower bound for five pre-processing moves is two. For seven and more pre-processing moves, the lower bound is zero.

Stochastic CRP

In the stochastic setting, both the lower bound for the number of expected relocation moves and the gain that can be achieved with pre-processing moves are different from the deterministic setting. However, the general idea of the lower bound is the same. In Algorithm 7.1, the lower bound for the SCRPCPP is described. The lower bound for the expected number of relocation moves is calculated in the first for-loop of Algorithm 7.1 and is the same as the expression in Equation (6.3).

In the second for-loop of Algorithm 7.1, it is determined whether a container is a one-move or a multiple-moves container. Two aspects are different in the stochastic setting than in the deterministic setting. In the deterministic setting, it was only beneficial to place a container c in a stack in which the minimum time frame of all containers was higher than the time frame of c. However, in the stochastic setting, it might also be beneficial to place container c in a stack in which the minimum time frame is the same as the time frame of container c. The value a in Algorithm 7.1 represents the relocation probability in the best stack for container c. Hence, the best improvement that can be made for moving container c in the pre-processing phase is given by $G(c)$, which is a value between 0 and 1.

A second difference with the deterministic setting is how one should determine if the next container of a stack should be checked for being a one-move container. The value $I(s)$ is the improvement that can be made by placing one-move containers from stack s to other stacks. Only if $I(s) + G(c)$ is larger than the number of containers that needs to be moved before container c can be moved plus one, then container c belongs to the one-move containers. Otherwise, the improvement that can be made by moving container c is less than the number of containers in its stack, plus one, and if there is one extra move needed, the container belongs to the multiple-moves containers.

After the improvement that can be made by only moving one-move containers is calculated, i.e., $\sum_{s \in \mathcal{S}} I(s)$, the final lower bound can be calculated. Similar to the deterministic case, if the number of pre-processing moves is less than the improvement made by one-move containers, then the lower bound equals the lower bound for the relocation moves of the initial bay minus the number of pre-processing moves. If the number of pre-processing moves lies between the improvement made by the one-move containers and that improvement plus one, then the lower bound equals the lower bound for the number of relocation moves for the initial bay minus the improvement of the one-move containers.

Algorithm 7.1: Lower bound for the objective function of the SCRPCPP.

Input: Bay B and maximum number of pre-processing moves ρ.
for $c \in \mathcal{C}$ **do**
  if $t(c) > u(c)$ **then**
    $lb(c) = 1$
  else
    if $t(c) = u(c)$ **then**
      $m(c) = |\{c' \in \mathcal{C} : s(c') = s(c) \wedge p(c') < p(c) \wedge t(c') = t(c) = u(c)\}|$
      $lb(c) = \frac{m(c)}{m(c)+1}$
    else
      $lb(c) = 0$
    end
  end
end
for $s \in \mathcal{S}$ **do**
  $t = n(s)$
  $l(s) = 0$
  while $t > 1$ **do**
    $c = C(t, s)$
    if $lb(c) > 0 \wedge t(c) \leq \max\{l(s') : s' \in \mathcal{S} \setminus s \wedge n(s') < H\}$ **then**
      $a = \min\{\frac{|\{c' \in \mathcal{C}: t(c) = t(c') \wedge s(c') = s'\}|}{|\{c' \in \mathcal{C}: t(c) = t(c') \wedge s(c') = s'\}|+1} : s' \in \mathcal{S} \setminus s \wedge t(c) \leq l(s') \wedge n(s') < H\}$
      $G(c) = \max\{lb(c) - a, 0\}$
      if $l(s) + G(c) > n(s) - t$ **then**
        $l(s) = l(s) + G(c)$
        $t = t - 1$
      else
        $t = 0$
      end
    else
      $t = 0$
    end
  end
end
if $\rho \leq \sum_{s \in \mathcal{S}} l(s)$ **then**
  $LB = \sum_{c \in \mathcal{C}} lb(c) - \rho$
else
  if $\rho \leq \sum_{s \in \mathcal{S}} l(s) + 1$ **then**
    $LB = \sum_{c \in \mathcal{C}} lb(c) - \sum_{s \in \mathcal{S}} l(s)$
  else
    $LB = \max\{\sum_{c \in \mathcal{C}} lb(c) - \rho + 1, 0\}$
  end
end
Output: LB.

Finally, in all other scenarios, the lower bound equals the lower bound for the relocation moves of the initial bay minus the number of pre-processing moves plus one.

Running example continued

Let us now compute the lower bound for the bay of the running example of Figure 5.2. The top container of the first and third stack has a relocation probability of $\frac{1}{2}$ because there are no containers with a strictly smaller time frame underneath them and only a single container with the same time frame. If the first stack consisted of three containers with time frame 4, then the top container would have had a relocation probability of $\frac{2}{3}$ and the middle container of $\frac{1}{2}$. For the containers with time frame 5 in stack 2, time frame 4 in stack 3, time frame 3 in stack 4, and time frame 5 in stack 4 at least one relocation move is needed and all other containers are correctly-placed. Hence, the lower bound for the relocation moves of the bay in Figure 5.2 is $4 + 2 \times \frac{1}{2} = 5$.

Although, the lower bound for the number of relocation moves for the top containers of the first two stacks is larger than zero, there is no stack in which they could be moved such that they are correctly-placed. Hence, the improvement that can be made for these two stacks is zero. The improvement for the fifth stack is also zero, because the top container is already correctly-placed. The top container of stack 4 can be placed correctly in the first stack, but there is no stack in which the container with time frame 5 underneath it can be placed such that the lower bound for the number of relocation moves for that container decreases. Therefore, the improvement for stack 4 is one.

Finally, we consider the most interesting stack, which is stack 3. If the top container of that stack is placed in stack 1, 4, or 5, then the lower bound for the relocation moves of that container decreases from $\frac{1}{2}$ to zero, thus the gain for that container is $\frac{1}{2}$. The container below it with time frame 4 has a lower bound for the number of relocation moves of 1, which can be improved by placing the container in stack 1. If it is placed there, then the lower bound for the number of relocation moves of that container is $\frac{2}{3}$, thus the gain obtained by moving that is $\frac{1}{3}$. The total gain of these two containers is $\frac{5}{6}$ for which two containers need to be moved. However, if a multiple-moves container is moved, then the total gain with two moves could be one. Therefore, in the third stack, only the top container is considered as a one-move container and the total improvement of that stack is $\frac{1}{2}$.

All in all, the top containers of stacks 3 and 4 are one-move containers, and moving these containers into a correct position reduces the number of relocation moves by at least $1\frac{1}{2}$. Given that the lower bound for the number of relocation moves for the original bay is five, the lower bound with a single

pre-processing move is four. With two pre-processing moves, the lower bound equals $3\frac{1}{2}$, and with three or more pre-processing moves, the lower bound is the maximum of zero and five minus the number of pre-processing moves plus one.

7.1.3 Branch-and-bound

The branch-and-bound algorithm of the SCRPCPP is similar to that of the SCRPPP given in Algorithm 6.6. In this section, we will only focus on the differences. The first difference is that the maximum depth of the search tree (d) is equal to the maximum number of pre-processing moves. Furthermore, the upper bound is calculated using the TC heuristic, and for the lower bound, we use Algorithm 7.1.

The final difference between the branch-and-bound algorithm for the SCRPPP and SCRPCPP is that in the latter, the best solutions are likely to be solutions with many pre-processing moves. In contrast, the SCRPPP usually has better solutions with fewer pre-processing moves. Hence, we change the order in which bays are investigated for the SCRPCPP compared to Algorithm 6.6. In Algorithm 6.6, first, the solution with the smallest lower bound was investigated, and in case of a tie, the bay with the most pre-processing moves was chosen. For the SCRPCPP, the first selection criterion is the number of pre-processing moves, and the lower bound is only a tie-breaker for bays with the same number of pre-processing moves.

Similar to the SCRPPP, also for the SCRCPP, the branch-and-bound algorithm will take too much computational time for practical instances for two reasons. The first reason is that the number of branches of the tree grows exponentially in the number of pre-processing moves. Second, the expected number of relocation moves has to be computed for each bay. However, we have seen in Section 6.1.2 that the number of relocation moves can also be estimated. If we use the rule-based estimation method of Section 6.1.2 instead of computing the optimal expected number of relocation moves, then the solution is not anymore guaranteed to be optimal, but the running time is also shorter. Hence, this could be an interesting heuristic for the SCRPCPP.

We refer to the branch-and-bound algorithm in which the optimal number of relocation moves is calculated as BB-O, and use BB-H to indicate the variant of Algorithm 6.6 in which the rule-based estimation method is used to calculate the number of relocation moves for a bay.

7.2 Extension to multiple bays

In this section, we show how we can apply the methods described in the previous section to a situation in which we consider multiple bays. We first give, in Section 7.2.1, the exact problem formulation for the extension to multiple bays.

Afterward, in Section 7.2.2, we present a method to solve this problem.

7.2.1 Problem formulation for multiple bays

Some containers in a bay become correctly-placed in one or a few pre-processing moves, while other containers require many pre-processing moves to decrease their expected number of relocation moves. Hence, the improvement in the objective function per pre-processing move becomes smaller if more pre-processing moves are performed. Therefore, if the crane driver is idle for a longer time, it could be better to visit multiple bays. Driving from one bay to the other is time-consuming, but if there are some containers that can be easily placed in a better position it might be worth moving the crane. Thus, in this section, we consider an extension of the SCRPCPP in which multiple bays are visited.

We assume that the bays are positioned in a line and are numbered from left to right as is shown in Figure 7.4. The crane in this figure is positioned above bay B_4, but at the beginning of the pre-processing phase it could be positioned above any bay. We call the bay above which the crane is positioned before the pre-processing phase the *starting bay*, which we denote by B_s. We do not require the crane to return to bay B_s at the end of the pre-processing phase, so any bay could be the last bay. In the extension to multiple bays, one needs to decide both the number of pre-processing moves to perform at each bay and a path that visits each bay for which at least one pre-processing move is performed. The total time needed to move the crane and perform the pre-processing times should again not exceed T. To reduce the problem's complexity, we do not allow the crane to take a container from one bay and place it in another bay.

The time needed to drive from bay B_i to bay B_j is given by t_{ij}. We restrict the travel times t_{ij} to the class of times that satisfy the following properties:

- $t_{ij} = t_{ji}$ for all B_i, B_j.
- $t_{ij} < t_{ik}$ for all $B_i < B_j < B_k$.
- $t_{ij} + t_{jk} - t_{ik} = t_{lm} + t_{mn} - t_{ln}$ for all $B_i < B_j < B_k$ and $B_l < B_m < B_n$.

In the first property, we assume the travel times to be symmetric. The second property states that if one bay is closer than another bay, the travel time is shorter. The last equality implies that the cost of stopping at a bay is independent of the bay itself and the origin and destination of the visited bays before and after that bay.

If one takes the layout of a container yard into account, as is illustrated in Figure 7.4, then the first two properties are realistic. With the last property, scenarios in which the crane drives at a constant speed and has no or a constant

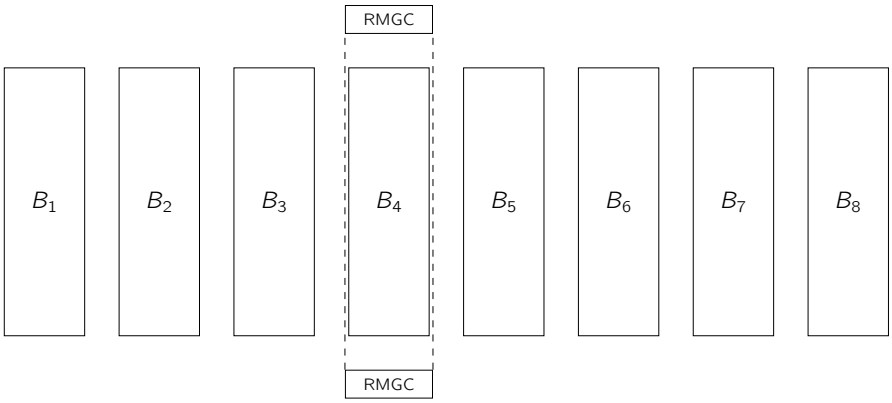


Figure 7.4 Illustration of the numbering of the bays (view from above).

time needed to accelerate en decelerate are included. In the literature, these types of travel times have been assumed before (Lee and Lee, 2010; Lin et al., 2015). Nevertheless, we do not include the fact that a crane can accelerate stronger if it has to drive for a longer distance.

These three properties for the travel time are useful to construct an optimal path for the crane. Given that the bays that need to be visited are known, then we can derive nice properties for the optimal path. In Lemma 7.1, it is shown how an optimal tour can be constructed.

Lemma 7.1. *There exists an optimal path P with the following properties:*

1. *The bays in between the leftmost and rightmost bay are visited in either increasing or decreasing order.*
2. *The endpoint of the tour is either the leftmost or rightmost visited bay.*
3. *The bay visited directly after the start bay is either the leftmost or rightmost visited bay.*

Proof. To prove that there exists an optimal path P with these three properties we go through these properties one by one. For Properties 1 and 2, we consider an optimal path P' for which these properties do not hold and show a contradiction. For the last property we show that an optimal path P' satisfying the first two but not the last property can be changed into a path P that satisfies all three properties without changing the objective function. We denote the leftmost and rightmost visited bay by, respectively, B_l and B_r.

1. For sake of contradiction, let us consider an optimal path P' in which the first property does not hold. Without loss of generality, let us assume

that bay B_l is visited before bay B_r. Let bay B_m be the first bay that is visited in path P' after bay B_n with $n > m$. On top of that, we denote the bay that is directly visited before bay B_n in path P' as B_k and the bay that is visited right after bay B_m as bay B_o.

Hence, the crane travels in path P' from bay B_k to B_n to B_m to B_o and this partial route takes $t_{kn} + t_{nm} + t_{mo}$ time units. If the order of bays B_m and B_n is swapped, then the total travel time to visit these bays is $t_{km} + t_{mn} + t_{no}$. As it is given that $t_{mn} = t_{nm}$ and that both $t_{km} < t_{kn}$ and $t_{no} < t_{mo}$, we know that swapping the bays B_k and B_l in path P' reduces the travel time of that tour, which contradicts the optimality of path P'.

2. For sake of contradiction, consider an optimal path P' that satisfies the first property, but that does not end at the rightmost or leftmost visited bay. Let us denote the bay at which path P' ends by B_e. Without loss of generality, we assume that the rightmost visited bay B_r is visited later than the leftmost visited bay B_l. Furthermore, because of the first property, we assume that the bays between B_l and B_r are visited in increasing order. We will show that visiting bay B_e before bay B_r reduces the cost of path P'. We distinguish between two different cases: the first case is the situation in which bay B_e is the only bay visited after bay B_r, and in the second case, multiple bays are visited after bay B_r.

   Consider the situation in which bay B_e is the only bay visited after bay B_r. Let us change the path P' into path P such that bay B_e is visited in between bay B_l and B_r. Using the first property, the optimal position in the path can be easily determined. Let us denote the bay that is visited before bay B_e in the new path by B_d and the bay that is visited after B_e in path P by B_f. Note that bay B_f might be equal to bay B_r. The increase in cost of visiting bay B_e in between bay B_e and B_f, instead of going directly from B_d to B_f equals $t_{de} + t_{ef} - t_{df}$. Moreover, after bay B_r, bay B_e does not need to be visited anymore which decreases the cost by t_{re}. Hence, the difference in cost by visiting bay B_e between bays B_d and B_f instead of after bay B_r equals

   $$t_{de} + t_{ef} - t_{df} - t_{re} < t_{ef} - t_{re} \leq 0.$$

   Here, the first inequality holds because $t_{de} < t_{df}$. The second inequality holds with equality if B_f is bay B_r. However, if B_f is not B_r, then by definition bay B_f is closer to bay B_e than B_r is to B_e and thus $t_{ef} < t_{re}$.

   Now consider the case in which there are one or multiple bays visited in between bay B_r and bay B_e. Let bay B_g be the bay visited directly before bay B_e and bay B_h be the bay directly before B_g. Again, we will show

that the cost of the tour decreases if bay B_e is visited between B_d and B_f. Note that now both B_f and B_h could be the same as bay B_r. The cost in path P' for going from B_d to B_f and from B_h via B_g to B_e equals

$$t_{df} + t_{hg} + t_{ge},$$

which can be rewritten into

$$\begin{aligned} t_{df} + t_{hg} + t_{ge} - t_{he} + t_{he} &= t_{df} + t_{de} + t_{ef} - t_{df} + t_{he} \\ &= t_{de} + t_{ef} + t_{he} \\ &> t_{de} + t_{ef} + t_{hg}. \end{aligned}$$

The first equality follows from the third property of the travel times and the final inequality from the fact that $t_{he} > t_{hg}$. The last expression is exactly the cost of the tour if bay B_e is visited in between B_d and B_f instead of after B_h. Therefore, the cost of P' can be reduced, which is in contradiction with the optimality of path P'.

3. Consider an optimal path P' in which the third property does not hold, but the first two are satisfied. Again we assume without loss of generality that bay B_l is visited before B_r. Let us define the bay that is visited directly after the start bay of the crane, namely B_s, in P' as B_t, and let bay B_u be the bay visited after bay B_t. Consider a path P in which bay B_t is not visited directly after bay B_s, but in the order that is suggested with the first property, between bays B_d and B_f. Again, note that bays B_u and B_d might be the same as B_l. The costs for going from B_s to B_t to B_u and from B_d to B_f in path P equals $t_{st} + t_{tu} + t_{df}$. In the equation below we see that these costs are exactly the same as $t_{dt} + t_{tf} + t_{su}$, which are the costs in path P' of visiting bay B_t between bays B_d and B_f and bay B_u directly after B_s:

$$\begin{aligned} t_{st} + t_{tu} + t_{df} &= t_{st} + t_{tu} - t_{su} + t_{su} + t_{df} \\ &= t_{dt} + t_{tf} - t_{df} + t_{su} + t_{df} \\ &= t_{dt} + t_{tf} + t_{su}. \end{aligned}$$

Hence, we can change path P' into a path P that satisfies the third property without changing the length of the path.

All in all, there exists an optimal path that satisfies all three properties described in the lemma. □

Two paths satisfy the properties of Lemma 7.1: one that ends at the bay B_l and one that ends at the bay B_r. The starting bay determines which of the two

paths is optimal. The only difference in the length of these two paths is in the time needed to travel from the starting bay to the second visited bay. By the third property of Lemma 7.1, this bay is either B_r or B_l. The second property ensures that the other extreme bay that is visited is the last bay of the path. By the first property, we know that all bays in between bay B_r and B_l are visited in the order they appear on the line. Since the travel times are symmetric, the costs of visiting the bays in between bays B_r and B_l are the same for starting in bay B_r or in bay B_l.

As the only difference between the two paths is the time needed to go from the starting bay to the second bay, the optimal path visits first of B_r and B_l the bay that is closest to B_s. Then the crane travels to the other bay, B_l or B_r, and stops at every bay in between that needs to be visited. Consider the bay in Figure 7.4 and assume that all bays of this figure should be visited. As bay B_1 is closer to B_4 than B_8, the optimal path visits after bay B_4 first B_1 and then all bays in increasing order without visiting bay B_4 again.

Let X_{kl} be a binary decision variable indicating whether the crane drives from bay B_k to bay B_l or not. A bay is never visited more than once because all moves that would be done in the second or later visits could also be done at the end of the first visit. Consequently, the constraint (5.3) for a single bay is replaced by the following constraint if there are multiple bays:

$$\sum_{k=1}^{m}\sum_{l=1}^{m} t_{kl}X_{kl} + \tau|\mathcal{P}| \leq T.$$

7.2.2 Solution method for multiple bays

In Sections 7.1.1 and 7.1.3, we have given an optimal algorithm and two heuristic methods to find pre-processing moves for a single bay. We use these methods in this section to derive a solution method for the extension to multiple bays. The idea of the method for multiple bays is to calculate, for each bay and number of pre-processing moves, the improvement that can be made by performing that number of pre-processing moves in that bay. For a single bay k and any number of λ_k pre-processing moves, the (optimal) pre-processing moves can be determined using the methods from Sections 7.1.1 and 7.1.3.

Let us define $S(B_k, \lambda_k)$ as (an estimation of) the expected number of relocation moves for bay B_k after λ_k pre-processing moves have been performed. Moreover, for a specific bay B_k, we can also determine the maximum number of pre-processing moves that will be done for that bay, which we denote by m_k. First of all, we know that m_k cannot exceed ρ because of the time limit. Second, if with λ_k pre-processing moves $S(B_k, \lambda_k) = 0$, then it is useless to perform more than λ_k pre-processing moves. In other

words, $m_k := \min\{\rho, \arg\min_\mu\{S(B_k, \mu) = 0\}\}$. Hence, the maximum number of pre-processing moves that can be performed in a bay is different per bay. Consequently, the values that λ_k can take is different for each bay k.

For each bay B_k we can calculate the value of $S(B_k, \lambda_k)$ for $\lambda_k = 0, 1, \ldots, m_k$. If the optimal pre-processing moves are performed and the value of $S(B_k, \lambda_k)$ is the optimal number of relocation moves, then $S(B_k, \lambda_k) \geq S(B_k, \lambda_k + 1)$. However, if either the pre-processing moves or the estimation of the relocation moves are not optimal, the inequality does not always hold. Nevertheless, if the estimates $S(B_k, \lambda_k)$ are calculated in an iterative order starting at $\lambda_k = 0$, then one can set the value $S(B_k, \lambda_k + 1)$ equal to $S(B_k, \lambda_k)$ if $S(B_k, \lambda_k + 1) > S(B_k, \lambda_k)$. Hence, we know that for each bay B_k it should hold that $S(B_k, 0) \geq S(B_k, 1) \geq \ldots, S(B_k, m_k)$.

Using the values for $S(B_k, \lambda_k)$, we can formulate an ILP to decide upon the number of pre-processing moves for each bay and the crane's route. Let $Y_{k\lambda_k}$ be a binary variable that equals one if λ_k pre-processing moves are performed in bay B_k. Furthermore, let X_{jk} be a binary variable indicating whether the crane travels from bay B_j to bay B_k or not. In Lemma 7.1, it has been shown that, given a set of bays that needs to be visited, an optimal path has some structural properties. Therefore, we know that either the leftmost bay (B_l) or rightmost bay (B_r) is the first bay to be visited after the initial bay and the other of the two is the last bay that is visited. We give the starting bay an index of 1. We could number the other bays either from left to right or from right to left. If we use the ILP below to solve both numberings, then the solution with the lowest objective function is optimal. All in all, the SCRPCPP for multiple bays can be formulated as the following ILP:

$$\min \quad \sum_{k=1}^{b} \sum_{\lambda_k=0}^{m_k} S(B_k, \lambda_k) Y_{k\lambda_k} \tag{7.1}$$

subject to

$$\sum_{\lambda_k=0}^{m_k} Y_{k\lambda_k} \leq 1 \qquad k = 1, \ldots, b \tag{7.2}$$

$$\sum_{\lambda_k=1}^{m_k} Y_{k\lambda_k} \leq \sum_{l=1}^{b} X_{lk} \qquad k = 2, \ldots, b \tag{7.3}$$

$$\sum_{o=1}^{b} X_{lo} \leq \sum_{k=1}^{b} X_{kl} \qquad l = 2, \ldots, b \tag{7.4}$$

$$\sum_{k=1}^{b} \sum_{l=1}^{b} t_{kl} X_{kl} + \sum_{k=1}^{b} \sum_{\lambda_k=0}^{m_k} \tau k Y_{k\lambda_k} \leq T \tag{7.5}$$

$$X_{ij} = 0 \qquad i = 1, \ldots, b \quad j = 1, \ldots, i \qquad (7.6)$$

$$X_{ij} \in \{0, 1\} \qquad i = 1, \ldots, b \quad j = 1, \ldots, b \qquad (7.7)$$

$$Y_{k\lambda_k} \in \{0, 1\} \qquad k = 1, \ldots, b \quad \lambda_k = 0, \ldots, m_k. \qquad (7.8)$$

In the objective function in (7.1), the sum of $S(B_k, \lambda_k)$ over all bays and possible pre-processing moves in each bay is taken. In constraint (7.2), it is forced that only a single number of pre-processing moves can be performed for each bay. The X- and Y-variables are coupled in constraint (7.3) by forcing all $Y_{k\lambda_k}$ to be zero if the crane does not visit bay k. Note that this does not hold for bay 1, because the crane starts in that bay. Constraint (7.4) ensures that the crane can only leave a bay if it has arrived at that specific bay. Again, this does not apply to the starting bay. The first sum in constraint (7.5) represents the total travel time of the crane between the bays, and the second sum of this constraint gives the total time the crane spends on moving containers inside the bays. Thus, in constraint (7.5), the total time the crane uses in the pre-processing phase is limited to T. In constraint (7.6), the properties of Lemma 7.1 are exploited. There exists an optimal tour in which the bays are visited in the way they are numbered. Finally, constraints (7.7) and (7.8) ensure that all variables are binary.

In the ILP above, the fact that the maximum number of pre-processing moves in a bay is different per bay is used. One could also decide to refrain from calculating m_k for each bay k and decide to set m_k equal to ρ. This decision has the advantage that notation can be simplified because λ_k and m_k do not depend on k anymore. However, this comes at the costs of having more Y-variables, and thus, we have decided not to do so.

7.3 Numerical results

In this section, numerical experiments are conducted to compare the three solution methods for a single bay presented in the previous section: the TC heuristic, the BB-H method, and the BB-O method. Moreover, the gain that can be obtained by investigating multiple bays is investigated. First, we compare in Section 7.3.1, the eight different variants of the TC heuristic. The results of the TC heuristic are compared with the branch-and-bound methods in Section 7.3.2. Finally, we compare the effects of extending the problem to multiple bays in Section 7.3.3.

Similar to Chapter 6, all the numerical experiments are done using the set of instances for the SCRP of Ku and Arthanhari (2016). We decided to restrict the experiments to the instances with five and ten stacks. Furthermore, we only focus on the instances with a fill rate of 67%, because they have more containers per stack. Hence, we can assume that the instances with a fill rate

1	Correct, No improvement, Ratio
2	Correct, No improvement, Greedy
3	Correct, Improvement, Ratio
4	Correct, Improvement, Greedy
5	Semi-correct, No improvement, Ratio
6	Semi-correct, No improvement, Greedy
7	Semi-correct, Improvement, Ratio
8	Semi-correct, Improvement, Greedy

Table 7.1 Numbering of the TC heuristic.

of 67% are more complicated than those with a fill rate of 50%. Finally, we solve every instance for three different number of pre-processing moves. We let the number of pre-processing moves depend on the number of containers in a bay. We set the number of pre-processing moves to 25%, 50%, and 75% of the total number of containers in a bay. It is important to note that if the number of pre-processing moves equals 25% of the total number of containers, some instances might already be reshuffled so that no relocation moves are needed. In that case, performing more pre-processing moves is not useful. However, there will also be instances for which still relocation moves will be needed if the number of pre-processing moves equals 75% of the total number of containers.

7.3.1 TC heuristic

In this section, the eight different variants of the TC heuristic, as discussed in Section 7.1.1, are compared with each other. In order to refer easily to the variants of the TC heuristic, we have given each variant of the TC heuristic a number which is given in Table 7.1. In Table 7.2, the effect of using eight variants of the TC heuristic is investigated. For each variant and instance, we calculated the number of expected relocation moves of the bay after the pre-processing phase. Calculating the optimal number of expected relocation moves for some of the instances takes already more than an hour. Hence, we decided to estimate the expected number of relocation moves by solving 10,000 realizations of the retrieval order using the EM heuristic.

On the rows of Table 7.2 are different subsets of the instances and on the columns are the number of variants of the TC heuristic that are used. Both the best combination of variants and the percentage difference with the objective function if all eight variants were used are given. For example, if all instances as described above, are considered, then heuristic 7 is the best if only a single variant is allowed. The percentage difference in objective function between the solution of heuristic 7 and the solution using all eight variants is 16.05%.

There are three main conclusions to be drawn from Table 7.2. The first is that the TC heuristic solution dramatically improves in the beginning by adding more variants, but that the contribution of the sixth, seventh, and eight best

Instances	# heuristics	1	2	3	4	5	6	7
All	% diff 8 heur.	16.05%	5.16%	1.75%	0.93%	0.40%	0.13%	0.01%
	Best heuristics	7	7,8	1,7,8	3,5,7,8	3,5,6,7,8	1,3,5,6,7,8	1,3,4,5,6,7,8
$\rho = 0.25C$	% diff 8 heur.	5.08%	1.52%	0.39%	0.17%	0.05%	0.01%	0.01%
	Best heuristics	8	5,8	5,7,8	2,5,7,8	2,3,5,7,8	3,4,5,6,7,8	1,3,4,5,6,7,8
$\rho = 0.5C$	% diff 8 heur.	20.92%	6.42%	1.62%	0.65%	0.13%	0.05%	0.01%
	Best heuristics	7	7,8	1,7,8	1,6,7,8	1,3,6,7,8	1,3,4,6,7,8	1,3,4,5,6,7,8
$\rho = 0.75C$	% diff 8 heur.	40.09%	16.82%	6.18%	3.26%	1.20%	0.17%	0.03%
	Best heuristics	3	3,8	3,5,8	1,3,7,8	1,3,6,7,8	1,3,5,6,7,8	1,3,4,5,6,7,8
$H = 3$	% diff 8 heur.	10,45%	2.55%	1.09%	0.55%	0.09%	0.00%	0.00%
	Best heuristics	8	7,8	1,7,8	3,5,7,8	2,3,5,7,8	2,3,5,6,7,8	2,3,4,5,6,7,8
$H = 4$	% diff 8 heur.	13.16%	4.26%	2.09%	1.03%	0.53%	0.06%	0.03%
	Best heuristics	8	3,8	3,6,8	3,5,6,8	3,5,6,7,8	3,4,5,6,7,8	1,3,4,5,6,7,8
$H = 5$	% diff 8 heur.	13.93%	3.81%	0.94%	0.41%	0.13%	0.02%	0.00%
	Best heuristics	8	5,8	1,7,8	1,3,7,8	1,3,6,7,8	1,3,5,6,7,8	1,3,4,5,6,7,8
$H = 6$	% diff 8 heur.	13.01%	5.44%	1.97%	0.87%	0.34%	0.11%	0.02%
	Best heuristics	7	7,8	1,7,8	3,5,7,8	3,5,6,7,8	1,3,5,6,7,8	1,3,4,5,6,7,8
$S = 5$	% diff 8 heur.	13.47%	4.51%	1.76%	0.71%	0.41%	0.13%	0.02%
	Best heur.	8	7,8	1,7,8	1,6,7,8	1,5,6,7,8	1,3,5,6,7,8	1,3,4,5,6,7,8
$S = 10$	% diff 8 heuristics	16.65%	5.78%	1.54%	0.64%	0.26%	0.13%	0.01%
	Best heuristics	7	7,8	3,5,8	3,5,7,8	3,5,6,7,8	3,4,5,6,7,8	1,3,4,5,6,7,8

Table 7.2 Comparison of different variants of the TC heuristic for different subsets of the instances.

variants is marginal. For example, if the number of pre-processing moves is 75% of the total number of containers in the bay ($\rho = 0.75C$), heuristic 3 is the best. The solution of heuristic 3 is more than 40% worse than the solution with all eight variants. However, when allowing three variants, the solution is already only 6.18% off from the solution using eight variants.

A second conclusion is that the number of stacks or the maximum height of a stack has no significant impact on the number of heuristics needed to obtain a solution close to the solution with eight variants. However, the number of pre-processing moves has a big impact on the number of heuristics needed. If the number of pre-processing moves is only 25% of the total number of containers, then the variants do not have many options to vary in the suggested pre-processing moves. In case the number of pre-processing moves is 75% of the total number of containers, then the number of possible different solutions is much bigger.

The third main conclusion is that no variant of the TC heuristic outperforms the others in all subsets of the instances. Heuristic 7 is the best for all instances, but that is partially caused by the fact that it performs well on the instances with more relocation moves, namely $H = 6$ and $S = 10$. Although heuristic 7 is the best for $S = 10$ if only a single heuristic is selected, the best three heuristics do not include 7. Another observation is that if two heuristics can be chosen, then heuristic 8 is always one of the two.

All in all, it is beneficial to include multiple variants in the TC heuristic. Although certain heuristics give better results than others, no variant is outperforming all other heuristics for every type of instance. In the remainder of the

		$\rho = 0$	$\rho = 0.25C$			$\rho = 0.5C$			$\rho = 0.75C$		
			BB-O	BB-H	TC	BB-O	BB-H	TC	BB-O	BB-H	TC
$H = 3$	Opt obj value	92.3	49.9	50.6	50.9	8.5	8.5	11.5	2	2	3
	EM obj value	92.3	49.9	50.6	50.9	8.5	8.5	11.5	2	2	3
	Avg run time (s.)	-	0.9	0.9	0.9	0.9	0.9	0.9	1.2	0.9	0.9
	Solved	-	30/30	30/30	30/30	30/30	30/30	30/30	30/30	30/30	30/30
$H = 4$	Opt obj value	167.4	95.2	97.5	99.5	39.6	41.3	45.3	7.0	7.2	25.3
	EM obj value	169.4	95.3	97.6	99.6	39.6	41.6	45.3	7.0	7.2	25.3
	Avg run time (s.)	-	3.7	1.5	1.4	3.2	1.5	1.4	53.7	7.9	1.4
	Solved	-	30/30	30/30	30/30	30/30	30/30	30/30	30/30	30/30	30/30
$H = 5$	Opt obj value	-	187.0	191.4	193.1	101.2	95.7	115.6	58.2	35.3	72.6
	EM obj value	314.4	191.2	202.0	198.6	102.8	98.7	117.1	58.8	35.3	72.8
	Avg run time (s.)	-	1200.6	2.2	2.2	1441.2	6.2	2.4	2864.7	1210.4	2.3
	Solved	-	22/30	30/30	30/30	22/30	30/30	30/30	8/30	22/30	30/30
$H = 6$	Opt obj value	-	-	-	-	199.9	145.5	210.1	136.7	92.8	140.7
	EM obj value	439.5	286.6	292.8	312.6	204.8	149.4	215.4	139.3	95.3	143.3
	Avg run time (s.)	-	3018.0	15.1	2.8	3491.0	674.6	2.8	3483.8	3371.1	3.0
	Solved	-	6/30	30/30	30/30	2/30	27/30	30/30	1/30	2/30	30/30

Table 7.3 Sum of objective function and average running times of the BB-O, BB-H, and TC solution methods over thirty instances for $S = 5$ and fill rate 67% for different stack heights and number of pre-processing moves.

paper, all eight variants of the TC heuristic will be applied to obtain the best possible results. However, if time does not allow to run all eight variants, only including the best four or five heuristics is probably the best option.

7.3.2 Branch-and-bound methods

In this section, the TC heuristic performance is compared with the two branch-and-bound methods introduced in Section 7.1.3: BB-H and BB-O. Again, we only focus on the set of instances of Ku and Arthanhari (2016) with five and ten stacks and a fill rate of 67%. In Table 7.3, the TC, BB-H, and BB-O methods are compared for the instances with five stacks. In Table 7.4, the same is done, but for the instances with ten stacks. Similar to the previous section, we set the maximum number of pre-processing moves to 25%, 50%, and 75% of the total number of containers in a bay. Moreover, we also include a column with $\rho = 0$ to investigate the improvement made in the pre-processing phase. For each combination of stack height and the maximum number of pre-processing moves, there are thirty instances, and for each instance, the maximum running time for the pre-processing phase is set to one hour.

We report in Tables 7.3 and 7.4 the average running time per instance. In calculating the average running time, we include the instances for which the pre-processing phase was stopped after one hour. For the TC heuristic, we ran all eight heuristics, which all have about the same running time. The running time of the TC heuristic reported in Tables 7.3 and 7.4 could be decreased by solving only a subset of the eight variants, but in that case, the solution quality also decreases as we have seen in Section 7.3.1. Furthermore, we show in the row 'Solved' for how many of the thirty instances the pre-processing phase terminated within one hour. This number is especially important for the

		$\rho = 0$	$\rho = 0.25C$			$\rho = 0.5C$			$\rho = 0.75C$		
			BB-O	BB-H	TC	BB-O	BB-H	TC	BB-O	BB-H	TC
$H = 3$	Opt obj value	158.2	40.5	40.7	50.0	1.0	1.0	2.5	0	0	0
	EM obj value	158.4	40.5	40.7	50.0	1.0	1.0	2.5	0	0	0
	Avg run time (s.)	-	3.3	2.7	2.6	150.5	144.0	3.5	4.1	4.1	4.1
	Solved	-	30/30	30/30	30/30	29/30	29/30	30/30	30/30	30/30	30/30
$H = 4$	Opt obj value	-	137.3	138.8	157.3	33.5	26.5	43.0	3.5	2.5	3.5
	EM obj value	308.8	137.5	139.3	157.6	33.5	26.5	43.0	3.5	2.5	3.5
	Avg run time (s.)	-	1028.3	13.2	4.8	2708.4	2061.7	5.8	487.9	487.2	6.9
	Solved	-	25/30	30/30	30/30	15/30	9/30	30/30	26/30	26/30	30/30
$H = 5$	Opt obj value	-	-	-	-	125.7	104.0	128.2	48.6	47.6	48.6
	EM obj value	485.7	271.9	253.4	284.5	125.7	104.0	128.2	48.6	47.6	48.6
	Avg run time (s.)	-	3094.4	331.8	7.0	3600.0	3564.3	9.4	3001.9	3001.7	10.7
	Solved	-	7/30	28/30	30/30	0/30	1/30	30/30	5/30	5/30	30/30
$H = 6$	Opt obj value	-	-	-	-	-	-	-	-	-	-
	EM obj value	696.0	425.5	394.8	432.5	230.6	198.4	230.6	111.3	109.8	111.3
	Avg run time (s.)	-	3600.0	1252.4	9.0	3600.0	3600.0	11.8	3600.0	3600.0	13.0
	Solved	-	0/30	22/30	30/30	0/30	0/30	30/30	0/30	0/30	30/30

Table 7.4 Sum of objective function and average running times of the BB-O, BB-H, and TC solution methods over thirty instances for $S = 10$ and fill rate 67% for different stack heights and number of pre-processing moves.

BB-O because if this method solves an instance within one hour, the solution is optimal. Therefore, if all thirty instances are solved, then the sum of objective values is optimal.

In order to calculate the objective function of the SCRPCPP, we need to calculate the remaining expected number of relocation moves after the pre-processing phase. We calculate these expected relocation moves in two ways: we use the optimal algorithm and the EM heuristic. We set the maximum running time for the relocation phase to one hour, and in that case, the relocation phase cannot be solved to optimality for the larger instances. Therefore, we also use the EM heuristic to obtain an estimate for the objective function for these instances. For the EM heuristic, we use 10,000 simulations to calculate the average number of relocation moves of an instance.

The first conclusion from Tables 7.3 and 7.4 is that the maximum height of the stacks has the biggest influence on the running time for all methods. For example, the number of containers in a bay is the same for $S = 5$ and $H = 6$, and $S = 10$ and $H = 3$. However, in the latter, the average running time of the optimal method is only a few seconds. In contrast, in the former, the optimal method could only solve six instances to optimality within one hour. Furthermore, the running times of the BB-O and BB-H methods also increase significantly with the maximum number of pre-processing moves.

For some parameter settings, for example, $S = 5$, $H = 5$, and $\rho = 0.5C$ and $S = 10$, $H = 4$, and $\rho = 0.25C$, the BB-H method can still solve the instances within seconds, whereas the BB-O method cannot find the optimal solution for every instance within one hour. There are other parameter settings, especially when $S = 10$, for which also the BB-H method cannot solve all instances within one hour. Consequently, for instances in which the maximum stack size

or the number of pre-processing moves is large, only the running time of the TC heuristic is small enough to be applied in practice.

We also see that if the optimal expected relocation moves could be calculated within one hour, the objective value approximated by the EM heuristic is always within 5% of the optimal objective function. If the number of containers in a bay is large, then it is impossible to calculate the optimal expected number of relocation. More interestingly, this is also the case when fewer pre-processing moves have been applied. For example, for $S = 5$ and $H = 6$, the optimal relocation moves could only be calculated for $\rho = 0.5C$ and $\rho = 0.75C$, but not for $\rho = 0$ and $\rho = 0.25C$. Hence, we can conclude that the relocation phase is simplified if more pre-processing moves are applied.

It is important to note that if the BB-O method does not solve all instances within one hour, it is often outperformed by the BB-H method. At first, this might be counterintuitive, but it can be explained by the fact that the BB-H method needs less time to investigate the quality of a node in the branch-and-bound tree. In the BB-O method, to investigate the quality of a node, the optimal number of expected relocation moves needs to be calculated, whereas only an estimation of the number of relocation moves is used in the BB-H method. As a result, the BB-H method visits more nodes in the branch-and-bound tree than the BB-O method and finds better solutions.

The instances used for the numerical experiments in this chapter are the same as in Chapter 6. Although it does not make sense to compare the objective functions for the two different problems, it is interesting to investigate the optimal branch-and-bound algorithm's running time. In Table 6.1, the running time for the optimal branch-and-bound algorithm for the SCRPPP is given. If we compare these results with the running time of the BB-O in Tables 6.1 and 7.3, then we can conclude that we have found more optimal solutions for the SCRPCPP than for the SCRPPP. For example, for the instances with ten stacks and a fill rate of 67%, only a few optimal solutions for the SCRPPP were found within one hour if H was four or larger. In Table 7.4, we see that for $H = 4$, the BB-O still solves the majority of the instances.

We have not looked into the reason why the branch-and-bound algorithm is faster for the SCRPCPP than for the SCRPPP, but we hypothesize that the difference lies in the depth of the tree. For the SCRPCPP, the depth of the tree is given by the number of pre-processing moves, whereas for the SCRPPP, it is derived from dividing the upper bound by α. Especially for small values of α, the depth can be quite large. Furthermore, for the SCRPCPP, it is expected that the best solutions can be found in the tree's leaf nodes. Hence, we can investigate these solutions early in the search. However, for the SCRPPP, it is hard to say how many pre-processing moves are used in the optimal solution.

If one wants to compute the optimality gap of the BB-H or TC method, it is important to realize that this can be done in two different ways that give entirely different results. For instance, for $S = 5$, $H = 4$, and $\rho = 0.75C$, one could compare the values 7.0 and 25.3 and argue that the optimality gap of the TC heuristic is more than 300%. However, one could also look at the reduction of the objective function, compared with $\rho = 0$. In this situation, the optimality gap of the TC heuristic is only about 10%.

Comparing the objective function for different values of ρ, we can also conclude that the first pre-processing moves yield the largest reduction in the number of relocation moves. For instance, looking at the difference of the objective function between $\rho = 0$ and $\rho = 0.75C$, we see that about 40% to 50% of this improvement is made for $\rho = 0.25C$. Whereas the improvement made between $\rho = 0.5C$ and $\rho = 0.75C$ is only about 10% to 15%. The observation that the first pre-processing moves yield a bigger reduction in the objective function than later pre-processing moves supports the idea to investigate multiple bays.

7.3.3 Multiple bays

If the idea of pre-processing is extended to multiple bays, the time units that are used are essential. We use as closely as possible the time units used in previous works in the literature (Lee and Lee, 2010; Lin et al., 2015). We assume that the time to move a container inside a bay is 30 seconds. Furthermore, the time to accelerate and decelerate the crane is set to be 40 seconds, and the time to move the crane a single bay is 3.5 seconds per bay. For example, going with the crane to an adjacent bay takes 43.5 seconds, and to a bay that is separated by two other bays in between takes 50.5 seconds.

As we are the first to study any variant of the SCRP in a multiple bay setting, no benchmark instances are available. We have decided to use the same instances from Ku and Arthanhari (2016) as for the single-bay case. For each parameter setting, thirty single-bay instances are available. An instance for multiple bays consists out of twenty randomly selected instances of these thirty instances. We investigate the instances with five and ten stacks and with a maximum height of five containers. For both of these parameter settings, we have created thirty instances consisting of twenty bays. Furthermore, we compare the effect of the starting position of the crane. The crane can be positioned at the beginning or in the middle of the yard. If the crane is positioned at the beginning of the yard, fewer bays are located nearby than if it is located in the center of the yard. Finally, we use four different variants of T, namely $T = 300$, $T = 600$, $T = 900$ and $T = 1800$, which corresponds to five, ten, fifteen and thirty minutes.

In Table 7.5, the results of the ILP of Section 7.2.2 are shown for the

	Start crane	$S = 5$ Begin	$S = 5$ Middle	$S = 10$ Begin	$S = 10$ Middle
$T = 300$	# pre-proc. moves	7.80	7.63	7.73	7.70
	# bays visited	1.43	1.27	1.57	1.67
	Reduction # reloc. moves	9.52	9.45	9.71	10.09
$T = 600$	# pre-proc. moves	15.73	15.70	15.77	15.70
	# bays visited	2.93	2.70	2.60	2.83
	Reduction # reloc. moves	19.05	18.75	19.37	19.62
$T = 900$	# pre-proc. moves	24.40	24.10	23.87	24.37
	# bays visited	3.83	3.70	3.80	3.83
	Reduction # reloc. moves	28.11	27.51	28.67	28.97
$T = 1800$	# pre-proc. moves	48.30	48.43	49.70	49.83
	# bays visited	8.03	7.77	7.00	6.77
	Reduction # reloc. moves	53.26	52.49	54.76	54.53

Table 7.5 The average number of pre-processing moves, number of bays visited and the improvement made in the objective function for different values of T, S, and the starting position of the crane.

parameter set described above. For the pre-processing phase, the TC heuristic is used, and to estimate the number of relocation moves, we have used the EM heuristic based on 10,000 simulations. We have chosen to do so because each bay has to be solved with possibly a large number of pre-processing moves. One conclusion that can be drawn from Table 7.5 is that the results are somewhat similar for instances with five and ten stacks per bay. However, with ten stacks, the number of relocation moves can be reduced slightly more.

There are two possible explanations of why the objective function can improve more for instances with ten stacks than five stacks. The first is that every initial bay has, on average, a larger number of relocation moves. Thus, performing more pre-processing moves in a single bay is likely to give a greater improvement in the objective function. This effect is the most clearly seen for $T = 1800$. For $S = 10$, more pre-processing moves per bay are performed than for $S = 5$.

A second explanation is that in a bay with ten stacks, a poorly-placed container is more likely to be placed correctly using a single pre-processing move than in a bay with five stacks. This property is caused by the fact that there are more potential stacks in a bay with ten stacks than in a bay with five stacks. Consequently, if $S = 5$, it is often the case that no pre-processing moves are made in the starting bay, whereas if $S = 10$, usually at least a few pre-processing moves are performed in the starting bay. The advantage of performing pre-processing moves in the starting bay is that no travel time is needed. Hence, if the time is limited, for instance, for $T = 300$, we see that the instances with ten stacks have more bays visited.

If the crane is positioned at the center of the yard, then the other bays are on

average closer than if the crane starts at the beginning of the yard. This position might be beneficial if only one or two bays could be visited. Nevertheless, if either the leftmost or rightmost bay is close to the end at which the crane is positioned, then starting at the beginning of the yard is more beneficial, because the crane does not need to travel that far in the beginning. If the available time is larger and more bays are visited, the leftmost and rightmost visited bays are likely closer to the end of the yards. Hence, we see that for $T = 1800$, the improvement is larger if the crane is positioned at the beginning of the yard. Obviously, these results heavily depend on the values for t and τ.

Furthermore, as expected, the improvement per pre-processing move decreases if T increases. If T is small, only the very best pre-processing moves are performed, whereas if T gets larger, then also less profitable pre-processing moves are performed. However, even for $T = 1800$, the improvement per pre-processing move is still larger than one, meaning that one pre-processing move reduces, on average, the number of relocation moves with more than one. If one does not allow the crane to move and let it only be positioned above a single bay, this improvement is much lower. For instance, if $T = 900$, the crane could perform thirty pre-processing moves, which yields an average improvement of 10.22 for $S = 5$ and when $S = 10$ of 15.67. However, even when the available time is only 300, already a substantial improvement can be made. In that case, ten pre-processing moves can be performed for a single bay, which yields an improvement of 7.05 and 8.34 for, respectively, $S = 5$ and $S = 10$.

A final remark to be made concerns the running time of the extension to multiple bays. The ILP (7.1)-(7.8) runs in less than a second for the instances we have considered. Hence, the only time-consuming part is to calculate the values of $S(B_k, \lambda_k)$ for each bay B_k and the number of pre-processing moves λ_k. Here a trade-off has to be made between the solution quality and the running time. If one uses a single variant of the TC heuristic for the pre-processing phase and estimates the number of relocation moves by the rule-based method, then even for the largest instances calculating $S(B_k, \lambda_k)$ takes at most two seconds. Nevertheless, for better solution quality, more computing time is needed. As the values of $S(B_k, \lambda_k)$ do not depend on the other bays, it could be possible in practice to calculate them offline. Only if a container arrives or leaves in a bay, the values of $S(B_k, \lambda_k)$ have to be updated.

7.4 Conclusion

In this chapter, we have studied solution methods for the SCRPCPP. We have adjusted the branch-and-bound algorithm of the previous chapter to solve this problem to optimality. However, for the SCRPCPP, this method's running time is also too large to be used in practice. Therefore, we have also developed two

heuristics. The first heuristic, the TC heuristic, is fast, but the gap between its solution and the optimal solution is for more difficult instances large. The BB-H heuristic is based on the optimal branch-and-bound algorithm. However, as it does not calculate the optimal number of relocation moves for a bay, its solutions are not optimal. However, its running time is also significantly lower than the optimal algorithm.

We have also developed a method for the pre-processing phase of a complete yard consisting of multiple bays. Using multiple bays, one can make a significant improvement per pre-processing move even when a large amount of time is given for the pre-processing phase. A direction for further research could be to improve the formulation for multiple bays. In the current method, for each combination of bays and the possible number of pre-processing moves, one has to calculate the improvement, which could be time-consuming. A fast method to decide upon the allocation of the number of pre-processing moves to a bay based on the initial layout would improve the current method. Nevertheless, in allocating the pre-processing moves, one should also consider the position of the bay in the yard. Furthermore, this new method might allow for moving a container from one bay to the other.

The TC and BB-H heuristic rely on the rule-based estimation method to estimate the number of relocation in a bay. If a better estimation method is found, then the performance of both methods can be improved. Furthermore, at the moment, the BB-O and BB-H heuristic do not use advanced branching decisions, it might also be worth investigating if their performance could improve if better branching decisions were used. Finally, one could speed up the BB-H heuristic by only constructing a partial tree or using a beam search approach to focus only on the tree's promising parts. It has to be investigated if the solution quality of such a method is much lower than the current BB-H heuristic.

The BB-H and TC heuristic can also be an inspiration for heuristics for the SCRPPP. The idea of heuristically calculating the expected number of relocation moves in a branch-and-bound tree can also be applied for the SCRPPP. However, as the optimal branch-and-bound tree had a longer running time for the SCRPPP than for the SCRPCPP, it is expected that the same will hold for the BB-H. The TC heuristic and the local search heuristic for the SCRPPP share that they select the best solution out of multiple solutions. The difference is that the local search heuristic from Chapter 6 uses randomization to create different solutions, and in the TC heuristic, different decisions are taken.

Randomization has the advantage that it is easier to create many different solutions. However, the drawback is a lack of explainability and consistency. If a solution is randomly created, it is hard to tell how this solution was created. Moreover, it is difficult to investigate what steps are often taken in a good

solution. Finally, the heuristic might produce a different solution if it is run a second time for the same instance, which might be undesirable. These differences make it worthwhile to develop a randomized heuristic for the SCRPCPP and a deterministic multi-heuristic for the SCRPPP.

In the problem formulation for the SCRPPP and the SCRPCPP, we have assumed that an RMGC was used to stack containers. However, some terminals use different equipment, such as reach stackers, to handle the containers. The concept of pre-processing moves is still relevant even if different terminal equipment is used. Nevertheless, it might be that certain assumptions are not valid anymore. For example, the assumption that the time of relocating is independent of the stack to which a container is relocated.

8

Approximation algorithms for cluster capacitated problems

8.1 Introduction

All problems studied in the previous chapters are NP-hard, so we have developed heuristics to solve larger instances of these problems in a reasonable amount of time. The quality of these heuristics has been investigated using numerical experiments. Although these experiments are carefully conducted on a broad range of instances, it could be that one of the heuristics has bad performance on a new type of instance. Only for the two-stage heuristic in Chapter 2, we were able to provide a theoretical bound for the objective function (see Lemma 2.5). Nevertheless, that bound depends on the input parameter, so by choosing certain parameters, the gap between the optimal and heuristic solution could theoretically be arbitrarily bad.

In this chapter, we change our focus towards *approximation algorithms*, which are defined as follows (Williamson and Shmoys, 2010).

Definition 8.1 (Approximation algorithm)**.** *An α-approximation algorithm is a polynomial-time algorithm that, for all instances of the problem, produces a solution whose value is within a factor α of the optimal solution.*

The value α is referred to as the *approximation ratio* or *performance guarantee*. For maximization problems, the approximation ratio is always less than or equal to 1, whereas $\alpha \geq 1$ for minimization problems. For example, a 2-approximation algorithm returns, for any instance of a minimization problem, a solution that is at most twice the optimal solution.

A key problem in approximating the optimal solution is that we usually do not know the optimal solution. Fortunately, if a problem can be formulated as an ILP, then its linear relaxation provides a lower bound for the optimal solution for minimization problems and an upper bound for maximization problems. Hence,

This chapter is based on G. Schäfer and B.G. Zweers. Maximum coverage with cluster constraints: an LP-based approximation technique. In *Proceedings of Workshop on Approximation and Online Algorithms*, Lecture Notes in Computer Science, 2020. To appear.

an often-used technique is to round the fractional solution of the LP relaxation to an integral solution. In this rounding, one has to make sure that the value of the integral solution remains within a certain factor of the value of the LP relaxation and thus the optimal solution value. In this chapter, we will focus on the development of LP-based approximation algorithms.

Many problems addressed before in this dissertation share the characteristic of having multiple levels of capacity. For instance, consider the problems discussed in Chapters 2 and 3. The number of containers that could be transported on a barge was constrained by both the capacity of a barge and the number of containers loaded on a terminal. Another example is the extension of the SCRPCPP to multiple bays in Chapter 7. The number of pre-processing moves that could be performed in a single bay was both limited by the maximum time for the pre-processing phase and the number of pre-processing moves after which the bay is perfectly ordered.

Many classical problems in the literature encompass only a single tier of capacity constraints. For instance, more than 20 variants of the *Knapsack Problem* are discussed in Kellerer et al. (2004), but none of these have multiple levels of capacities. For the sake of concreteness, consider the *Multiple Knapsack Problem* (MKP). In this problem, the goal is to find a most profitable selection of items that can be assigned to the multiple knapsacks without violating their capacities, i.e., there is a single capacity constraint per knapsack that needs to be satisfied. This problem can be extended by partitioning the knapsacks into *clusters*, each imposing an additional capacity constraint on all knapsacks contained in it. This extension results in a new optimization problem that we call the *Multiple Knapsack Problem with Cluster Constraints* (MKPC).

In this chapter, we present a technique that can be used to extend LP-based approximation algorithms for capacitated problems to problems with an additional cluster capacity. Our idea is to extend the ILP formulation of the original problem (i.e., without cluster capacities) by incorporating the respective cluster capacity constraints. However, crucially, these new constraints are set up in such a way that an optimal solution to the LP relaxation of this formulation defines some *reduced capacities* for each knapsack individually. These constraints enable us to reduce the cluster capacitated problem to the respective original problem with knapsack constraints only. We then use an LP-based approximation algorithm for the original problem to round the optimal LP solution. This rounding requires some care because of the reduced capacities.

We apply this technique to three new problems, namely the MKPC, the *Maximum Coverage Problem with Cluster Constraints* (MCPC), and the *Capacitated Facility Location Problem with Cluster Constraints* (CFLPC). The MCPC is a generalization of the MKPC in which we are given a collection of

subsets of items. Each subset is associated with some cost, and each item has a profit. The goal is to determine a feasible assignment of a selection of the subsets to the knapsacks such that both the knapsack and the cluster capacities are not exceeded, and the total profit of all items covered by the selected subsets is maximized. For this problem, we derive a $\frac{1}{3}\left(1-\frac{1}{e}\right)$-approximation algorithm. In deriving the approximation algorithm for the MCPC, we use a new $\frac{1}{2}\left(1-\frac{1}{e}\right)$-approximation algorithm for the *Maximum Coverage Problem with Knapsack Constraints* (MCPK), which is the problem arising from MCPC without cluster constraints. In this algorithm, we adapt the *pipage rouding* technique of (Ageev and Sviridenko, 2004) to partially round the solution of the LP relaxation.

The MCPC is a generalization of the MKPC, and for the latter problem we can improve the approximation ratio from $\frac{1}{3}\left(1-\frac{1}{e}\right)$ to $\frac{1}{3}$. If the clusters satisfy a certain *isolation property*, this performance guarantee can be improved further to $\frac{1}{2}$ by applying a more sophisticated iterative rounding technique. This isolation property implies that after removing an arbitrary set of clusters, there always exists an *isolated cluster*. An isolated cluster is a cluster to which, in the LP relaxation, no items are assigned that are also assigned to smaller knapsacks.

For the MCPC and MKPC, we lose something in the approximation ratio compared to the problems without cluster capacity. However, the CFLPC is different because we do not lose anything in the performance guarantee for this problem. In Aardal et al. (2015), a $(4.562+\varepsilon)$-approximation algorithm is presented for the *Capacitated Facility Location Problem* (CFLP) with fixed opening costs. We will show that this algorithm can be used to derive a $(4.562+\varepsilon)$-approximation algorithm for the CFLPC.

The remainder of this chapter is organized as follows. First, we review in Section 8.2 the relevant literature. In Section 8.3, we give the preliminaries needed for formally describing the MCPC and its generalizations. In Section 8.4, we first study the MCKP and present a $\frac{1}{2}\left(1-\frac{1}{e}\right)$-approximation algorithm for this problem. Afterward, we show in Section 8.5 how this approach can be extended to a problem with cluster capacities, and give a $\frac{1}{3}\left(1-\frac{1}{e}\right)$-approximation algorithm for the MCPC. In Section 8.6, we show that the algorithm for the MCPC, gives a $\frac{1}{3}$-approximation algorithm for the MKPC. Furthermore, we show that we can improve the approximation ratio to $\frac{1}{2}$ for instances of the MKPC that satisfy a specific condition. The $(4.562+\varepsilon)$-approximation algorithm for the CFLPC is given in Section 8.7. Finally, we conclude this chapter in Section 8.8.

8.2 Literature review

The objective of the MCPC is a special case of a submodular function and there is a vast literature concerning the problem of maximizing a (monotone) submodular function. Nemhauser and Wolsey (1978) were the first to study this problem under a cardinality constraint. Note that a cardinality constraint is a special case of a budget constraint with unit costs. Nemhauser and Wolsey (1978) propose a natural greedy algorithm to solve this problem and showed that it achieves a $(1-\frac{1}{e})$-approximation ratio. Later, Feige (1998) showed that the factor of $(1-\frac{1}{e})$ is best possible for this problem, unless P = NP.

Khuller et al. (1999) use a modification of the greedy algorithm of Nemhauser and Wolsey (1978) in combination with partial enumeration to obtain a $(1-\frac{1}{e})$-approximation algorithm for the Maximum Coverage Problem with a budget constraint. Ageev and Sviridenko (2004) introduce the technique of *pipage rounding* and also derive a $(1-\frac{1}{e})$-approximation algorithm for this problem. Sviridenko (2004) observed that the algorithm of Khuller et al. (1999) can be applied to submodular functions and achieves a $(1-\frac{1}{e})$-approximation ratio. Badanidiyuru and Vondrák (2014) use another approach for maximizing a submodular function given a knapsack constraint. They derive a $(1-\frac{1}{e}-\varepsilon)$-approximation algorithm whose running time decreases in $\varepsilon > 0$. Ene and Nguyen (2019) show that there are some technical issues with the approach of Badanidiyuru and Vondrák (2014), but they build upon their idea to derive a faster algorithm with the same approximation ratio. For non-monotone submodular functions, a $(1-\frac{1}{e}-\varepsilon)$-approximation algorithm is given by Kulik et al. (2013).

Further, the problem of maximizing a submodular function subject to multiple knapsack constraints has also been studied (see Chekuri et al. (2014), Kulik et al. (2013), and Lee et al. (2010)): Kulik et al. (2013) obtain a randomized $(1-\varepsilon)(1-\frac{1}{e})$-approximation algorithm (for any $\varepsilon > 0$) for the Maximum Coverage Problem with a d-dimensional knapsack constraint. The technique also extends to (monotone) submodular maximization with a d-dimensional knapsack constraint. Lee et al. (2010) give a $(\frac{1}{5}-\varepsilon)$-approximation algorithm for non-monotone submodular maximization with a d-dimensional knapsack. The randomized rounding technique that is used in Lee et al. (2010) is similar to that of Kulik et al. (2013), but the algorithm to solve the fractional relaxation is different. However, in all these three works the interpretation of multiple knapsack constraints is different from the one we use here. In particular, in their setting the costs of the sets and the capacity of a single knapsack are d-dimensional and a feasible assignment needs to satisfy the capacity in every dimension. On the other hand, in our definition there are multiple knapsacks, but their capacity is only one-dimensional. Hence, the techniques in Chekuri

et al. (2014), Kulik et al. (2013), and Lee et al. (2010) do not apply to our problem.

Parallel to our work, two papers have appeared that also discuss the MCPK, namely Fairstein et al. (2020) and Sun et al. (2020). Fairstein et al. (2020), present a randomized $\left(1-\frac{1}{e}-\varepsilon\right)$-approximation algorithm for the MCPK in which a greedy approach for submodular maximization is combined with a partitioning of knapsacks in groups of approximately the same size. In Sun et al. (2020), a deterministic greedy $\left(1-\frac{1}{e}-\varepsilon\right)$-approximation algorithm is given for the case in which all knapsacks have the same size. For the general case, their deterministic algorithm has an approximation ratio of $\frac{1}{2}-\varepsilon$, and they present a randomized $\left(1-\frac{1}{e}-\varepsilon\right)$-approximation algorithm.

Chekuri et al. (2014) show that the approximation ratio for non-monotone submodular maximization can be improved to 0.309 if the dimension of the knapsack is assumed to be a constant. They use the technique of Vondrák (2013) to solve the relaxation of the problem and then apply a *contention resolution scheme* to round the solution. The contention resolution scheme and the method of Badanidiyuru and Vondrák (2014) can also be applied to polytopes that are an intersection of knapsack and matroid constraints. Bansal et al. (2012) consider a different type of packing constraint. They assume that each set is in at most k packing constraints. They propose a randomized algorithm with an approximation ratio of $e-1/(ke^2+(e-1)o(k))$ for monotone submodular functions.

Another problem that is related to our MCPC is the Cost-Restricted Maximum Coverage Problem with Group Budget Constraints (CMCG) introduced by Chekuri and Kumar (2004). In this problem, there are predefined groups that consist of a union of sets. Each group has a budget that must not be exceeded by its assigned sets, and there is a single knapsack constraint for all selected sets. The authors give a greedy algorithm that achieves an approximation ratio of $\frac{1}{12}$. Besides this cost-restricted version, Chekuri and Kumar (2004) also study the cardinality-restricted version for the number of sets assigned to a group. For this problem, they obtain a $\frac{1}{\alpha+1}$-approximation algorithm that uses an oracle that, given a current solution, returns a set which contribution to the current solution is within a factor α of the optimal contribution of a single set. For the cost-restricted version, Farbstein and Levin (2017) obtain a $\frac{1}{5}$-approximation. Guo et al. (2019) present a pseudo-polynomial algorithm for CMCG whose approximation ratio is $1-\frac{1}{e}$.

The MCPC with a single cluster is a special case CMCG, in which each group corresponds to a knapsack and each set has a copy for each feasible knapsack. If the solution for a single cluster is seen as a new set, then the MCPC with multiple clusters can be seen as the cardinality restricted version

in which each cluster corresponds to a group to which at most one set can be assigned. Combining the approaches of Farbstein and Levin (2017) and Chekuri and Kumar (2004) for the cost-restricted and cardinalty-restricted version, respectively, we obtain a $\frac{1}{6}$-approximation algorithm. In this chapter, we improve this ratio to $\frac{1}{3}\left(1-\frac{1}{e}\right)$.

Finally, we elaborate on the relationship between MKPC and closely related problems. We focus on three knapsack problems that look similar to MKPC but are still slightly different. Nip and Wang (2018) consider the *Two-Phase Knapsack Problem* (2-PKP) and obtain a $\frac{1}{4}$-approximation algorithm for this problem. In 2-PKP, items are assigned to knapsacks and the full knapsack needs to be packed in a cluster. Note that MKPC and 2-PKP are different because in MPKC only the costs assigned to a knapsack are restricted by the cluster capacity, whereas in 2-PKP each knapsack contributes with its maximum budget to the cluster.

Dudzinski and Walukiewicz (1987) study the *Nested Knapsack Problem* (NKP), where there are multiple subsets of items and each subset has a capacity. That is, in NKP there are no predefined knapsacks to which the items can be assigned, but from each set we have to select the most profitable items. If these sets are disjoint, the problem is called the *Decomposed Knapsack Problem*. The authors present an exact branch-and-bound method, based on the Lagrangean relaxation.

Xavier and Miyazawa (2006) consider the *Class Constraint Shelf Knapsack Problem* (CCSKP). Here, there is one knapsack with a certain capacity and the items assigned to the knapsack should also be assigned to a shelf. The shelf has a maximum capacity, and between every two shelves a divisor needs to be placed. In the CCSKP, each item belongs to a class and each shelf must only have items of the same class. The authors derive a PTAS for CCSKP in Xavier and Miyazawa (2006).

8.3 Preliminaries

In this section, we formally define the MCPC and with that also its generalizations, the MCPK and MKPC. To start, we introduce some notation that is also summarized in Table 8.1. The MCPC is defined as follows. We are given a collection of m subsets $\mathcal{S} = \{S_1, \ldots, S_m\}$ over a ground set of items $\mathcal{I} = 1, 2, \ldots, n$. Each subset $S_j \subseteq \mathcal{I}$, $j = 1, 2, \ldots, m$, is associated with a cost $c_j > 0$ and each item $i \in \mathcal{I}$ has a profit $p_i > 0$. For notational simplicity, we identify $\mathcal{S} = \{1, 2, \ldots, m\}$, and for every subset $\mathcal{S}' \subseteq \mathcal{S}$, define $\cup\mathcal{S}' = \cup_{j\in\mathcal{S}'} S_j \subseteq \mathcal{I}$ as the set of items covered by $\mathcal{S}'$. Further, we use $\mathcal{S}(i) \subseteq \mathcal{S}$ to refer to the subsets in $\mathcal{S}$ that contain item $i \in \mathcal{I}$, i.e., $\mathcal{S}(i) = \{j \in \mathcal{S} \mid i \in S_j\}$.

In addition, we are given a set of knapsacks $\mathcal{K} = \{1, 2, \ldots, p\}$ and each

$\mathcal{I}$	Collection of items
$\mathcal{S}$	Collection of subsets
$\mathcal{K}$	Collection of knapsacks
$\mathcal{C}$	Collection of clusters
$\mathcal{S}(i)$	Collection of subsets containing item i
$\mathcal{K}(l)$	Collection of knapsacks contained in cluster l
$\mathcal{C}(k)$	Collection of clusters containing knapsack k
σ	Assignment of subsets to knapsacks
$\mathcal{S}_\sigma$	Collection of assigned subsets under σ
$\mathcal{S}_\sigma(k)$	Collection of subsets assigned to knapsack k under σ
n	Number of items
m	Number of subsets
p	Number of knapsacks
q	Number of clusters
$q(l)$	Number of knapsacks in cluster l
p_i	Value of item i
c_j	Cost of subset j
B_k	Budget of knapsack k
U_l	Capacity of cluster l

Table 8.1 Notation used in Chapter 8.

knapsack $k \in \mathcal{K}$ has a capacity $B_k > 0$. These knapsacks are partitioned into a set of q clusters $\mathcal{C} = \{1, 2, \ldots, q\}$ and each cluster $l \in \mathcal{C}$ has a separate capacity $U_l > 0$. We denote by $\mathcal{K}(l) \subseteq \mathcal{K}$ the subset of knapsacks that are contained in cluster l. The number of knapsacks in the set $\mathcal{K}(l)$ is given by $q(l) := |\mathcal{K}(l)|$. Also, we use $\mathcal{C}(k) \in \mathcal{C}$ to refer to the cluster containing knapsack $k \in \mathcal{K}$.

Our goal is to determine a feasible assignment $\sigma : \mathcal{S} \to \mathcal{K} \cup \{0\}$ of subsets to knapsacks such that the total profit of all covered items is maximized. Each subset $j \in \mathcal{S}$ can be assigned to at most one knapsack in $\mathcal{K}$ and we define $\sigma(j) = 0$ if j remains unassigned. We say that an assignment σ is *feasible* if (i) for every knapsack $k \in \mathcal{K}$, the total cost of all subsets assigned to k is at most B_k, and (ii) for every cluster $l \in \mathcal{C}$, the total cost of all subsets assigned to the knapsacks in $\mathcal{K}(l)$ of cluster l is at most U_l. Given an assignment σ, let $\mathcal{S}_\sigma(k) \subseteq \mathcal{S}$ be the set of subsets assigned to knapsack $k \in \mathcal{K}$ under σ, and let $\mathcal{S}_\sigma = \cup_{k \in \mathcal{K}} \mathcal{S}_\sigma(k) \subseteq \mathcal{S}$ be the set of all assigned subsets. Using this notation, the MCPC problem is formally defined as follows:

$$\max_{\sigma} \quad \sum_{i \in \cup \mathcal{S}_\sigma} p_i$$

subject to:

$$\sum_{j \in \mathcal{S}_\sigma(k)} c_j \leq B_k \qquad \forall k \in \mathcal{K}$$

$$\sum_{k \in \mathcal{K}(l)} \sum_{j \in \mathcal{S}_\sigma(k)} c_j \leq U_l \qquad \forall l \in \mathcal{C}.$$

For the MCPC, we make the following two assumptions.

Assumption 8.1. *For every cluster $l \in \mathcal{C}$ and every knapsack $k \in \mathcal{K}(l)$, it holds that $B_k \leq U_l$.*

Assumption 8.2. *For every cluster $l \in \mathcal{C}$, it holds that $\sum_{k \in \mathcal{K}(l)} B_k > U_l$.*

Note that these assumptions are without loss of generality. It is impossible to assign more than U_l to any knapsack in $\mathcal{K}(l)$, so if Assumption 8.1 is violated by some $k \in \mathcal{K}(l)$, then we can simply redefine $B_k = U_l$. If Assumption 8.2 is not satisfied, then the capacity of cluster l is redundant. In that case, the sum of all budget constraints of the individual knapsacks is more restrictive than the cluster constraint.

The MCPK is the special case of MCPC in which all cluster capacities are redundant. Another special case of MCPK is the classical MKP which we obtain if $\mathcal{S} = \{1, 2, \ldots, n\}$ and $S_i = \{i\}$ for all $i \in \mathcal{S}$. We refer to the generalization of MKP with non-redundant cluster capacities as the MKPC.

8.4 Maximum coverage problem with knapsack constraints

We start by deriving an approximation algorithm for the Maximum Coverage Problem with Knapsack Constraints. A natural ILP is given below.

$$\max \quad L(x) = \sum_{i \in \mathcal{I}} p_i y_i \tag{8.1}$$

subject to

$$\sum_{j \in \mathcal{S}} c_j x_{jk} \leq B_k \qquad \forall k \in \mathcal{K} \tag{8.2}$$

$$\sum_{k \in \mathcal{K}} x_{jk} \leq 1 \qquad \forall j \in \mathcal{S} \tag{8.3}$$

$$\sum_{j \in \mathcal{S}(i)} \sum_{k \in \mathcal{K}} x_{jk} \geq y_i \qquad \forall i \in \mathcal{I} \tag{8.4}$$

$$x_{jk} \in \{0, 1\} \qquad \forall j \in \mathcal{S} \qquad \forall k \in \mathcal{K} \tag{8.5}$$

$$y_i \in \{0, 1\} \qquad \forall i \in \mathcal{I}. \tag{8.6}$$

The decision variable x_{jk} indicates whether set $j \in \mathcal{S}$ is assigned to knapsack $k \in \mathcal{K}$. In addition, the decision variable y_i indicates whether item $i \in \mathcal{I}$ is covered. Constraint (8.2) ensures that the total cost assigned to each knapsack is at most its budget and constraint (8.3) makes sure that each set is assigned to at most one knapsack. Constraint (8.4) enforces that an item can only be covered

if at least one of the sets containing it is assigned to a knapsack. We refer to above ILP (8.1)–(8.6) as (IP) and to its corresponding linear programming relaxation as (LP). Furhtermore, let OPT refer to the objective function value of an optimal solution to (IP). It is important to realize that for any feasible fixing of the x_{jk}-variables, the optimal y_i-variables are easily determined by setting $y_i = \min\{1, \sum_{j \in \mathcal{S}(i)} \sum_{k \in \mathcal{K}} x_{jk}\}$. As a consequence, an optimal solution (x, y) of the above program is fully specified by the corresponding x-part.

A subtle but important point is that a set $j \in \mathcal{S}$ cannot be assigned to a knapsack $k \in \mathcal{K}$ whenever $c_j > B_k$. While these restrictions are taken care of implicitly in the ILP formulation (IP), they are not in the LP relaxation (LP). In fact, we would have to add these restrictions explicitly by defining variables x_{jk} only for all $j \in \mathcal{S}$ and $k \in \mathcal{K}(j)$, where $\mathcal{K}(j) \subseteq \mathcal{K}$ is the set of knapsacks whose capacity is at least c_j, and adapt the constraints accordingly. However, these adaptations are straightforward. For notational convenience, we do not state them explicitly. In the remainder, whenever we refer to (LP), our understanding is that we refer to the corresponding LP relaxation with these assignment restrictions incorporated. No confusion shall arise.

In the approximation algorithm for the MCPC and its generalizations, we crucially exploit the fact that we can find a solution to the LP relaxation, which satisfies the *bounded split property*. The bounded split property is defined in Definition 8.2, but we need to define some other concepts first. Let x be a fractional solution of (LP). We define the *support graph* H_x of x as the bipartite graph $H_x := (\mathcal{S} \cup \mathcal{K}, E_x)$ with node sets $\mathcal{S}$ and $\mathcal{K}$ on each side of the bipartition, respectively, and the edge set E_x contains all edges $\{j, k\}$ for which x_{jk} is non-integral, i.e., $E_x = \{\{j, k\} \in \mathcal{S} \times \mathcal{K} \mid 0 < x_{jk} < 1\}$. Let $M \subseteq E_x$ be a *matching* of the support graph H_x. A matching $M \subseteq E_x$ is a subset of the edges such that no two edges in M have a node in common. Matching M *saturates* all nodes in $\mathcal{S}$ if for every (non-isolated) node $j \in \mathcal{S}$ there is a matching edge $\{j, k\} \in M$. In that case, we say that M is an *$\mathcal{S}$-saturating matching* of H_x. The concept of the $\mathcal{S}$-saturating matching is crucial in the definition for the bounded split property below.

Definition 8.2 (Bounded split property)**.** *A feasible solution x of (LP) satisfies the* bounded split property *if the support graph H_x of x has an $\mathcal{S}$-saturating matching.*

For MKP, it is known that there exists an optimal solution x^* of the LP relaxation for which the bounded split property holds (see, e.g., Chekuri and Khanna (2005) and Shmoys and Tardos (1993)). Deriving a $\frac{1}{2}$-approximation algorithm for the problem is then easy. The idea behind this algorithm is to decompose x^* into an integral and a fractional part. The integral part naturally corresponds to a feasible integral assignment. Exploiting the bounded

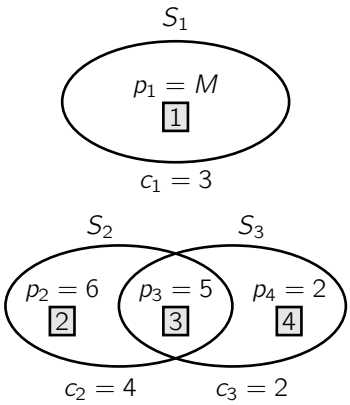


Figure 8.1 Illustration of the instance for which the (unique) optimal solution does not satisfy the bounded split property if $B_1 = 6$.

split property, the corresponding $\mathcal{S}$-saturating matching gives rise to a feasible integral assignment for the fractional part, namely assigning every fractional assigned set to the knapsack to which it is matched in the $\mathcal{S}$-saturating matching. Thus, by taking the better of the two assignments, we recover at least half of the optimal LP solution.

Unfortunately, for the more general MCPK problem the optimal solution of (LP) does not necessarily satisfy the bounded split property. To see that consider the following example, illustrated in Figure 8.1, with a collection $\mathcal{S} = \{S_1, S_2, S_3\}$ over the item set $\mathcal{I} = \{1, 2, 3, 4\}$, where $S_1 = \{1\}$, $S_2 = \{2, 3\}$ and $S_3 = \{3, 4\}$ with corresponding costs $c_1 = 3$, $c_2 = 4$, and $c_3 = 2$. There is a single knapsack of capacity $B_1 = 6$, and the profits of the items are $p_1 = M$ (some large number), $p_2 = 6$, $p_3 = 5$, and $p_4 = 2$. The unique optimal solution to (LP) is $x^*_{\text{LP}} = \left(1, \frac{1}{2}, \frac{1}{2}\right)$, meaning that there are two sets fractionally assigned to the knapsack. Thus, the bounded split property does not hold for this solution.

Instead, below we show that a solution to the LP relaxation of MCPK exists that always satisfies the bounded split property and is only a factor $1 - \frac{1}{e}$ away from the optimal solution. In order to do so, we use the technique of *pipage rounding* (Ageev and Sviridenko, 2004). To this aim, we define a new program (CP) as follows:

$$\max \quad F(x) = \sum_{i \in \mathcal{I}} p_i \Big(1 - \prod_{j \in \mathcal{S}(i)} \Big(1 - \sum_{k \in \mathcal{K}} x_{jk}\Big)\Big) \tag{8.7}$$

subject to

$$\sum_{j \in \mathcal{S}} c_j x_{jk} \leq B_k \qquad \forall k \in \mathcal{K} \tag{8.8}$$

$$\sum_{k \in \mathcal{K}} x_{jk} \leq 1 \qquad \forall j \in \mathcal{S} \tag{8.9}$$

$$x_{jk} \in [0,1] \qquad \forall j \in \mathcal{S} \qquad \forall k \in \mathcal{K}. \tag{8.10}$$

It is easy to see that a feasible solution x for the problem (LP) is also a feasible solution for (CP). In addition, as is usually the case in pipage formulations, the objective function values of (LP) and (CP) are the same for every integral solution, i.e., $L(x) = F(x)$ for every integer solution x. Our formulation (CP) is even slightly stronger than standard pipage formulations in the sense that $L(x) = F(x)$ if for all $j \in \mathcal{S}$ it holds that $\sum_{k \in \mathcal{K}} x_{jk} \in \{0,1\}$. Moreover, for fractional solutions x the value $F(x)$ is lower bounded by $L(x)$ as we show in the next lemma.

Lemma 8.1. *For every feasible solution x of (LP), we have that $F(x) \geq \left(1 - \frac{1}{e}\right) L(x)$.*

Proof. The proof of this lemma is similar to the one given in Ageev and Sviridenko (2004) for the maximum coverage problem. Let x be a feasible solution for (LP). Fix an item $i \in \mathcal{I}$ and let $s = |\mathcal{S}(i)|$ be the number of sets containing i. We obtain

$$\begin{aligned}
1 - \prod_{j \in \mathcal{S}(i)} \left(1 - \sum_{k \in \mathcal{K}} x_{jk}\right) &\geq 1 - \left(\frac{1}{s} \sum_{j \in \mathcal{S}(i)} \left(1 - \sum_{k \in \mathcal{K}} x_{jk}\right)\right)^s \\
&\geq 1 - \left(1 - \frac{1}{s} \sum_{j \in \mathcal{S}(i)} \sum_{k \in \mathcal{K}} x_{jk}\right)^s \\
&\geq 1 - \left(1 - \frac{1}{s} \min\left\{1, \sum_{j \in \mathcal{S}(i)} \sum_{k \in \mathcal{K}} x_{jk}\right\}\right)^s \\
&\geq \left(1 - \left(1 - \frac{1}{s}\right)^s\right) \min\left\{1, \sum_{j \in \mathcal{S}(i)} \sum_{k \in \mathcal{K}} x_{jk}\right\} \\
&\geq \left(1 - \frac{1}{e}\right) \min\left\{1, \sum_{j \in \mathcal{S}(i)} \sum_{k \in \mathcal{K}} x_{jk}\right\} \\
&\geq \left(1 - \frac{1}{e}\right) y_i.
\end{aligned}$$

The first inequality follows from the arithmetic/geometric mean inequality (see, e.g., Goemans and Williamson (1994)) which states that for any n non-

negative numbers $a_1, a_2, \ldots, a_n$, we have $\prod_{i=1}^{n} a_i \leq (\frac{1}{n}\sum_{i=1}^{n} a_i)^n$. In the second inequality, we take the 1 out of the summation, and the third inequality holds because $\min\{1, \sum_{j\in\mathcal{S}(i)}\sum_{k\in\mathcal{K}} x_{jk}\}$ is always less than or equal to $\sum_{j\in\mathcal{S}(i)}\sum_{k\in\mathcal{K}} x_{jk}$. The fourth inequality follows from the concavity of the function in $\min\{1, \sum_{j\in\mathcal{S}(i)}\sum_{k\in\mathcal{K}} x_{jk}\}$ on the interval between 0 and 1, and thus its minimum is attained at those endpoints. Finally, the last inequality follows from the fact that for every $n \geq 1$ it holds that $\left(1-\frac{1}{n}\right)^n \leq \frac{1}{e}$.

Using the above, we can conclude that

$$\begin{aligned} F(x) &= \sum_{i\in\mathcal{I}} p_i \left(1 - \prod_{j\in\mathcal{S}(i)} \left(1 - \sum_{k\in\mathcal{K}} x_{jk}\right)\right) \geq \sum_{i\in\mathcal{I}} p_i \left(1-\frac{1}{e}\right) y_i \\ &= \left(1-\frac{1}{e}\right) L(x), \end{aligned}$$

which proves the lemma. □

In Theorem 8.1 below, we show that we can transform an optimal LP solution x^*_{LP} into a solution x that satisfies the bounded split property without decreasing the objective value of (CP).

Theorem 8.1. *There exists a feasible solution x of (LP) that satisfies the bounded split property and for which $F(x) \geq F(x^*_{LP})$.*

Before we proceed with the proof of Theorem 8.1, we introduce some more terminology. Let x be a feasible solution to (LP) and let $H_x = (\mathcal{S} \cup \mathcal{K}, E_x)$ be the support graph of x. Consider a maximal path $P = \langle u_1, \ldots, u_t\rangle$ in H_x that starts and ends with a node of degree one. We call P an *$\mathcal{S}$-$\mathcal{S}$-path* if $u_1, u_t \in \mathcal{S}$, an *$\mathcal{S}$-$\mathcal{K}$-path* if $u_1 \in \mathcal{S}$ and $u_t \in \mathcal{K}$, and a *$\mathcal{K}$-$\mathcal{K}$-path* if $u_1, u_t \in \mathcal{K}$.

We use Lemmas 8.2, 8.3, and 8.4 to prove Theorem 8.1. An outline of the proof of Theorem 8.1 is as follows. First, we show in Lemma 8.2 that there exists an optimal solution $x^* = x^*_{\mathrm{LP}}$ for (LP) such that the support graph H_{x^*} is acyclic. Second, we prove in Lemma 8.3 that from x^* we can derive a solution x' whose support graph $H_{x'}$ does not contain any $\mathcal{S}$-$\mathcal{S}$-paths such that the objective function value of (CP) does not decrease. Finally, we show in Lemma 8.4 that this solution x' satisfies the bounded split property. Combining these lemmas proves Theorem 8.1.

Lemma 8.2. *There is an optimal solution $x^* = x^*_{LP}$ of (LP) whose support graph H_{x^*} is acyclic.*

Proof. Let x be an optimal solution for (LP) and suppose the support graph $H_x = (\mathcal{S} \cup \mathcal{K}, E_x)$ of x contains a cycle that visits nodes $u_1, \ldots, u_t, u_1$, i.e.,

$C = \langle u_1, \ldots, u_t, u_1 \rangle$. Note that the length, given in the number of edges, of C is even because H_x is bipartite. We can, therefore, decompose C into two matchings M_1 and M_2 with $|M_1| = |M_2|$. Define

$$\varepsilon := \min \Big\{ \min_{\{j,k\} \in M_1} \{c_j x_{jk}\}, \min_{\{j,k\} \in M_2} \{c_j (1 - x_{jk})\} \Big\}.$$

We call each edge on C for which the minimum is attained a *critical edge*; note that by definition there is at least one critical edge. We use ε to define a new solution $x(\varepsilon)$ as follows:

$$x_{jk}(\varepsilon) = \begin{cases} x_{jk} + \frac{\varepsilon}{c_j} & \text{if } \{j,k\} \in M_1 \\ x_{jk} - \frac{\varepsilon}{c_j} & \text{if } \{j,k\} \in M_2 \\ x_{jk} & \text{otherwise.} \end{cases} \tag{8.11}$$

In the way ε is defined, the value of each critical edge $\hat{e}$ on C with respect to $x(\varepsilon)$ is integral, and more specifically, $x_{\hat{e}}(\varepsilon) = 0$ if $\hat{e} \in M_1$ and $x_{\hat{e}}(\varepsilon) = 1$ if $\hat{e} \in M_2$. Thus, $\hat{e}$ is not part of the support graph $H_{x(\varepsilon)}$. As a consequence, $H_{x(\varepsilon)}$ is a subgraph of H_x that has at least one cycle less.

It remains to show that $x(\varepsilon)$ is a feasible solution to (LP) and has the same objective function value as x. Every node $u = u_i$, $i = 1, 2, \ldots, t$, on the cycle C has two incident edges in C, one belonging to M_1 and one to M_2. We distinguish two cases:

Case 1: $u = k \in \mathcal{K}$. Let the two incident edges be $\{i,k\} \in M_1$ and $\{j,k\} \in M_2$. The combined cost of these two edges in $x(\varepsilon)$ is:

$$c_i x_{ik}(\varepsilon) + c_j x_{jk}(\varepsilon) = c_i \left(x_{ik} + \frac{\varepsilon}{c_i} \right) + c_j \left(x_{jk} - \frac{\varepsilon}{c_j} \right) = c_i x_{ik} + c_j x_{jk}.$$

That is, the total cost assigned to knapsack k in $x(\varepsilon)$ is the same as in x. As x satisfies constraint (8.2), we conclude that $x(\varepsilon)$ also does that.

Case 2: $u = j \in \mathcal{S}$. Let the two incident edges be $\{j,k\} \in M_1$ and $\{j,l\} \in M_2$. By the definition of $x(\varepsilon)$, we obtain

$$x_{jk}(\varepsilon) + x_{jl}(\varepsilon) = x_{jk} + \frac{\varepsilon}{c_j} + x_{jl} - \frac{\varepsilon}{c_j} = x_{jk} + x_{jl}.$$

In particular, this implies that $x(\varepsilon)$ satisfies constraints (8.3) and (8.4) because x does. Furthermore, from the equation above, it also follows that the solutions $x(\varepsilon)$ and x result in the same objective function value of (LP), i.e., $L(x(\varepsilon)) = L(x)$.

By repeating the above procedure, we eventually obtain a feasible solution x^* of (LP) such that H_{x^*} is a subgraph of H_x that does not contain any cycles and $L(x^*) = L(x)$, i.e., x^* is an optimal solution. □

From Lemma 8.2, we can conclude that there exists an optimal solution for (LP) which support graph does not contain any cycles. In Lemma 8.3, we remove all $\mathcal{S}$-$\mathcal{S}$-paths form the support graph. After that transformation, the solution is not longer the optimal solution for (LP), but the objective function for (P) has not decreased.

Lemma 8.3. *There exists a feasible solution x' of (LP) whose support graph $H_{x'}$ is acyclic and does not contain any $\mathcal{S}$-$\mathcal{S}$-paths, and which satisfies $F(x') \geq F(x^*_{LP})$.*

Proof. Let x be an optimal solution for (LP) whose support graph H_x is acyclic; we know that such a solution exists by Lemma 8.2. Suppose H_x contains an $\mathcal{S}$-$\mathcal{S}$-path $P = \langle u_1, \ldots, u_t \rangle$; recall that nodes u_1 and u_t have degree one in H_x. The path P has even length because H_x is bipartite. We can thus decompose P into two matchings M_1 and M_2 with $|M_1| = |M_2|$. We define $x(\varepsilon)$ as in (8.11) for every $\varepsilon \in [-\varepsilon_1, \varepsilon_2]$, where

$$\varepsilon_1 := \min\left\{ \min_{\{j,k\} \in M_1} \{c_j x_{jk}\}, \min_{\{j,k\} \in M_2} \{c_j (1 - x_{jk})\} \right\}$$
$$\varepsilon_2 := \min\left\{ \min_{\{j,k\} \in M_1} \{c_j (1 - x_{jk})\}, \min_{\{j,k\} \in M_2} \{c_j x_{jk}\} \right\}.$$

We first show that $x(\varepsilon)$ is a feasible solution for (CP) for every $\varepsilon \in [-\varepsilon_1, \varepsilon_2]$. By following the same line of arguments as in the proof of Lemma 8.2, we can show that the solution $x(\varepsilon)$ satisfies constraint (8.8). Further, for every $j \in \mathcal{S}$, $j \neq u_1, u_t$, we can also show, as in the proof of Lemma 8.2, that $\sum_{k \in \mathcal{K}} x_{jk}(\varepsilon) = \sum_{k \in \mathcal{K}} x_{jk}$ and thus constraints (8.9) and (8.10) are satisfied in $x(\varepsilon)$. By definition, the endpoints $j = u_1, u_t$ of P have degree one in H_x and thus j is fractionally assigned to a single knapsack in x. Consider $j = u_1$ and let $k = u_2$ be the knapsack to which j is assigned in x. We have $x_{jk} \in (0, 1)$ and, from the definition of ε_1 and ε_2, it follows that $x_{jk}(\varepsilon) \in [0, 1]$. The same argument holds for $j = u_t$. Thus, $x(\varepsilon)$ also satisfies constraints (8.9) and (8.10) for $j = u_1, u_t$ if $\varepsilon \in [-\varepsilon_1, \varepsilon_2]$. We conclude that $x(\varepsilon)$ is a feasible solution for (CP) if $\varepsilon \in [-\varepsilon_1, \varepsilon_2]$.

Next, we show that $F(x(-\varepsilon_1))$ or $F(x(\varepsilon_2))$ is at least as large as $F(x)$. To this aim, we show that $F(x(\varepsilon))$ as a function of ε is convex. In fact, we show convexity for each item $i \in \mathcal{I}$ separately. Observe that the first and the last edge of P are in different matchings. Without loss of generality, we assume that $\{u_1, u_2\} \in M_1$ and $\{u_{t-1}, u_t\} \in M_2$. The contribution of i to the objective function $F(x(\varepsilon))$ can be written as $p_i f_i(\varepsilon)$, where

$$f_i(\varepsilon) = \left(1 - \prod_{j \in \mathcal{S}(i)} \left(1 - \sum_{k \in \mathcal{K}} x_{jk}(\varepsilon)\right)\right)$$

$$= \left(1 - \prod_{j \in \mathcal{S}(i) \cap \{u_1, u_t\}} \left(1 - \sum_{k \in \mathcal{K}} x_{jk}(\varepsilon)\right) \prod_{j \in \mathcal{S}(i) \setminus \{u_1, u_t\}} \left(1 - \sum_{k \in \mathcal{K}} x_{jk}\right)\right).$$

Here, we adopt the convention that the empty product is defined to be 1. Note that the latter product $\prod_{j \in \mathcal{S}(i) \setminus \{u_1, u_t\}} (1 - \sum_{k \in \mathcal{K}} x_{jk})$ is independent of ε and has a value between 0 and 1. We can now distinguish three cases:

- $|\{u_1, u_t\} \cap \mathcal{S}(i)| = 2$. In this case, we have

$$\prod_{j \in \mathcal{S}(i) \cap \{u_1, u_t\}} \left(1 - \sum_{k \in \mathcal{K}} x_{jk}(\varepsilon)\right) = \left(1 - \left(x_{u_1 u_2} + \frac{\varepsilon}{c_{u_1}}\right)\right)\left(1 - \left(x_{u_t u_{t-1}} - \frac{\varepsilon}{c_{u_t}}\right)\right).$$

  That is, $f_i(\varepsilon)$ is a quadratic function of ε and the coefficient of the quadratic term is $1/(c_{u_1} c_{u_t})$, which is positive.

- $|\{u_1, u_t\} \cap \mathcal{S}(i)| = 1$. In this case, $f_i(\varepsilon)$ is a linear function in ε.

- $|\{u_1, u_t\} \cap \mathcal{S}(i)| = 0$. In this case, $f_i(\varepsilon)$ is independent of ε.

Concluding, the function $f_i(\varepsilon)$ is a quadratic function in ε with a positive quadratic term. Hence, $f_i(\varepsilon)$ is convex in ε and so is $F(x(\varepsilon))$. Therefore, the maximum of $F(x(\varepsilon))$ over $[-\varepsilon_1, \varepsilon_2]$ is attained at one of the endpoints:

$$\max\{F(x(-\varepsilon_1)), F(x(\varepsilon_2))\} = \max_{\varepsilon \in [-\varepsilon_1, \varepsilon_2]} \{F(x(\varepsilon))\} \geq F(x(0)) = F(x).$$

As a result, we can find a feasible solution $x' \in \{x(-\varepsilon_1), x(\varepsilon_2)\}$ with the property that $F(x') \geq F(x)$. Further, x' has at least one fractional variable on P less than x. Thus, $H_{x'}$ is a subgraph of H_x with at least one edge of P removed. By repeating this procedure, we eventually obtain a feasible solution x' of (CP) whose support graph does not contain any $\mathcal{S}$-$\mathcal{S}$-paths, and for which $F(x') \geq F(x)$. □

In the previous two lemmas, we have constructed a solution x' that is feasible for (LP), which support graph does not contain any cycles and $\mathcal{S}$-$\mathcal{S}$-paths, and for which holds that $F(x') \geq F(x^*_{LP})$. In Lemma 8.4, we show that this solution x' satisfies the bounded split property.

Lemma 8.4. *Let x' be a feasible solution of (LP) whose support graph $H_{x'}$ is acyclic and does not contain any $\mathcal{S}$-$\mathcal{S}$-paths. Then x' satisfies the bounded split property.*

Proof. In order to prove that the solution x' satisfies the bounded split property, we have to show that there is a matching M in $H_{x'}$ that saturates all (non-isolated) nodes in $\mathcal{S}$. For simplicity, we can assume in this proof that all isolated

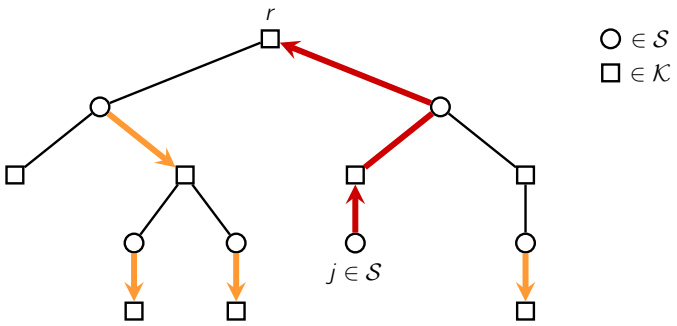


Figure 8.2 Illustration of the matching procedure described in the proof of Lemma 8.4.

nodes are removed from $H_{x'}$. Note that we assume that $H_{x'}$ is acyclic, and thus it is a forest. The idea of this proof is to construct a matching M_T for each tree T of the forest $H_{x'}$. Our final matching M is then simply the union of all these matchings, i.e., $M = \cup_T M_T$.

Consider a tree T of $H_{x'}$ and root it at an arbitrary node $r \in \mathcal{K}$. By assumption, T has at most one leaf in $\mathcal{S}$ because otherwise there would exist an $\mathcal{S}$-$\mathcal{S}$-path in $H_{x'}$. Let us slightly abuse notation by letting T refer to both the tree and the set of nodes it spans. We now show how to construct a matching M_T that matches all the nodes in $T \cap \mathcal{S}$. This procedure is illustrated in Figure 8.2. If there is a unique $\mathcal{S}$-leaf, say $j \in \mathcal{S}$, then we match each $\mathcal{S}$-node on the path from j to the root r to its unique parent in T. In Figure 8.2, this matching is indicated with red arrows. Each remaining $\mathcal{S}$-node in T is matched to one of its children, which is illustrated with orange arrows in Figure 8.2. This matching is done arbitrarily, which is possible because there is always at least one child as j is the only $\mathcal{S}$-leaf in T. Note that in this procedure, all nodes in $T \cap \mathcal{S}$ are matched in matching M_T. Hence, we have created an $\mathcal{S}$-saturating matching. □

Proof of Theorem 8.1. Using Lemmas 8.2, 8.3, and 8.4, we have shown how to construct a feasible solution x that satisfies the bounded split property and for which $F(x) \geq F(x^*_{LP})$. Hence, the claim from Theorem 8.1 follows immediately from this procedure. □

In Theorem 8.1, we have shown that there exists a feasible solution for (LP) that satisfies the bounded split property. In Algorithm 8.1, we use that fractional solution to construct two feasible solutions for the integral problem. The first solution is equal to all integral assignments in the solution satisfying the bounded split property. In the second solution, all fractional sets are integrally

Algorithm 8.1: $\frac{1}{2}(1-\frac{1}{e})$-approximation algorithm for MCPK.

Compute an optimal solution x^* to (LP).
Derive a solution x from x^* satisfying the bounded split property using the procedures described in the proofs of Lemmas 8.2, 8.3, and 8.4.
Let M be the corresponding $\mathcal{S}$-saturating matching.
Decompose the fractional solution x into x^1, x^2 as follows:

$$x^1_{jk} = \begin{cases} x^1_{jk} = 1 & \text{if } x_{jk} = 1 \\ x^1_{jk} = 0 & \text{otherwise} \end{cases}$$

and

$$x^2_{jk} = \begin{cases} x^2_{jk} = 1 & \text{if } x_{jk} \in (0,1),\ \{j,k\} \in M \\ x^2_{jk} = 0 & \text{otherwise.} \end{cases}$$

Output: $x_{\text{alg}} \in \arg\max\{L(x^1), L(x^2)\}$

assigned to the knapsack to which they are matched. In Theorem 8.2, we show that Algorithm 8.1 is a $\frac{1}{2}\left(1-\frac{1}{e}\right)$-approximation algorithm for MCPK.

Theorem 8.2. *Algorithm 8.1 is a $\frac{1}{2}\left(1-\frac{1}{e}\right)$-approximation algorithm for MCPK.*

Proof. Note that all the procedures described in the proof of Theorem 8.1 to derive a solution x from x^* can be implemented to take polynomial time. The running time of Algorithm 8.1 is thus polynomial.

The solution x^1 constructed by the algorithm corresponds to the integral part of the solution x satisfying the bounded split property. Clearly, this is a feasible solution to MCPK. Further, the solution x^2 is derived from the fractional part of x using the $\mathcal{S}$-saturating matching M. It is important to note that in the procedures of Lemmas 8.2 and 8.3 only edges are removed from the subgraph and no edges are added. Hence, by construction, every set $j \in \mathcal{S}$ is matched to a knapsack $k \in \mathcal{K}$ for which $x^*_{LP} > 0$, i.e., $B_k \geq c_j$. Thus, x^2 is also a feasible integral solution.

It remains to bound the approximation ratio of the algorithm. We have

$$\begin{aligned} L(x_{\text{alg}}) &= \max\{L(x^1), L(x^2)\} \geq \frac{1}{2}(L(x^1) + L(x^2)) \\ &= \frac{1}{2}(L(x^1 + x^2) = \frac{1}{2}(F(x^1 + x^2)) \geq \frac{1}{2}F(x) \\ &\geq \frac{1}{2}F(x^*) \geq \frac{1}{2}\left(1-\frac{1}{e}\right)L(x^*) \geq \frac{1}{2}\left(1-\frac{1}{e}\right)\text{OPT}. \end{aligned}$$

Here, the first inequality follows from the fact that the maximum of two elements is at least half of the sum of the elements. The second equality holds because of the linearity of $L(x)$ and the third equality follows from the fact

that the objective function values of (LP) and (CP) are the same for integer solutions, as we have observed before. To see that the second inequality holds, note that by the definitions of x^1 and x^2 we have

$$\sum_{j\in\mathcal{S}}\sum_{k\in\mathcal{K}} x_{jk} \leq \sum_{j\in\mathcal{S}}\sum_{k\in\mathcal{K}}(x^1_{jk} + x^2_{jk}),$$

and the function $F(x)$ is non-decreasing in x. The third inequality follows from Lemma 8.3 and the fourth inequality holds because of Lemma 8.1. The final inequality holds because (LP) is a relaxation of the ILP (IP) of the MCPK. □

8.5 Maximum coverage problem with cluster constraints

In this section, we derive an approximation algorithm for the MCPC. Our algorithm exploits the existence of an LP-based approximation algorithm for the problem without cluster constraints. In particular, we use our algorithm for MCPK derived in the previous section as a subroutine to obtain a $\frac{1}{3}(1-\frac{1}{e})$-approximation algorithm for MCPC.

Recall that in MCPC we have to determine a feasible assignment of sets in $\mathcal{S}$ to the knapsacks in $\mathcal{K}$ such that the total profit of all covered items is maximized. But, in addition, the knapsacks are partitioned into q clusters $\mathcal{C} = [q]$ and the total cost of all sets assigned to the knapsacks $k \in \mathcal{K}(l)$ of cluster $l \in \mathcal{C}$ may not exceed the given cluster capacity U_l.

The difficulty of the MCPC lies in the fact that one cannot look at a single knapsack $k \in \mathcal{K}$ to determine whether an assignment is feasible. Instead, an assignment has to satisfy, for every cluster $l \in \mathcal{C}$, both the cluster capacity U_l and the knapsack capacities B_k for every $k \in \mathcal{K}(l)$. A first approach for an approximation algorithm for the MCPC might be to simply add the following constraint to the ILP formulation of (8.1)-(8.6):

$$\sum_{j\in\mathcal{S}}\sum_{k\in\mathcal{K}(l)} c_j x_{jk} \leq U_l \qquad \forall l \in \mathcal{C}. \tag{8.12}$$

We could solve the LP relaxation of the resulting problem and let x^*_{LP} be the optimal solution. If this problem formulation is used, Theorem 8.1 is also true. To see that, it is important to realize that in the transformation of x^*_{LP} to a solution that satisfies the bounded split property, the total costs that were assigned to a knapsack did not change. Hence, it is tempting to believe that Algorithm 8.1 also gives a $\frac{1}{2}\left(1-\frac{1}{e}\right)$-approximation algorithm for the MCPC.

However, the solution x^2 in Algorithm 8.1 might be infeasible for the MCPC, namely if $\sum_{j\in\mathcal{S}}\sum_{k\in\mathcal{K}(l)} c_j x^2_{jk} > U_l$ for any cluster $l \in \mathcal{C}$. In the worst case, every

knapsack $k \in \mathcal{K}(l)$ has a capacity B and the cluster capacity $U_l = B$. If for every knapsack $k \in \mathcal{K}(l)$ holds that $\sum_{j \in \mathcal{S}} c_j x_{jk}^2 = B$, then there does not exists a feasible solution for MCPC for which two sets $j \in \mathcal{S}$ with $x_{jk}^2 = 1$ are assigned to the knapsacks $k \in \mathcal{K}(l)$.

A key element of our approach is to integrate the cluster capacities into the ILP formulation of MCPK by introducing a variable z_{kl} for every cluster $l \in \mathcal{C}$ and every knapsack $k \in \mathcal{K}(l)$, which specifies the fraction of the cluster capacity U_l that is assigned to knapsack k. We obtain the following ILP formulation for MCPC:

$$\max \quad L(x, z) = \sum_{i \in \mathcal{I}} p_i y_i \tag{8.13}$$

subject to

$$\text{Constraints } (8.2) - (8.6) \tag{8.14}$$

$$\sum_{j \in \mathcal{S}} c_j x_{jk} \leq U_l z_{kl} \qquad \forall l \in \mathcal{C} \qquad \forall k \in \mathcal{K}(l) \tag{8.15}$$

$$\sum_{k \in \mathcal{K}(l)} z_{kl} \leq 1 \qquad \forall l \in \mathcal{C} \tag{8.16}$$

$$z_{kl} \geq 0 \qquad \forall l \in \mathcal{C} \qquad \forall k \in \mathcal{K}(l). \tag{8.17}$$

Note that constraints (8.15)–(8.16) ensure that the capacity of cluster $l \in \mathcal{C}$ is not exceeded. The advantage of this formulation is that, once we know the optimal values of the z_{kl}-variables, the remaining problem basically reduces to an instance of MCPK. However, there is a subtle difference, as is explained below. We use (IP) and (LP) to refer to the integer formulation above and its relaxation. Let OPT refer to the objective function value of an optimal solution to MCPC.

Similar to MCPK, any optimal solution (x, y, z) of the above program is fully characterized by the respective (x, z)-part: given the values of the x_{jk} and the z_{kl}-variables, we can infer the optimal values of the y_i-variables as before. As a consequence, an optimal solution (x, y, z) of the above program is fully specified by (x, z).

It will be convenient to assume that the knapsacks $\mathcal{K}(l) = \{1, \ldots, q(l)\}$ of each cluster $l \in \mathcal{C}$ are ordered by non-increasing capacities, i.e., if $k, k' \in \mathcal{K}(l)$ with $k < k'$ then $B_k \geq B_{k'}$. In every cluster, there is one *critical knapsack*, which is crucial in the remainder of the analysis. This critical knapsack is defined in Definition 8.3.

Definition 8.3 (Critical knapsack)**.** *The knapsack $\kappa(l) \in \mathcal{K}(l)$ which satisfies $\sum_{k=1}^{\kappa(l)-1} B_k \leq U_l < \sum_{k=1}^{\kappa(l)} B_k$ is called the* critical knapsack *of cluster l.*

Note that by Assumption 8.2 there always exists such a critical knapsack. The concept of the critical knapsack is used in Lemma 8.5, to show that the z-variables of an optimal LP solution admit a specific structure. Intuitively, the lemma states that the capacity U_l of each cluster l is shared maximally among the first $\kappa(l) - 1$ knapsacks in $\mathcal{K}(l)$ and the remaining capacity is assigned to the critical knapsack $\kappa(l)$.

Lemma 8.5. *There is an optimal solution (x^*, z^*) of (LP) such that for every cluster $l \in \mathcal{C}$,*

$$z^*_{kl} = \begin{cases} \frac{B_k}{U_l} & \text{for } k < \kappa(l) \\ 1 - \sum_{t=1}^{\kappa(l)-1} z^*_{tl} & \text{for } k = \kappa(l) \\ 0 & \text{otherwise.} \end{cases}$$

Proof. First of all, we show that z^* as defined above is feasible, i.e., satisfies constraints (8.15), (8.16), and (8.17). Fix some cluster $l \in \mathcal{C}$. Clearly, for all $k \neq \kappa(l)$ we have $z^*_{kl} \geq 0$ and by definition $\sum_{k=1}^{q(l)} z^*_{kl} = 1$. It remains to show that $z_{\kappa(l)l} \geq 0$ or, equivalently, $\sum_{k=1}^{\kappa(l)-1} z^*_{kl} \leq 1$. The latter follows by exploiting the definition above and the fact that $\kappa(l)$ is the critical knapsack of l and thus

$$\sum_{k=1}^{\kappa(l)-1} z^*_{kl} = \sum_{k=1}^{\kappa(l)-1} \frac{B_k}{U_l} \leq 1.$$

Next, we show that any optimal solution (x, z) can be transformed into an optimal solution (x^*, z^*), where z^* is defined as above. We argue cluster by cluster. Fix some cluster $l \in \mathcal{C}$ and let $\mathcal{K}(l) = \{1, \ldots, q(l)\}$ be the (ordered) set of knapsacks. First of all, we assume without loss of generality that there is no knapsack k for which $z_{kl} > \frac{B_k}{U_l}$ because the cost assigned to knapsack k cannot exceed B_k. Furthermore, we assume that $\sum_{k \in \mathcal{K}(l)} z_{kl} = 1$ because by Assumption 8.2, we know that a solution (x, z) for which $\sum_{k \in \mathcal{K}(l)} z_{kl} < 1$ cannot be an optimal solution.

Let $k \in \mathcal{K}(l)$ be the first knapsack (in this order) satisfying $z_{kl} < z^*_{kl}$. If no such knapsack exists, then we are done because $z_{kl} \geq z^*_{kl}$ implies $z_{kl} = z^*_{kl}$ for all k by the definition of z^*. Let $k' > k$ be the last knapsack with $z_{k'l} > z^*_{k'l}$. We know that such knapsack exists because $\sum_{k \in \mathcal{K}(l)} z_{kl} = 1$ and any knapsack $k' < k$ has the property that $z^*_{kl} = \frac{B_k}{U_l}$. Hence, there has to be a knapsack $k' \in \mathcal{K}(l)$ with $z_{k'l} > z^*_{k'l}$ and that knapsack k' has to have a larger index than k.

Let $\Delta(k, k') = \min\{(z^*_{kl} - z_{kl})B_k, (z_{k'l} - z^*_{k'l})B_{k'}\}$. Given that the cost c_j of each set $j \in \mathcal{S}$ is independent of the knapsack to which it is assigned to and $B_{k'} \geq B_k$ by our ordering, we can reassign a total contribution, in terms

Figure 8.3 Illustration of the instance for which the (unique) optimal solution does not satisfy the bounded split property if $B_1 = 3$, $B_2 = 3$, and $U_1 = 4$.

of cost, of $\Delta(k, k')$ units from knapsack k' to k by changing some $x_{jk}, x_{jk'}$-variables accordingly. Note that this shift is feasible because every set $j \in \mathcal{S}$ that is fractionally assigned to knapsack k' also fits on knapsack k. Further, this shift does not change any of the y-variables and thus the objective function value remains the same. By continuing this way, we eventually obtain a feasible solution (x^*, z^*) which is also optimal. □

Given that it is possible to construct an optimal solution for the z-values, one might think of the following approximation algorithm for the MCPC. Fix the z^*-values as in Lemma 8.5, and let LP(z^*) be the respective LP relaxation. Note that LP(z^*) is basically the same as the LP relaxation of MCPK, where each knapsack $k \in \mathcal{K}(l)$, $l \in \mathcal{C}$, has a *reduced capacity* of $\min\{B_k, U_l z^*_{kl}\}$. So we could compute an optimal solution x^* to LP(z^*) and use Algorithm 8.1 to derive an integral solution x_{alg} satisfying

$$L(x_{\text{alg}}) \geq \frac{1}{2}(1 - \frac{1}{e})L(x^*, z^*) \geq \frac{1}{2}(1 - \frac{1}{e})\text{OPT}.$$

Unfortunately, however, this approach fails because of the following subtle point. If a set $j \in \mathcal{S}$ is fractionally assigned to some critical knapsack $k \in \mathcal{K}$ in the optimal LP solution x^* of LP(z^*), then it might be infeasible to assign j to k integrally. We could exclude these infeasible assignments beforehand by setting $x_{jk} = 0$ whenever $c_j > U_l z^*_{kl}$ for a critical knapsack k, but then an optimal LP solution might not recover a sufficiently large fraction of OPT.

To see that consider a collection $\mathcal{S} = \{S_1, S_2\}$ over the item set $\mathcal{I} = [2]$, where $S_1 = \{1\}$ and $S_2 = \{2\}$ with corresponding costs $c_1 = c_2 = 2$. The profits of the items are $p_1 = p_2 = M$, in which M is an arbitrarily large number. Moreover, there is a single cluster over the knapsack set $\mathcal{K} = [2]$ with a capacity of $U_1 = 4$. The two knapsacks have capacities $B_1 = B_2 = 3$. An illustration of the instance is given in Figure 8.3.

The z^*-values as given by Lemma 8.5 are $z^*_{11} = \frac{3}{4}$ and $z^*_{21} = 1 - \frac{3}{4} = \frac{1}{4}$. Consequently, the first knapsack has a reduced capacity of $B_1 = 3$ and the

Algorithm 8.2: $\frac{1}{3}(1-\frac{1}{e})$-approximation algorithm for the MCPC.

Fix z^* as in Lemma 8.5 and compute an optimal solution x^* to (LP(z^*)).
Derive a solution x from x^* that satisfies the bounded split property.
Let M be the corresponding $\mathcal{S}$-saturating matching.
Decompose the fractional solution x into x^1, x^2, x^3 as follows:

$$x^1_{jk} = \begin{cases} x^1_{jk} = 1 & \text{if } x_{jk} = 1 \\ x^1_{jk} = 0 & \text{otherwise} \end{cases}$$

$$x^2_{jk} = \begin{cases} x^2_{jk} = 1 & \text{if } x_{jk} \in (0,1),\ k \text{ not critical},\ \{j,k\} \in M \\ x^2_{jk} = 0 & \text{otherwise} \end{cases}$$

$$x^3_{jk} = \begin{cases} x^3_{jk} = 1 & \text{if } x_{jk} \in (0,1),\ k \text{ critical},\ \{j,k\} \in M \\ x^3_{jk} = 0 & \text{otherwise} \end{cases}$$

Output: $x_{\text{alg}} \in \arg\max\{L(x^1), L(x^2), L(x^3)\}$.

second knapsack has a reduced capacity of $U_1 z^*_{21} = 1$. If we exclude assigning a set S_j to a knapsack k whenever its cost exceeds the reduced capacity of k, then neither S_1 nor S_2 can be assigned to knapsack 2. As a result, an optimal LP solution recovers a total profit of $\frac{3}{2}M$ only. On the other hand, it is easy to verify that assigning sets S_1 and S_2 to knapsacks 1 and 2, respectively, is a feasible solution for MCPC and achieves a total profit of $2M > \frac{3}{2}M$.

Instead, we can fix this problem using a slightly more refined algorithm, which is given in Algorithm 8.2. In this algorithm, we decompose the fractionally assigned sets into two solutions, one using only non-critical knapsacks and one that only uses the critical knapsacks. Choosing the better of those two solutions and the integral solution, gives a $\frac{1}{3}(1-\frac{1}{e})$-approximation algorithm for MCPC, as we prove in Theorem 8.3.

Theorem 8.3. *Algorithm 8.2 is a $\frac{1}{3}(1-\frac{1}{e})$-approximation algorithm for MCPC.*

Proof. Note that by Theorem 8.1 the fractional solution x derived in the algorithm is a feasible solution to (LP(z^*)). Clearly, x^1 is a feasible (integral) solution. Further, the $\mathcal{S}$-saturating matching M ensures that $c_j \le B_k$ for every edge $\{j,k\} \in M$. In particular, this implies that the solution x^2 is feasible because for every (non-critical) knapsack $k \in \mathcal{K}(l)$, $l \in \mathcal{C}$, we have $B_k = U_l z^*_{kl}$ by Lemma 8.5, and thus $\sum_{j\in\mathcal{S}} c_j x^2_{jk} \le B_k = U_l z^*_{kl}$.

It remains to argue that x^3 is feasible. This holds because for every cluster $l \in \mathcal{C}$ there is at most one set $j \in \mathcal{S}$ assigned to this cluster, if there is any set then it is the set assigned to the critical knapsack $\kappa(l) \in \mathcal{K}(l)$ with $\{j,k\} \in M$. In particular, for $k = \kappa(l)$ we have $\sum_{j\in\mathcal{S}} c_j x^3_{jk} \le B_k \le U_l$, where the latter inequality holds because of Assumption 8.1.

It remains to bound the approximation factor of the algorithm. Using the

same arguments as in the proof of Theorem 8.2, we obtain

$$\begin{aligned} L(x_{\text{alg}}) &\geq \frac{1}{3}(L(x^1) + L(x^2) + L(x^3)) = \frac{1}{3}(F(x^1) + F(x^2) + F(x^3)) \\ &\geq \frac{1}{3}F(x) \geq \frac{1}{3}F(x^*) \geq \frac{1}{3}\left(1 - \frac{1}{e}\right)L(x^*) = \frac{1}{3}\left(1 - \frac{1}{e}\right)L(x^*, z^*) \\ &\geq \frac{1}{3}\left(1 - \frac{1}{e}\right)\text{OPT}. \end{aligned}$$

The last equality holds because x^* is an optimal solution to (LP(z^*)). □

8.6 Multiple knapsack problem with cluster constraints

The technique that we used in Section 8.5 to approximate the clustered variant of the maximum coverage problem can also be applied to the MKPC. Recall that in this problem we have $\mathcal{S} = \mathcal{I}$ and each set contains a single item, i.e., $S_j = \{j\}$ for all $j \in \mathcal{S}$. We first derive, in Section 8.6.1, a $\frac{1}{3}$-approximation algorithm for MKPC, and in Section 8.6.2, we present a more sophisticated iterative rounding scheme that provides a $\frac{1}{2}$-approximation algorithm for certain special cases of MKPC.

Note that for MKPC, the notions of sets and items coincide, and we simply refer to them as items. In particular, each item $j \in \mathcal{S}$ now has a profit p_j and a cost c_j. Thus, we can also drop the y-variables in the ILP formulation of the problem. Throughout this section, we assume that the knapsacks in $\mathcal{K}$ are ordered by non-increasing capacities, i.e., if $k, k' \in \mathcal{K}$ with $k < k'$ then $B_k \geq B_{k'}$.

8.6.1 General clusters

The MKPC is a generalization of the classical multiple knapsack problem. For the MKP, the optimal solution of the LP relaxation satisfies the bounded split property (see, e.g., Shmoys and Tardos (1993)), and there is a natural greedy algorithm to find an optimal solution of the LP relaxation (see, e.g., Kellerer et al. (2004)). Here we exploit some ideas of the previous section to derive a greedy algorithm for MKPC. This greedy algorithm is given in Algorithm 8.3. In this section, we prove that it computes an optimal solution to the LP relaxation, and show that the constructed solution satisfies the bounded split property. Exploiting this, we can then easily obtain a $\frac{1}{3}$-approximation algorithm for MKPC.

Algorithm 8.3 first fixes the optimal z^*-variables as defined in Lemma 8.5, and then runs an adapted version of the greedy algorithm in Kellerer et al. (2004) on the instance with the reduced capacities. There is one difference

Algorithm 8.3: Algorithm to compute an optimal LP solution for MKPC.

```
Fix z* as in Lemma 8.5 and initialize x* = 0.
Order the items by non-increasing efficiency ratios: p_1/c_1 ≥ ··· ≥ p_n/c_n.
for j = 1, ..., n do
    while Σ_{k∈K} x*_jk < 1 do
        Find the smallest capacity knapsack k that can hold j and has residual
          capacity, i.e., k := arg max_{k'∈K}{B_k' | B_k' ≥ c_j and res_{C(k')}(x*, z*) > 0}.
        if No such knapsack exists then
            Continue to the next item
        else
            Set x*_jk = min{1 − Σ_{k∈K} x*_jk, (1/c_j) res_{C(k)}(x*, z*)}.
        end
    end
end
```

in the assignment of items to knapsacks between our algorithm and that of Kellerer et al. (2004). Their greedy algorithm operates on a per-knapsack basis, while our algorithm proceeds on a per-item basis.

In Algorithm 8.3, the items are assigned in non-increasing order of their *efficiency ratios*, where the efficiency ratio of item $j \in \mathcal{S}$ is defined as $\frac{p_j}{c_j}$. When item j is considered, it is assigned to the knapsack with the smallest capacity that can hold the item and has some positive residual capacity. More formally, a knapsack $k \in \mathcal{K}$ *can hold* item $j \in \mathcal{S}$ if $B_k \geq c_j$. Furthermore, we define the *residual capacity* of knapsack k with respect to (x^*, z^*) as

$$res_k(x^*, z^*) := U_{\mathcal{C}(k)} z^*_{k\mathcal{C}(k)} - \sum_{j \in \mathcal{S}} c_j x^*_{jk}.$$

We continue this way until either item j is assigned completely, possibly split over several knapsacks, or all knapsacks that can hold j have zero residual capacity. We then continue with the next item in the order. We show in Lemma 8.6 that Algorithm 8.3 computes an optimal fractional solution.

Lemma 8.6. *Algorithm 8.3 computes an optimal solution (x^*, z^*) to the LP relaxation of MKPC.*

Proof. We show that any optimal solution (x, z^*) can be transformed into the solution (x^*, z^*) output by Algorithm 8.3. Let $j \in \mathcal{S}$ be the first item such that $\sum_{k \in \mathcal{K}} x^*_{jk} < \sum_{k \in \mathcal{K}} x_{jk} \leq 1$. If no such item exists we are done because then the total profit of (x^*, z^*) is at least the profit of (x, z^*). Note that since $\sum_{k \in \mathcal{K}} x^*_{jk} < 1$, the combined residual capacity on all knapsacks $k \in \mathcal{K}$ that can hold j is $\sum_{k \in \mathcal{K}} c_j x^*_{jk}$ before iteration j of Algorithm 8.3. However, if we had assign, in iterations $1, \ldots, j-1$ of Algorithm 8.3, x_{jk} instead of x^*_{jk}, then the residual capacity on all knapsacks that could hold item j would have been at

least $\sum_{k\in\mathcal{K}} c_j x_{jk} > \sum_{k\in\mathcal{K}} c_j x^*_{jk}$. Hence, we know that

$$\sum_{j'=1}^{j-1}\sum_{k\in\mathcal{K}} x_{j'k} < \sum_{j'=1}^{j-1}\sum_{k\in\mathcal{K}} x^*_{j'k}.$$

In particular, this implies that there has to be an item $j' \in \mathcal{S}$ for which

$$1 \geq \sum_{k\in\mathcal{K}} x^*_{j'k} > \sum_{k\in\mathcal{K}} x_{j'k},$$

and

$$\frac{p_{j'}}{c_{j'}} \geq \frac{p_j}{c_j}.$$

Let k' be a knapsack for which $x^*_{jk'} < x_{jk'}$. Let us change the solution x such that the contribution of item j on knapsack k' decreases and the contribution of item j' on knapsack k' increases. Let

$$\Delta(j,j',k') := \min\left\{c_j(x_{jk'} - x^*_{jk'}),\, c_{j'}\left(\sum_{k\in\mathcal{K}} x^*_{j'k} - \sum_{k\in\mathcal{K}} x_{j'k}\right)\right\},$$

and let us define $x(\Delta)$ as follows:

$$x(\Delta)_{jk'} = x_{jk'} - \frac{\Delta(j,j',k')}{c_j}$$
$$x(\Delta)_{j'k'} = x_{j'k'} + \frac{\Delta(j,j',k')}{c_{j'}},$$

and otherwise $x(\Delta) = x$. We first show that the new solution $x(\Delta)$ is feasible. Since by definition $x_{jk'} > x^*_{jk'}$, $\sum_{k\in\mathcal{K}} x^*_{j'k} > \sum_{k\in\mathcal{K}} x_{j'k}$, and $c_j, c_{j'} > 0$, we know that $\Delta(j,j',k') > 0$. An immediate consequence is that $x(\Delta)_{j'k'} \geq 0$ and $\sum_{k\in\mathcal{K}} x(\Delta)_{jk} \leq 1$. Furthermore, from the definition of $\Delta(j,j',k')$ we obtain the following two inequalities:

$$x(\Delta)_{jk'} = x_{jk'} - \frac{\Delta(j,j',k')}{c_j} \geq x_{jk'} - \frac{c_j(x_{jk'} - x^*_{jk'})}{c_j} = x^*_{jk'} \geq 0,$$

and

$$\begin{aligned}\sum_{k\in\mathcal{K}} x(\Delta)_{j'k} &= \frac{\Delta(j,j',k')}{c_{j'}} + \sum_{k\in\mathcal{K}} x_{j'k} \\ &\leq \frac{c_{j'}\left(\sum_{k\in\mathcal{K}} x^*_{j'k} - \sum_{k\in\mathcal{K}} x_{j'k}\right)}{c_{j'}} + \sum_{k\in\mathcal{K}} x_{j'k} \\ &= \sum_{k\in\mathcal{K}} x^*_{j'k} \leq 1.\end{aligned}$$

Hence, we know that $x(\Delta)_{jk} \geq 0$ for every $j \in \mathcal{S}$ and $k \in \mathcal{K}$, and that constraint (8.3) is satisfied for every for every $j \in \mathcal{S}$. Remains to show that the solution $x(\Delta)$ satisfies the budget constraints. The only knapsack for which solution $x(\Delta)$ is different from x is knapsack k'. The costs assigned on knapsack k' for $x(\Delta)$ are

$$\begin{aligned}\sum_{j'' \in \mathcal{S}} c_{j''} x(\Delta)_{j''k'} &= -c_j \left(\frac{\Delta(j, j', k')}{c_j} \right) + c_{j'} \left(\frac{\Delta(j, j', k')}{c_{j'}} \right) + \sum_{j'' \in \mathcal{S}} c_{j''} x_{j''k'} \\ &= \sum_{j'' \in \mathcal{S}} c_{j''} x_{j''k'} \leq B_{k'}.\end{aligned}$$

As a result, the solution $x(\Delta)$ is feasible. Finally, we show that he difference in the objective for $x(\Delta)$ and x is at least 0:

$$\begin{aligned}\sum_{j'' \in \mathcal{S}} p_{j''} \left(\sum_{k \in \mathcal{K}} x(\Delta)_{j''k} - \sum_{k \in \mathcal{K}} x_{j''k} \right) &= p_{j'} \frac{\Delta(j, j', k')}{c_{j'}} - p_j \frac{\Delta(j, j', k')}{c_j} \\ &= \Delta(j, j', k') \left(\frac{p_{j'}}{c_{j'}} - \frac{p_j}{c_j} \right) \geq 0,\end{aligned}$$

in which the last inequality follows from the fact that $\frac{p_j}{c_j} \leq \frac{p_{j'}}{c_{j'}}$. All in all, it is possible to transform the solution (x, z^*) into the solution (x^*, z^*) as constructed by Algorithm 8.3. □

After we have shown that Algorithm 8.3 gives the optimal solution for the LP relaxation, we need to show that its solution also satisfies the bounded split property. For that reason, we need to introduce some definitions. We say that an item $j \in \mathcal{S}$ is a *split item* if it is fractionally assigned to one or multiple knapsacks. Let SS refer to the set of all split items. Next, we introduce the notion of a split item of a knapsack.

Definition 8.4 (Split set)**.** *An item $j \in SS$ is a* split item of knapsack $k \in \mathcal{K}$ *if k is the knapsack with the smallest capacity to which j is assigned in Algorithm 8.3.*

Using this definition, we show below that the solution obtained by Algorithm 8.3 satisfies the bounded split property.

Lemma 8.7. *Algorithm 8.3 computes a solution (x^*,z^*) to the LP relaxation of MKPC which satisfies the bounded split property.*

Proof. Let $H_{x^*} = (\mathcal{S} \cup \mathcal{K}, E_{x^*})$ be the support graph of x^*. Note that the set of split items SS corresponds to the set of non-isolated nodes in $\mathcal{S}$ of the support

graph H_{x^*}. Let M be the set of all edges $\{j,k\} \in E_{x^*}$ such that $j \in SS$ is a split item of knapsack $k \in \mathcal{K}$. The proof follows if we can show that M is an $\mathcal{S}$-saturating matching.

Clearly, every node $j \in SS$ is matched to some knapsack. Consider an item $j \in SS$ and let k be the knapsack for which j is a knapsack split, i.e., $\{j,k\} \in M$. By definition, k is the first, i.e., the smallest capacity, knapsack to which Algorithm 8.3 assigns some fractional contribution of j. By construction, j is only fractionally assigned to knapsack k because the residual capacity of k is insufficient to assign j integrally. However, this implies that no other item can be a split item of knapsack k. Thus, M is a matching. □

In the analysis of Algorithm 8.2 for the MCPC, a factor $(1-\frac{1}{e})$ is lost in the approximation ratio by transforming the optimal solution of the LP relaxation to a solution that satisfies the bounded split property. By Lemmas 8.6 and 8.7, we can avoid this loss here. Hence, it follows that Algorithm 8.2 is a $\frac{1}{3}$-approximation algorithm for the MKPC. This claim is summarized in Theorem 8.4.

Theorem 8.4. *Algorithm 8.2 is a $\frac{1}{3}$-approximation algorithm for the MKPC.*

8.6.2 Isolation property

In this section, we derive $\frac{1}{2}$-approximation algorithm for instances of MKPC that satisfy a certain *isolation property* based on iterative rounding. We first introduce some more notation to define the isolation property.

Let (x^*, z^*) be the optimal solution to the LP relaxation of MKPC computed by Algorithm 8.3. We say that an item $j \in \mathcal{S}$ is an *unsplit item* if it is integrally assigned to some knapsack, i.e., $x^*_{jk} = 1$ for some $k \in \mathcal{K}$. Recall that an item $j \in \mathcal{S}$ is a *split item* if it is fractionally assigned to one or multiple knapsacks. Let US and SS refer to the sets of unsplit and split items, respectively. Recall from Definition 8.4, the definition of a split item of knapsack k. As argued in the proof of Lemma 8.7, the split item $j \in SS$ of knapsack $k \in \mathcal{K}$ is unique, and we use $\varsigma_k = j$ to refer to the split item of knapsack k. Furthermore, we use $US(k)$ to denote the set of all unsplit items assigned to knapsack k. For each cluster $l \in \mathcal{C}$, we denote by $US(l) = \cup_{k \in \mathcal{K}(l)} US(k)$ the set of all unsplit items and by $SS(l) = \cup_{k \in \mathcal{K}(l)} \varsigma_k$ the set of all split items assigned to l. Using these definitions, it is possible to define an *isolated cluster*.

Definition 8.5 (Isolated Cluster)**.** *A cluster $\iota \in \mathcal{C}$ is said to be* isolated *if for every item $j \in US(l) \cup SS(l)$ assigned to some cluster $l \neq \iota$ it holds that $x^*_{jk} = 0$ for all $k \in \mathcal{K}(\iota)$.*

Intuitively, a cluster ι is isolated if there is no x^*-contribution to cluster ι coming from items assigned to other clusters $l \neq \iota$ or put differently, the total x^*-contribution of items to knapsacks in ι is entirely due to the unsplit and split items assigned to cluster ι. The definition of an isolated cluster is used below to define when an instance of MKPC satisfies the *isolation property*.

Definition 8.6 (Isolation property)**.** *A class of instances of MKPC satisfies the* isolation property *if, after removing an arbitrary set of clusters, there always exists an isolated cluster.*

A practical situation in which the isolation property holds is when standardized containers with the same capacity are used. Moreover, also the case in which there is a single cluster satisfies the isolation property. In general, the isolation property holds true for instances whose clusters can be *disentangled* in the sense that we can impose an order on the set of clusters $\mathcal{C} = \{1, 2, \ldots, q\}$ such that for any two clusters $l, l' \in \mathcal{C}$ with $l < l'$ it holds that $\min_{k \in \mathcal{K}(l)}\{B_k\} \geq \max_{k \in \mathcal{K}(l')}\{B_k\}$. Lemma 8.8 below shows that the isolation property is satisfied if the clusters are disentangled.

Lemma 8.8. *If the set of clusters $\mathcal{C} = \{1, 2, \ldots, q\}$ is disentangled then $\iota = q$ is an isolated cluster.*

Proof. Consider a cluster $l \neq q$ and an arbitrary item $j \in US(l) \cup SS(l)$ assigned to l. The claim follows if we can prove that $x^*_{jk} = 0$ for all $k \in \mathcal{K}(q)$. Clearly, this holds true if $j \in US(l)$ because $x_{jk'} = 1$ for some $k' \in \mathcal{K}(l)$ by definition. Assume $j \in SS(l)$ and let $k' \in \mathcal{K}(l)$ be the knapsack for which j is the split item, i.e., $\varsigma_{k'} = j$. By definition, k' is the knapsack of smallest capacity to which j was assigned fractionally by Algorithm 8.3. Further, it must hold that $B_{k'} \geq B_k$ for each $k \in \mathcal{K}(q)$ because $l < q$ and the clusters are disentangled. But this implies that $x_{jk} = 0$ for all $k \in \mathcal{K}(q)$. □

After we have defined an isolated cluster, we now start with deriving the $\frac{1}{2}$-approximation algorithm for the MKPC that satisfies the isolation property. The first step of our iterative rounding scheme is given in Lemma 8.9. In this lemma, it is shown that we can always find a feasible assignment σ which recovers at least half of the fractional profit of an isolated cluster ι. Recall that $\mathcal{S}_\sigma$ refers to the set of items assigned under σ.

Lemma 8.9. *Let ι be an isolated cluster. Then there exists a feasible assignment $\sigma : US(\iota) \cup SS(\iota) \to \mathcal{K}(\iota)$ such that*

$$\sum_{j \in \mathcal{S}_\sigma} p_j \geq \sum_{j \in \mathcal{S}_\sigma} \sum_{k \in \mathcal{K}(\iota)} p_j x^*_{jk} \geq \frac{1}{2}\left(\sum_{j \in \mathcal{S}} \sum_{k \in \mathcal{K}(\iota)} p_j x^*_{jk}\right). \tag{8.18}$$

Proof. We only consider the knapsacks $k \in \mathcal{K}(\iota)$ for which $z^*_{k\iota} > 0$. By Lemma 8.5, there are $\kappa(\iota)$ such knapsacks, where $\kappa(\iota)$ denotes the critical knapsack of cluster ι. Recall that we assume that the knapsacks are ordered by non-increasing capacities.

Below, we use a few case distinctions to prove that we can always construct two feasible assignments σ_1 and σ_2 such that $\mathcal{S}_{\sigma_1} \cup \mathcal{S}_{\sigma_2} = US(\iota) \cup SS(\iota)$ and $\mathcal{S}_{\sigma_1} \cap \mathcal{S}_{\sigma_2} = \emptyset$. By exploiting that ι is an isolated cluster, we obtain

$$\sum_{j\in\mathcal{S}} \sum_{k\in\mathcal{K}(\iota)} p_j x^*_{jk} = \sum_{j\in US(\iota)\cup SS(\iota)} \sum_{k\in\mathcal{K}(\iota)} p_j x^*_{jk}$$
$$= \sum_{j\in\mathcal{S}_{\sigma_1}\cup\mathcal{S}_{\sigma_2}} \sum_{k\in\mathcal{K}(\iota)} p_j x^*_{jk} \leq \sum_{j\in\mathcal{S}_{\sigma_1}\cup\mathcal{S}_{\sigma_2}} p_j.$$

Inequality (8.18) then follows by choosing $\sigma \in \{\sigma_1, \sigma_2\}$ as the assignment of maximum profit.

First, assume that $\sum_{j\in SS(\iota)} c_j \leq U_\iota$. Then, assigning every split item $\varsigma_k \in SS(\iota)$ to knapsack k corresponds to a feasible assignment σ_1. Further, assigning each unsplit item $j \in US(\iota)$ to its respective knapsack corresponds to a feasible assignment σ_2. The claim follows.

We can thus assume that $\sum_{j\in SS(\iota)} c_j > U_\iota$. Let us define the set of *fractionally split* items $FS(\iota) \subseteq SS(\iota)$ of cluster ι as

$$FS(\iota) = \{\varsigma_{k'} \in SS(\iota) \mid \sum_{k\in\mathcal{K}(\iota)} x^*_{jk} < 1\}.$$

By definition, the split item of knapsack 1 is always a fractional split and thus $\varsigma_1 \in FS(\iota)$.

We first consider the case $|FS(\iota)| = 1$ and thus ς_1 is the only fractional split item. Note that then all items in $US(\iota) \cup SS(\iota)$ except ς_1 are completely assigned to cluster ι and ς_1 is only partially assigned to cluster ι. Thus,

$$\sum_{j\in US(\iota)\cup SS(\iota)} c_j - c_{\varsigma_1} \leq U_\iota < \sum_{j\in US(\iota)\cup SS(\iota)} c_j.$$

Since $\sum_{j\in SS(\iota)} c_j > U_\iota$, the first inequality implies that $\sum_{j\in US(\iota)} c_j \leq c_{\varsigma_1}$. The latter implies that assigning all unsplit items in $US(\iota)$ to the first knapsack and each split item $\varsigma_k \in SS(\iota) \setminus \{\varsigma_1\}$ to its respective knapsack k is a feasible assignment σ_1. Assigning the only missing item ς_1 by its own to knapsack 1 is a feasible assignment σ_2. The claim follows.

Next consider the case $|FS(\iota)| > 1$. In this case, we focus on finding an assignment for two knapsacks with a fractional split item. After that is done, we are either left with none or a single fractional split item. The previously

described assignment can be applied for finding an assignment for the remaining knapsacks.

Let $k_1, k_2 \in \mathcal{K}(\iota)$ be two knapsacks with $k_1 > k_2$ for which $\varsigma_{k_1}, \varsigma_{k_2} \in FS(\iota)$. Note that by our ordering we have $B_{k_1} \leq B_{k_2}$. We introduce some more notation. Given a subset $S \subseteq \mathcal{S}$ of items, let $P(S)$ and $C(S)$ denote the total fractional profit and cost, respectively, of the items in S in cluster ι, i.e.,

$$P(S) = \sum_{j \in S} \sum_{k \in \mathcal{K}(\iota)} p_j x^*_{jk} \quad \text{and} \quad C(S) = \sum_{j \in S} \sum_{k \in \mathcal{K}(\iota)} c_j x^*_{jk}.$$

For notational convenience, we use $P_k^{US} = P(US(k))$ and $P_k^{\varsigma_k} = P(\{\varsigma_k\})$ to refer to the total (fractional) profit of the unsplit items in $US(k)$ and the fractional split item ς_k, respectively. Similarly, we use $C_k^{US} = C(US(k))$ and $C_k^{\varsigma_k} = C(\{\varsigma_k\})$ to refer to the total (fractional) cost of the unsplit items in $US(k)$ and the fractional split item ς_k, respectively. Finally, define $P_k = P_k^{US} + P_k^{\varsigma_k}$ and $C_k = C_k^{US} + C_k^{\varsigma_k}$.

We will exhibit an assignment for knapsacks k_1 and k_2 whose profit is at least $\frac{1}{2}(P_{k_1} + P_{k_2})$, and whose cost is at most $C_{k_1} + C_{k_2}$. To this aim, we distinguish three cases:

1. $c_{\varsigma_{k_1}} + c_{\varsigma_{k_2}} \leq C_{k_1} + C_{k_2}$,
2. $c_{\varsigma_{k_1}} + c_{\varsigma_{k_2}} > C_{k_1} + C_{k_2}$ and $C_{k_1}^{US} + C_{k_2}^{US} + c_{\varsigma_{k_1}} \leq C_{k_1} + C_{k_2}$, and
3. $c_{\varsigma_{k_1}} + c_{\varsigma_{k_2}} > C_{k_1} + C_{k_2}$ and $C_{k_1}^{US} + C_{k_2}^{US} + c_{\varsigma_{k_1}} > C_{k_1} + C_{k_2}$.

In the first case, assigning the split items ς_{k_1} and ς_{k_2} to their respective knapsacks k_1 and k_2 is a feasible assignment. Also, assigning $US(k_1)$ and $US(k_2)$ to knapsacks k_1 and k_2, respectively, is a feasible assignment. Thus, choosing the assignment of maximum fractional profit recovers at least $\frac{1}{2}(P_{k_1} + P_{k_2})$ as claimed.

In the second case, the conditions imply that

$$c_{\varsigma_{k_2}} > C_{k_1} + C_{k_2} - c_{\varsigma_{k_1}} \geq C_{k_1}^{US} + C_{k_2}^{US}.$$

This, in combination with $c_{\varsigma_{k_2}} \leq B_{k_2}$, implies that assigning ς_{k_1} to knapsack k_1, and $US(k_1)$ and $US(k_2)$ to knapsack k_2 is a feasible assignment. Further, assigning ς_{k_2} on its own is a feasible assignment as well. Again, selecting the assignment of maximum fractional profit recovers at least $\frac{1}{2}(P_{k_1} + P_{k_2})$.

In the last case, we exploit the fact that $\varsigma_{k_1} \in FS(\iota)$, which implies that knapsack k_2 must have been completely filled already at the time when item ς_{k_1} was assigned to knapsack k_1. As $B_{k_1} \leq B_{k_2}$, we know that it is possible to assign ς_{k_1} to knapsack k_2. As a consequence, the items in $US(k_2)$ were assigned

before ς_{k_1} in Algorithm 8.3, and thus they have a larger efficiency ratio, i.e., $\frac{p_{\varsigma_{k_1}}}{c_{\varsigma_{k_1}}} \leq \frac{p_j}{c_j}$ for every item $j \in US(k_2)$. In particular, this implies that $\frac{p_{\varsigma_{k_1}}}{c_{\varsigma_{k_1}}} \leq \frac{P^{US}_{k_2}}{C^{US}_{k_2}}$. Using this inequality, we can bound the fractional profit of ς_{k_1} by

$$P^{\varsigma_{k_1}}_{k_1} = \sum_{k \in \mathcal{K}(\iota)} p_{\varsigma_{k_1}} x^*_{\varsigma_{k_1} k} \leq \frac{P^{US}_{k_2}}{C^{US}_{k_2}} \sum_{k \in \mathcal{K}(\iota)} c_{\varsigma_{k_1}} x^*_{\varsigma_{k_1} k} = \frac{C^{\varsigma_{k_1}}_{k_1}}{C^{US}_{k_2}} P^{US}_{k_2}.$$

Finally, note that, by definition, the split item ς_{k_2} is assigned after all items in $US(k_2)$ were assigned to k_2 and thus also

$$\frac{p_{\varsigma_{k_2}}}{c_{\varsigma_{k_2}}} \leq \frac{P^{US}_{k_2}}{C^{US}_{k_2}}.$$

Using the same arguments as above, we can conclude that

$$P^{\varsigma_{k_2}}_{k_2} \leq \frac{C^{\varsigma_{k_2}}_{k_2}}{C^{US}_{k_2}} P^{US}_{k_2}.$$

Thus,

$$P^{\varsigma_{k_1}}_{k_1} + P^{\varsigma_{k_2}}_{k_2} \leq \frac{C^{\varsigma_{k_1}}_{k_1} + C^{\varsigma_{k_2}}_{k_2}}{C^{US}_{k_2}} P^{US}_{k_2}. \tag{8.19}$$

The second condition of the assumptions of Case 3 implies that $c_{\varsigma_{k_1}} > C^{\varsigma_{k_1}}_{k_1} + C^{\varsigma_{k_2}}_{k_2}$. Using that each item $j \in US(k_2)$ was assigned before ς_{k_1} in Algorithm 8.3, it follows that $c_j > B_{k_1}$. Thus, $C^{US}_{k_2} > B_{k_1} \geq c_{\varsigma_{k_1}}$. Combining this with the inequality above, we obtain $C^{US}_{k_2} > C^{\varsigma_{k_1}}_{k_1} + C^{\varsigma_{k_2}}_{k_2}$. Exploiting this inequality with inequality (8.19), we obtain that

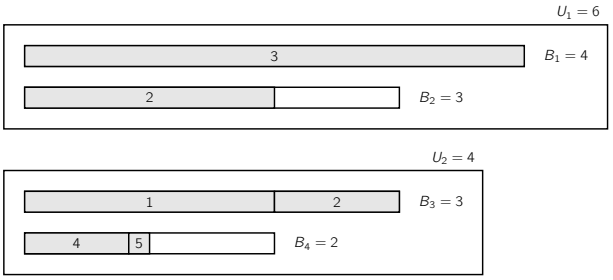
$$P^{\varsigma_{k_1}}_{k_1} + P^{\varsigma_{k_2}}_{k_2} < P^{US}_{k_2}.$$

As a consequence, assigning all unsplit items in $US(k_1)$ and $US(k_2)$ to their respective knapsacks k_1 and k_2 is a feasible assignment of fractional profit

$$P^{US}_{k_1} + P^{US}_{k_2} > P^{US}_{k_1} + \frac{1}{2} P^{US}_{k_2} + \frac{1}{2}(P^{\varsigma_{k_1}}_{k_1} + P^{\varsigma_{k_2}}_{k_2}) \geq \frac{1}{2}(P_{k_1} + P_{k_2}),$$

as desired. □

Although the assignment constructed in the proof of Lemma 8.9 recovers at least half of the fractional profit that is assigned to cluster ι, it does not take into account the contribution of items in $SS(\iota)$ to other clusters. Suppose one ignores the items in $SS(\iota)$ because they do not contribute sufficiently enough



item j	p_j	c_j
1	$2\frac{1}{5}$	$2\frac{1}{10}$
2	$2\frac{9}{10}$	$2\frac{9}{10}$
3	$\frac{1}{2}$	4
4	$\frac{1}{10}$	$\frac{9}{10}$
5	$\frac{1}{5}$	2

Figure 8.4 Illustration of an instance and its optimal LP solution, for which ignoring the contribution of $SS(\iota)$ outside cluster ι does not result in a recovery of half of the fractional profit.

to the profit inside cluster ι. In that case, one might not be able to recover half of the total fractional profit over all clusters. To see that consider the following example.

In Figure 8.4, an instance with five items, four knapsacks, and two clusters is given. The knapsacks have capacities $B_1 = 4$, $B_2 = B_3 = 3$, and $B_4 = 2$. Cluster 1 contains knapsacks 1 and 2 and has capacity $U_1 = 6$, and cluster 2 contains knapsacks 3 and 4 and has capacity $U_2 = 4$. The optimal solution to the LP relaxation computed by Algorithm 8.3 is as depicted in Figure 8.4. The total value of the LP relaxation is $5\frac{71}{100}$.

In cluster 2, an isolated cluster, two splits items are only fractionally assigned to cluster 2, namely items 2 and 5. Both the set of items $\{2, 5\}$ and $\{1, 4, 5\}$ have a cost higher than the cluster capacity and thus do not form a feasible assignment. Consequently, as shown in Lemma 8.9, the unsplit items 1 and 4 recover at least half of the entire profit made in cluster 2 and are thus selected as the assignment. These two items together have a profit of $2\frac{3}{10}$. In cluster 1, there is no split item and $US(1)$ only consists of item 3. As a result, for cluster 2, item 3 is selected as the assignment. The total profit of the resulting integral assignment is $p_1 + p_3 + p_4 = 2\frac{4}{5} < \frac{1}{2}(5\frac{71}{100})$.

For this instance, assigning item 2 alone corresponds to a feasible assignment that recovers half of the fractional profit as $p_2 = 2\frac{9}{10} \geq \frac{1}{2}(5\frac{71}{100})$. However, this is the split item of knapsack 3 in cluster 2, while its main contribution is assigned in cluster 1. That is why it is not taken into account if one uses the procedure described in Lemma 8.9.

Therefore, a more advanced iterative rounding scheme is needed that is described in Algorithm 8.4. In this algorithm, we first compute an optimal solution (x^*, y^*) to the LP relaxation of MKPC using Algorithm 8.3. Because of the isolation property, there exists an isolated cluster ι. We then apply Lemma 8.9 to obtain an assignment σ for cluster ι. After that, we fix the corresponding

Algorithm 8.4: Iterative rounding algorithm for MKPC with isolation property.

Let $\mathrm{LP}^{(1)}$ be the original LP relaxation of MKPC.
for $i = 1, \ldots, q$ **do**
  Compute an optimal solution $(x^{(i)}, z^{(i)})$ to $\mathrm{LP}^{(i)}$ using Algorithm 8.3.
  Identify an isolated cluster $\iota^{(i)}$ with respect to $(x^{(i)}, z^{(i)})$.
  Obtain a feasible assignment $\sigma^{(i)}$ for cluster $\iota^{(i)}$ using Lemma 8.9.
  Add the constraints (8.20)–(8.21) fixing the assignment $\sigma^{(i)}$ for $\iota^{(i)}$ to obtain $\mathrm{LP}^{(i+1)}$.
end

variables in the LP relaxation accordingly and repeat. By iterating this procedure, we obtain a sequence of isolated clusters $\iota^{(1)}, \ldots, \iota^{(q)}$ and assignments $\sigma^{(1)}, \ldots, \sigma^{(q)}$, where $\sigma^{(l)}$ is the assignment obtained by applying Lemma 8.9 to cluster $\iota^{(l)}$.

Let $\sigma(i) = \langle \sigma^{(1)}, \ldots, \sigma^{(i)} \rangle$ be the combined assignment for the first i clusters $\iota^{(1)}, \ldots, \iota^{(i)}$ that we obtain at the end of iteration i. The LP relaxation $\mathrm{LP}^{(i+1)}$ that we solve in iteration $i+1$ is then defined as the LP (8.13)–(8.17) with the following additional constraints:

$$x_{jk} = 1 \qquad \forall l \in \{\iota^{(1)}, \ldots, \iota^{(i)}\} \qquad \forall k \in \mathcal{K}(l) \qquad \forall j \in \mathcal{S}_{\sigma(i)}(k), \quad (8.20)$$

$$x_{jk} = 0 \qquad \forall l \in \{\iota^{(1)}, \ldots, \iota^{(i)}\} \qquad \forall k \in \mathcal{K}(l) \qquad \forall j \notin \mathcal{S}_{\sigma(i)}(k). \quad (8.21)$$

Note that with these constraints no item that is not in $\mathcal{S}_{\sigma(i)}$ can be assigned to knapsacks in clusters $\iota^{(1)}, \ldots, \iota^{(i)}$, and every item that is in $\mathcal{S}_{\sigma(i)}$ cannot be assigned to any knapsack in clusters $\iota^{(i+1)}, \ldots, \iota^{(q)}$. Let $(x^{(i+1)}, z^{(i+1)})$ be the optimal solution of $\mathrm{LP}^{(i+1)}$ computed by Algorithm 8.3 in iteration $i+1$. The next lemma establishes that the assignment for cluster $i+1$ according to the procedure of Lemma 8.9 recovers at least half of the optimal solution for the clusters 1 up to $i+1$.

Lemma 8.10. *Fix some $1 \leq i \leq q$ and let $\sigma(i)$ be the assignment at the end of iteration i. Further, let (x^*, z^*) be the optimal solution to the original LP relaxation constructed by Algorithm 8.3. Then*

$$\sum_{j \in \mathcal{S}_{\sigma(i)}} p_j \geq \frac{1}{2} \sum_{l \in \{\iota^{(1)}, \ldots, \iota^{(i)}\}} \sum_{k \in \mathcal{K}(l)} \sum_{j \in \mathcal{S}} p_j x^*_{jk}. \quad (8.22)$$

Proof. Throughout this proof, we assume for notational convenience that the clusters are renamed such that $\iota^{(l)} = l$ for each $1 \leq l \leq i$. For $i = 1$ the inequality follows from Lemma 8.9. Suppose the claim is true for $1, \ldots, i-1$, then we prove by induction that it remains true for i as well. We construct the assignment $\sigma(i)$ using the procedure of Lemma 8.9 with the solution $(x^{(i)}, z^{(i)})$.

For the profit of all items in $\mathcal{S}_{\sigma(i)}$, we have

$$\begin{aligned}
\sum_{j\in\mathcal{S}_{\sigma(i)}} p_j &= \sum_{j\in\mathcal{S}_{\sigma(i-1)}} p_j + \sum_{j\in\mathcal{S}_{\sigma(i)}} p_j \\
&\geq \sum_{j\in\mathcal{S}_{\sigma(i-1)}} p_j + \frac{1}{2}\left(\sum_{j\in\mathcal{S}}\sum_{k\in\mathcal{K}(i)} p_j x_{jk}^{(i)}\right) \\
&\geq \sum_{j\in\mathcal{S}_{\sigma(i-1)}} \left(\sum_{l=1}^{i-1}\sum_{k\in\mathcal{K}(l)} p_j x_{jk}^{*} + \sum_{k\in\mathcal{K}(i)} p_j x_{jk}^{*}\right) \\
&\quad + \frac{1}{2}\left(\sum_{j\in\mathcal{S}}\sum_{k\in\mathcal{K}(i)} p_j x_{jk}^{(i)}\right) \\
&\geq \sum_{j\in\mathcal{S}_{\sigma(i-1)}} \left(\sum_{l=1}^{i-1}\sum_{k\in\mathcal{K}(l)} p_j x_{jk}^{*}\right) \\
&\quad + \frac{1}{2}\left(\sum_{k\in\mathcal{K}(i)}\left(\sum_{j\in\mathcal{S}} p_j x_{jk}^{(i)} + \sum_{j\in\mathcal{S}_{\sigma(i-1)}} p_j x_{jk}^{*}\right)\right) \\
&\geq \sum_{j\in\mathcal{S}_{\sigma(i-1)}} \left(\sum_{l=1}^{i-1}\sum_{k\in\mathcal{K}(l)} p_j x_{jk}^{*}\right) + \frac{1}{2}\left(\sum_{k\in\mathcal{K}(i)}\sum_{j\in\mathcal{S}} p_j x_{jk}^{*}\right) \\
&\geq \frac{1}{2}\left(\sum_{j\in\mathcal{S}}\sum_{l=1}^{i-1}\sum_{k\in\mathcal{K}(l)} p_j x_{jk}^{*}\right) + \frac{1}{2}\left(\sum_{k\in\mathcal{K}(i)}\sum_{j\in\mathcal{S}} p_j x_{jk}^{*}\right) \\
&= \frac{1}{2}\left(\sum_{j\in\mathcal{S}}\sum_{l=1}^{i}\sum_{k\in\mathcal{K}(l)} p_j x_{jk}^{*}\right).
\end{aligned}$$

Here, the first inequality follows from Lemma 8.9. The second inequality exploits that for any $j \in \mathcal{S}_{\sigma(i-1)}$ it holds that $\sum_{l=1}^{i}\sum_{k\in\mathcal{K}(l)} x_{jk}^{*} \leq 1$. In the third inequality, the terms are rearranged and a fraction of $\frac{1}{2}(\sum_{j\in\mathcal{S}_{\sigma(i-1)}}\sum_{k\in\mathcal{K}(i)} p_j x_{jk}^{*})$ is discarded.

The fourth inequality trivially holds if for every $j \notin \mathcal{S}_{\sigma(i-1)}$ holds that $\sum_{k\in\mathcal{K}(i)} x_{jk}^{(i)} \geq \sum_{k\in\mathcal{K}(i)} x_{jk}^{*}$. Otherwise, we make use of the fact that the solution $x^{(i)}$ is the optimal solution for the items in $\mathcal{S}\setminus\mathcal{S}_{\sigma(i-1)}$ on clusters $i, i+1, \ldots, q$ constructed via Algorithm 8.3. Moreover, cluster i is an isolated cluster, and thus

$$\sum_{j\notin US(i)\cup SS(i)}\sum_{k\in\mathcal{K}(i)} x_{jk}^{(i)} = 0.$$

Let $j \notin \mathcal{S}_{\sigma(i-1)}$ be an item for which

$$\sum_{k \in \mathcal{K}(i)} x_{jk}^{(i)} < \sum_{k \in \mathcal{K}(i)} x_{jk}^{*}.$$

This implies that there was not enough residual capacity left on cluster i at iteration j of Algorithm 8.3 to assign item j to an extent of $\sum_{k \in \mathcal{K}(i)} x_{jk}^{*}$. Therefore, there must be an item j' with $\frac{p_{j'}}{c_{j'}} \geq \frac{p_j}{c_j}$ for which

$$\sum_{k \in \mathcal{K}(i)} x_{j'k}^{(i)} > \sum_{k \in \mathcal{K}(i)} x_{j'k}^{*}.$$

Hence, it follows that

$$\sum_{j \notin \mathcal{S}_{\sigma(i)}} \sum_{k \in \mathcal{K}(i)} p_j x_{jk}^{(i)} \geq \sum_{j \notin \mathcal{S}_{\sigma(i)}} \sum_{k \in \mathcal{K}(i)} p_j x_{jk}^{*}.$$

Finally, the fifth inequality holds by our induction hypothesis, and the final equality follows from the combination of the two sums. □

With the use of Lemma 8.10, the claim that Algorithm 8.4 is a $\frac{1}{2}$-approximation algorithm follows, as is shown in the following theorem.

Theorem 8.5. *Algorithm 8.4 is a $\frac{1}{2}$-approximation algorithm for instances of MKPC satisfying the isolation property.*

Proof. By Lemma 8.10, the final assignment $\sigma = \sigma(q)$ returned by the algorithm has profit at least

$$\sum_{j \in \mathcal{S}_\sigma} p_j \geq \frac{1}{2} \sum_{l \in \mathcal{C}} \sum_{k \in \mathcal{K}(l)} \sum_{j \in \mathcal{S}} p_j x_{jk}^{*} \geq \frac{1}{2} \mathrm{OPT}.$$

□

8.7 Capacitated facility location problem with cluster constraints

The technique to add a decision variable to the LP formulation to distribute the cluster capacity over the knapsacks in a cluster does not only apply to the MCPC and its variants. In this section, we consider a variation of the Capacitated Facility Location Problem (CFLP). In Aardal et al. (2015), an LP-based $(4.562 + \varepsilon)$-approximation algorithm (for any $\varepsilon > 0$) for the CFLP in which all facilities have equal opening costs is given. We extend this problem to a problem in which there are clusters of facilities that all have a capacity

associated with them. The new problem is called the Capacitated Facility Location Problem with Cluster constraints (CFLPC).

In the CFLPC, we have a set of customers $\mathcal{I}$ that needs to be served by a set of facilities $\mathcal{K}$. Every customer $i \in \mathcal{I}$ has a weight of w_i that needs to be covered by the facilities. A customer can be served by multiple facilities. The costs of assigning the entire weight of customer $i \in \mathcal{I}$ to facility $k \in \mathcal{K}$ is given by c_{ik}. If only a fraction of the weight of customer i is covered by facility k, then only that fraction of c_{ik} needs to be paid. Every facility $k \in \mathcal{K}$ has opening costs f_k and capacity B_k. Furthermore, we are given a set of clusters $\mathcal{C}$, and every cluster $l \in \mathcal{C}$ has an associated capacity U_l. The set of facilities contained in cluster l is given by $\mathcal{K}(l) \subseteq \mathcal{K}$ but contrary to the MCPC, we do not assume the sets $\mathcal{K}(l)$ to be disjoint. Hence, the set $\mathcal{C}(k) \subseteq \mathcal{C}$, which represents the clusters in which knapsack k is contained, could have a cardinality larger than one.

Using the same technique as in Section 8.5, the capacity of the clusters can be distributed over the facilities using the z-variables. Hence, the CFLPC can be formulated as the following ILP.

$$\min \quad L(x,y,z) = \sum_{k\in\mathcal{K}} f_k y_k + \sum_{i\in\mathcal{I}} \sum_{k\in\mathcal{K}} c_{ik} x_{ik} \tag{8.23}$$

subject to

$$\sum_{k\in\mathcal{K}} x_{ik} \geq 1 \qquad \forall i \in \mathcal{I} \tag{8.24}$$

$$x_{ik} \leq y_k \qquad \forall i \in \mathcal{I} \qquad \forall k \in \mathcal{K} \tag{8.25}$$

$$\sum_{i\in\mathcal{I}} w_i x_{ik} \leq B_k \qquad \forall k \in \mathcal{K} \tag{8.26}$$

$$\sum_{i\in\mathcal{I}} w_i x_{ik} \leq \sum_{l\in\mathcal{C}(k)} U_l z_{kl} \qquad \forall k \in \mathcal{K} \tag{8.27}$$

$$\sum_{k\in\mathcal{K}} z_{kl} \leq 1 \qquad \forall l \in \mathcal{C} \tag{8.28}$$

$$x_{ik} \geq 0 \qquad \forall i \in \mathcal{I} \qquad \forall k \in \mathcal{K} \tag{8.29}$$

$$y_k \in \{0,1\} \qquad \forall k \in \mathcal{K} \tag{8.30}$$

$$z_{kl} \geq 0 \qquad \forall k \in \mathcal{K} \qquad \forall l \in \mathcal{C}. \tag{8.31}$$

In the objective function in (8.23), the opening costs of the facilities and the costs to serve the customers are minimized. Constraint (8.24) ensures that the demand of every customer is completely served by the facilities. In constraint (8.25), it is enforced that a customer can only be assigned to an open facility. The total weight of the customers that can be assigned to a facility is restricted by the facility capacity in constraint (8.26), and by the cluster capacity in constraint (8.27). Note that, in constrast to constraint (8.15), there

Algorithm 8.5: Algorithm for the CFLPC.

Solve the LP relaxation of problem (8.23)-(8.31).
Let (x^*, y^*, z^*) be the optimal solution.
Solve the capacitated facility location problem with each facility having a capacity of $\min\{B_k, z^*_{jk} U_l\}$, using the algorithm of Aardal et al. (2015)

are multiple clusters from which a knapsack k can get some cluster capacity in constraint (8.27). The constraint (8.28) ensures that the fraction of the capacity U_l that is assigned to the knapsacks does not exceed 1. The x and z-variables are positive because of, respectively constraints (8.29) and (8.31), and the y-variables are enforced to be binary in constraint (8.30).

If the constraints (8.27), (8.28), and (8.31) from the ILP formulation (8.23)-(8.31), then the ILP formulation for the CFLP is obtained. We use (IP) to refer to the ILP for the CFLP and (LP) for its LP relaxation. Moreover, (IP^C) and (LP^C) are used to denote the ILP (8.23)-(8.31) and its LP relaxation.

The crucial difference between the CFLPC and the MCPC is that in the CFLPC, fractional assignments of customers to facilities are allowed. Only the y-variables are integral in (IP^C). As a consequence, a fractional solution for the x-variables that is feasible with respect to the reduced capacity of a knapsack ($\min\{B_k, \sum_{l \in \mathcal{C}(k)} U_l z^*_{kl}\}$) is a feasible solution for (IP^C). In Aardal et al. (2015), an LP-based $(4.562 + \varepsilon)$-approximation algorithm is given if all facilities have the same opening costs, i.e., $f_k = f$ for all $k \in \mathcal{K}$. In Algorithm 8.5, this approximation algorithm for the CFLPC is used to derive a $(4.562 + \varepsilon)$-approximation algorithm for the CFLPC with fixed opening costs.

Lemma 8.11. *Algorithm 8.5 is a* $(4.562 + \varepsilon)$*-approximation algorithm for the CFLPC if* $f_k = f$ *for all* $k \in \mathcal{K}$.

Proof. Let us start the proof of this lemma by introducing some notation. Let (x^*, y^*, z^*) be the optimal solution of (LP^C). Furthermore, let $(\bar{x}, \bar{y}, z^*)$ be the optimal solution for (LP) with each facility having the reduced capacity $\min\{B_k, \sum_{l \in \mathcal{C}(k)} U_l z^*_{kl}\}$, and let $(\hat{x}, \hat{y}, z^*)$ be the solution produced by the algorithm of Aardal et al. (2015) for the same problem. Hence, the solution $(\hat{x}, \hat{y}, z^*)$ is the solution produced by Algorithm 8.5.

From Aardal et al. (2015), we know that $L(\hat{x}, \hat{y}, z^*) \leq (4.562 + \varepsilon) L(\bar{x}, \bar{y}, z^*)$. As the solution (x^*, y^*, z^*) is the optimal solution for (LP^C), we know that $L(x^*, y^*, z^*) \leq L(\bar{x}, \bar{y}, z^*)$. However, the solution (x^*, y^*, z^*) is also feasible for (LP) if each facility has a reduced capacity of $\min\{B_k, \sum_{l \in \mathcal{C}(k)} U_l z^*_{kl}\}$. Consequently, $L(x^*, y^*, z^*) \geq L(\bar{x}, \bar{y}, z^*)$, and thus it must hold that $L(x^*, y^*, z^*) = L(\bar{x}, \bar{y}, z^*)$.

All in all, we can conclude, if all facilities have the same opening costs, that

$$\begin{aligned} L(\hat{x}, \hat{y}, z^*) &\leq (4.562 + \varepsilon) L(\bar{x}, \bar{y}, z^*) = (4.562 + \varepsilon) L(x^*, y^*, z^*) \\ &\leq (4.562 + \varepsilon) \mathrm{OPT}, \end{aligned}$$

which proves the lemma. □

Note that Algorithm 8.5 does not use any specific details of the algorithm of Aardal et al. (2015) except that it is LP-based. Consequently, any LP-based α-approximation algorithm for the CFLP can be used to give an approximation algorithm for the CFLPC with the same approximation guarantee.

8.8 Conclusion

In the previous chapters, we have seen some problems in which there are two levels of capacities. For these problems, we have only used numerical experiments to evaluate the heuristic. Hence, in this chapter, we have developed approximation algorithms for cluster capacitated problems. We have introduced a decision variable that decides on the fraction of the cluster capacity that is distributed to lower tiers. With that decision variable, we can use (known) LP-based approximation algorithms for problems without cluster constraints for problems with cluster constraints.

For the Capacitated Facility Location Problem with cluster constraints and fixed opening costs and the Multiple Knapsack Problem with clusters that satisfies the isolation property, the approximation ratio remains the same if cluster constraints have been added. For the Capacitated Maximum Coverage Problem and the general case of the Multiple Knapsack problem, the approximation ratio got slightly worse by adding a cluster, respectively, from $\frac{1}{2}\left(1 - \frac{1}{e}\right)$ to $\frac{1}{3}\left(1 - \frac{1}{e}\right)$ and from $\frac{1}{2}$ to $\frac{1}{3}$. Hence, we can draw the tentative conclusion that adding cluster capacities to an optimization problem does not make the problem much harder. In further research, it has to be investigated if this claim indeed also holds for other problems.

In this chapter, we have only introduced a single extra level of capacities, namely the clusters. It would be interesting to develop approximation algorithms for an arbitrary number of levels of capacity. What if the capacity of a cluster is also restricted by the capacity of a super-cluster in which it is contained? A problematic aspect of these problems lies in the distribution of capacity of a super-cluster among clusters. In Lemma 8.5, it is shown that the cluster capacity should be distributed among the largest knapsacks. Nevertheless, it is unclear if it is better to distribute the capacity of a super-cluster to a cluster with a large capacity but many knapsacks with a small capacity or to a cluster with a smaller capacity but more knapsacks with a larger capacity.

We have not proven that our approximation ratios are tight, which means that there might be better performance guarantees possible for the problems presented. For instance, the pipage rounding technique is, in general, used to round a fractional solution to an integral solution. In contrast, we have only rounded it to a solution that satisfies the bounded split property. With a more advanced rounding technique, it might be possible to use the pipage rounding technique to obtain an integral solution. In that situation, we would obtain a $(1 - \frac{1}{e})$-approximation algorithm for the MCPK. Moreover, we have found using iterative rounding a method to improve the $\frac{1}{3}$-approximation algorithm for the MKPC if it satisfies the isolation property to $\frac{1}{2}$. With another line of argumentation, it might be possible to show that this is possible for more general cases.

9
Conclusion

In this dissertation, we have studied operational problems arising in container hinterland transportation. These problems can be partitioned into two categories: (i) the *multimodal transportation* part in which it is decided how a container is transported and (ii) the *terminal operations* part in which the goal is to place the container in the correct position at the terminal. For these problems, we have developed exact algorithms, that give the optimal solution, but also efficient heuristics. The goal of these problems was to obtain cost-efficient and reliable solutions.

All the problems discussed in this dissertation are new in the scientific literature. The reason why these problems have not been discussed before could be that there has been little collaboration between academic researchers and inland container terminals. However, as the number of containers that are transported around the world has increased significantly over the past few decades (recall Figure 1.1 in Chapter 1), efficient hinterland transportation is getting increasingly important. In future research, the exact gain of using the proposed solution methods remains to be investigated. On the one hand, it is expected that the obtained solutions result in lower transportation costs for containers and more reliable transportation plans. On the other hand, the algorithms make the planning process faster. Hence, the planner has more time to focus on the difficult cases and to investigate multiple scenarios.

We have used a variety of different techniques to solve the problems in this dissertation. The optimal algorithms, such as integer linear programming, branch-and-bound and sample average approximation are only applicable in practice if one has hours time to compute a solution. Due to the dynamic environment of a container terminal in which continuously new information arrives, fast solutions are essential. Nevertheless, the optimal solutions are useful as a benchmark for the heuristics. One could try to find faster optimal algorithms that exploit more of the specific problem structure. However, a big improvement has to be made for the algorithms to be fast enough for practice.

The potential gain of improving the heuristics is likely to be bigger. The heuristics presented in this dissertation are rather intuitive, which has the benefit

that the heuristics are no black boxes, and the solutions they provide can be explained. A disadvantage could be that the solution quality of these heuristics might not be the best possible. It is likely that applying metaheuristics, such as simulated annealing, tabu search, and genetic algorithms, would result in better solutions than the current heuristics. However, that comes at the price of longer running times and less explainability.

Most of the problems studied in this dissertation exist because of the lack of information sharing by different partners. The problem studied in Chapter 3 is the most obvious example because the number of containers that could be loaded and unloaded is not communicated. Hence, we treated that number of moves as a stochastic variable. If the number of moves is communicated well in advance, then the deterministic model of Chapter 2 would be sufficient.

However, also the stacking problems studied in Chapters 5, 6, and 7 would be easier, or maybe even non-existent, if the communication between different partners in the transportation improves. First of all, the relocation phase's stochasticity resulted from the unknown retrieval order of the containers in a time interval. As everyone before in the literature, we have assumed that the retrieval order is a random uniform permutation because no information is usually known about which truck will arrive earlier. However, as these trucks have an appointment within the same time interval, they should be in the terminal vicinity. Hence, if the truck that picks up a container shares its location, it would be possible to change the uniform distribution to another distribution that would result in fewer relocation moves.

The second aspect in which information could help is deciding where to place a container the moment it arrives at the terminal. In case the exact moment a container will leave the terminal is known at the time it arrives, it could be possible to position a container such that it is on top of a stack at its departure time. This storage assignment problem is not easy to solve, but it would make the pre-processing phase redundant.

The models in this dissertation could support building a business case for information sharing. The difference in the objective function between when information is uncertain and when it is known is called the *value of information*. In case this value of information is large, it makes sense to invest in an infrastructure to share information. We have seen in Chapter 3 that there was already a big difference between the total costs for the transportation of containers if the number of moves follows a Poisson or geometric distribution. Another example is that one could use the problem of Chapter 4 to calculate how much a shipper is willing to pay to remove the possibility of its shipment being overbooked.

It has to be said that information sharing could also lead to less flexibility.

If everything needs to be planned well in advance, it is no longer possible to use real-time information to adjust the planning. In the end, some factors that influence transportation, such as the weather, are almost inherently stochastic. It is nowadays possible to make quite an accurate weather forecast, but it is still impossible to forecast days in advance, precisely at what time the wind will be too strong for cranes to operate. Therefore, it is crucial to have the right balance in flexibility of the operations and predictability for other parts in the chain.

This dissertation is all but a complete work on all operational problems arising in hinterland transportation. We have merely focused on specific parts of the entire process of getting a container from the deep-sea port to the final destination. Examples of topics that have not been covered are the stowage plan of a barge and the transportation from the inland terminal to the final destination. An important direction of further research would be to use a more *integrated approach*. For instance, consider an example in which x containers need to be delivered to a single customer on the same day. Assume that a single truck can ship in one day at most $x - 1$ containers from the inland terminal to that customer. In that case, it could be wise transport only $x - 1$ containers per barge from the deep-sea terminal to the inland terminal and use a truck to deliver a single container directly from the deep-sea terminal to the customer. Another example in which an integrated approach could be beneficial is the stowage plan. Suppose one considers already the storage assignment of containers at the inland terminal when loading the barge. In that case, it could be possible to obtain a stowage plan that results in fewer movements of containers at the inland terminal.

There are two main problems with a more integrated approach. First, an optimal solution in an integrated approach is usually sub-optimal for a specific part of the transportation chain. Consequently, the benefits of an integrated approach should be divided in a fair way among all participating partners. A crucial aspect here is that all stakeholders trust each other. The second problem is that integrated problems are generally much harder to solve from a computational perspective. For instance, if one combines the stowage and the storage assignment plan in one problem, then the solution space is enormous. Furthermore, there are many dependencies. The position of a container in a barge has a main influence on where it can be positioned at the terminal.

Another direction for further research is the use of *machine learning*. One way machine learning has been applied before is determining the best branching strategy when solving an ILP (see, e.g., Lodi and Zarpellon (2017)). In Chapter 6, we have developed a simple algorithm to estimate the number of remaining relocation moves in a bay. A more sophisticated machine learning algorithm will probably have higher accuracy. An improved prediction algorithm

is also likely to improve the quality of the heuristics of Chapters 6 and 7.

An exciting idea is that an algorithm with higher accuracy for predicting the number of relocation moves does not necessarily lead to better outcomes for the pre-processing phase. We use the method to estimate the number of relocation moves in the heuristics of Chapters 6 and 7 to find the best pre-processing move. Ideally, we do not want a method that gives the highest accuracy for the expected number of relocation moves, but a method that produces estimates which lead to the best pre-processing move. Ban and Rudin (2019) show that for the newsvendor problem, better solutions can be obtained if an integrated approach is used in which the demand estimation and the decision on the inventory are combined.

Machine learning can also be used to find the difference between a human planner's solution and the solution produced by an algorithm. If both these solutions are encoded as a solution from an ILP, then one could build a decision tree to find the difference between the two solutions. The resulting problem is then a classification problem. For example, consider the problem of Chapter 2. Suppose the solution from the algorithm often visits three specific terminals with a barge. However, the transportation plan made by the planner never visits these three terminals with the same barge. In that case, that feature distinguishes the optimal solution from the human solution.

When implementing an algorithm in practice, it could be beneficial to know the difference between the current solution and the new algorithm for two reasons. First of all, it might be that a specific constraint is missing in the new model. Second, if the new solution is very different from the solution that the planner would make, it is less likely that he or she will accept the solution produced by the algorithm. If one adds specific constraints to the algorithm such that the solution looks more like the solution that the planner would make, then the acceptance will be higher. Afterward, it is possible to gradually remove these constraints to get closer to the optimal solution.

All in all, this dissertation shows that it is possible to balance the costs and reliability of hinterland transportation. This research can be seen as a step toward obtaining a modal shift from trucks towards barges and trains. However, many operational challenges remain to be solved.

Bibliography

K. Aardal, P.L van den Berg, D. Gijswijt, and S. Li. Approximation algorithms for hard capacitated k-facility location problems. *European Journal of Operational Research*, 242:358–368, 2015.

A.A. Ageev and M.I. Sviridenko. Pipage rounding: a new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*, 8(3):307–328, 2004.

M.H. Akyüz and C.-Y. Lee. A mathematical formulation and efficient heuristics for the dynamic container relocation problem. *Naval Research Logistics*, 61(2):101–118, 2014.

K. N. Androutsopoulos and K. G. Zografos. Solving the multi-criteria time-dependent routing and scheduling problem in a multimodal fixed scheduled network. *European Journal of Operational Research*, 192:18–28, 2009.

A. Badanidiyuru and J. Vondrák. Fast algorithms for maximizing submodular functions. In *Proceedings of the 2014 Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1497–1514, 2014.

G.-Y. Ban and C. Rudin. The big data newsvendor: Practical insights from machine learning. *Operations Research*, 67(1):90–108, 2019.

N. Bansal, N. Korula, V. Nagarajan, and A. Srinivasan. Solving packing integer programs via randomized rounding with alternations. *Theory of Computing*, 8:533–565, 2012.

A. Baykasoglu and K. Subulan. A multi-objective sustainable load planning model for intermodal transportation networks with a real-life application. *Transportation Research Part E*, 95(207-247), 2016.

B. Behdani, Y. Fan, B. Wiegmans, and R. Zuidwijk. Multimodal schedule design for synchromodal freight transport systems. *European Journal of Transport and Infrastructure Research*, 16(3):424–444, 2016.

J. Benders and J. van Nunen. A property of assignment type mixed integer linear programming problems. *Operations Research Letters*, 2(2):47–52, 1983.

D. M. Bernhofen, Z. El-Sahli, and R. Kneller. Estimating the effects of the container revolution on world trade. *Journal of International Economics*, 98:36–50, 2016.

J.R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer Series in Operations Research and Financial Engineering. Springer, 2011.

E. den Boer, M. Otten, and H. van Essen. Comparison of various transport modes on a EU scale with the STREAM database. Technical report, CE Delft, 2011.

A. Bortfeld and F. Forster. A tree search procedure for the container pre-marshalling problem. *European Journal of Operational Research*, 217:531–540, 2012.

A. Caris and G. Janssens. Container drayage operations at intermodal terminals: a deterministic annealing approach. In W. Wang and G. Wets, editors, *Computational Intelligence for Traffic and Mobility*, volume 8. Atlantis Press, 2013.

A. Caris, C. Macharis, and G.K. Janssens. Corridor network design in hinterland transportation systems. *Flexible Services and Manufacturing Journal*, 24:294–319, 2012.

A. Caris, S. Limbourg, C. Macharis, T. van Lier, and M. Cools. Integration of inland waterway transport in the intermodal supply chain: a taxonomy of research challenges. *Journal of Transport Geography*, 41:126–136, 2014.

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

M. Caserta, S. Voß, and M. Sniedovich. Applying the corridor method to a blocks relocation problem. *OR Spectrum*, 33(4):915–929, 2011.

M. Caserta, S. Schwarze, and S. Voß. A mathematical formulation and complexity considerations for the blocks relocation problem. *European Journal of Operational Research*, 219:96–104, 2012.

B. Casey and E. Kozan. Optimising container storage processes at multimodal terminals. *The Journal of the Operational Research Society*, 63(8):1126–1142, 2012.

E. Chang, E. Floros, and A. Ziliaskopoulos. An intermodal time-dependent minimum cost path algorithm. In V. Zeimpekis, C. D. Tarantilis, G.M. Giaglis, and I. Minis, editors, *Dynamic Fleet Management*, volume 38 of *Operations Research/Computer Science Interfaces Series*. Springer, 2007.

T. Chang. Best routes selection in international intermodal networks. *Computers & Operations Research*, 35:2877–2891, 2008.

T. Chang, L.K. Nozick, and M.A. Turnquist. Multiobjective path finding in stochastic dynamic networks, with application to routing hazardous materials shipments. *Transportation Science*, 39(3):383–399, 2005.

C. Chekuri and S. Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 35:719–728, 2005.

C. Chekuri and A. Kumar. Maximum coverage problem with group budget constraints and applications. In K. Jansen, S. Khanna, J. D. P. Rolim, and D. Ron, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 72–83. Springer, 2004.

C. Chekuri, J. Vondrák, and R. Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. *SIAM Journal on Computing*, 43(6):1831–1879, 2014.

J. Cheng and A. Lisser. A second-order cone programming approach for linear programs with joint probabilistic constraints. *Operations Research Letters*, 40:325–

328, 2012.

J.H. Cho, H.S. Kim, and H.R. Choi. An intermodal transport network planning algorithm using dynamic programming—a case study: from Busan to Rotterdam in intermodal freight routing. *Applied Intelligence*, 36:529–541, 2012.

R. Cominetti and A. Torrico. Additive consistency of risk measures and its application to risk-averse routing in networks. *Mathematics of Operations Research*, 41(4): 1510–1521, 2016.

T.G. Crainic and K.H. Kim. Intermodal transportation. In C. Barnhart and G. Laporte, editors, *Handbook in Operations Research and Management Science*, volume 14, chapter 8, pages 467–537. Elsevier, 2007.

T.G. Crainic and G. Laporte. Planning models for freight transportation. *European Journal of Operational Research*, 97:409–438, 1997.

C. Dong, R. Boute, A. McKinnon, and M. Verelst. Investigating synchromodality from a supply chain perspective. *Transportation Research Part D*, 61:42–57, 2018.

K. Dudzinski and S. Walukiewicz. Exact methods for the knapsack problem and its generalizations. *European Journal of Operational Research*, 28:3–21, 1987.

A. Ene and H. L. Nguyen. A nearly-linear time algorithm for submodular maximization with a knapsack constraint. In C. Baier, I. Chatzigiannakis, P. Flocchini, and S. Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming*, volume 132 of *Leibniz International Proceedings in Informatics*, pages 53:1–53:12, 2019.

European Commission. Roadmap to a single European transport area, 2011.

Eurostat. Modal split of freight transport. `ec.europa.eu/eurostat/web/products-datasets/product?code=t2020_rk320`, 2020. Accessed: 15 July 2020.

C. Expósito-Izquierdo, B. Melián-Batista, and M. Moreno-Vega. Pre-marshalling problem: Heuristic solution method and instances generator. *Expert Systems with Applications*, 39:8337–8349, 2012.

Y. Fairstein, A. Kulik, J. Naor, D. Raz, and H. Shachnai. A $(1-e^{-1}-\epsilon)$-approximation for the monotone submodular multiple knapsack problem. *arXiv*, 2020. 2004.12224.

B. Farbstein and A. Levin. Maximum coverage problem with group budget constraints. *Journal of Combinatorial Optimization*, 34:725–735, 2017.

S. Fazi, J. Fansoo, and T. van Woensel. A decision support system tool for the transportation by barge of import containers: a case study. *Decision Support Systems*, 79:33–45, 2015.

U. Feige. A threshold of ln *n* for approximating set cover. *Journal of the ACM*, 45 (4):634–652, 1998.

D. Feillet, S.N. Parragh, and F. Tricoire. A local-search based heuristic for the unrestricted block relocation problem. *Computers & Operations Research*, 108:44–56, 2019.

M. Fisher, R. Jaikumar, and L. van Wassenhove. A multiplier adjustment method for

the generalized assignment problem. *Management Science*, 3246(9):1095–1103, 1986.

L. Fu and L.R. Rilett. Expected shortest path in dynamic and stochastic traffic networks. *Transportation Research Part B*, 32(7):499–516, 1998.

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

V. Galle, V.H. Manshadi, S. Borjian Boroujeni, C. Barnhart, and P. Jaillet. The stochastic container relocation problem. *Transportation Science*, 52(5):1035–1058, 2018b.

M. X. Goemans and D. P. Williamson. New $\frac{3}{4}$-approximation algorithms for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7(4): 656–11, 1994.

J.A.S. Gromicho, E. Oudshoorn, and G. Post. Generating price-effective intermodal routes. *Statistica Neerlandica*, 65(4):432–445, 2011.

L. Guo, M. Li, and D. Xu. Efficient approximation algorithms for maximum coverage with group budget constraints. *Theoretical Computer Science*, 788:53–65, 2019.

R.W. Hall. The fastest path through a network with random time-dependent travel times. *Transportation Science*, 20(3):182–188, 1986.

D.R. Headrick. *Technology: a world history*. Oxford University Press, second edition, 2009.

H. Heggen, Y. Molenbruch, A. Caris, and K. Braekers. Intermodal container routing: integrating long-haul routing and local drayage decisions. *Sustainability*, 11(6): 1–36, 2019.

A. Hottung and K. Tierney. A biased random-key genetic algorithm for the container pre-marshalling problem. *Computers & Operations Research*, 75:83–102, 2016.

A. Hottung, S. Tanaka, and K. Tierney. Deep learning assisted heuristic tree search for the container pre-marshalling problem. *Computers & Operations Research*, 113-104781, 2020.

M. Hussein and M. E. H. Petering. Genetic algorithm-based simulation optimization of stacking algorithms for yard cranes to reduce fuel consumption at seaport container transshipment terminals. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–8, 2012.

K.R. Hutson and D.R. Shier. Extended dominance and a stochastic shortest path problem. *Computers & Operations Research*, 36:584–596, 2009.

L. Häme and H. Hakula. Dynamic journeying under uncertainty. *European Journal of Operational Research*, 225:455–471, 2013.

F. Iannone. A model optimizing the port-hinterland logistics of containers: the case of Campania region in Southern Italy. *Maritime Economics & Logistics*, 14(1):33–72, 2012.

P. Jaillet, J. Qi, and M. Sim. Routing optimization under uncertainty. *Operations Research*, 64(1):186–200, 2016.

M. Ji, W. Guo, H. Zhu, and Y. Yang. Optimization of loading sequence and rehandling strategy for multi-quay crane operations in container terminals. *Transportation Research Part E*, 80:1–19, 2015.

B. Jin, W. Zhu, and A. Lim. Solving the container relocation problem by an improved greedy look-ahead heuristic. *European Journal of Operational Research*, 240:837–847, 2015.

R. Jovanovic, M. Tuba, and S. Voß. A multi-heuristic approach for solving the pre-marshalling problem. *Central European Journal of Operations Research*, 25(1): 1–28, 2017.

R. Jovanovic, S. Tanaka, T. Nishi, and S. Voß. A GRASP approach for solving the blocks relocation problem with stowage plan. *Flexible Services and Manufacturing Journal*, 31:702–729, 2019a.

R. Jovanovic, M. Tuba, and S. Voß. An efficient ant colony optimization algorithm for the blocks relocation problem. *European Journal of Operational Research*, 274: 78–90, 2019b.

M. Karlaftis, K. Kepaptsoglou, and E. Sambracos. Containership routing with time deadlines and simultaneous deliveries and pick-ups. *Transportation Research Part E*, 45:210–221, 2009.

H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

L.G. Khachiyan. The problem of calculating the volume of a polyhedron is enumerably hard. *Russian Mathematical Surveys*, 44(3):199–200, 1989.

S. Khuller, A. Moss, and J. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70:39–45, 1999.

K.H. Kim and H.-O. Günther. Container terminals and terminal operations. In *Container Terminals and Cargo Systems*, chapter 1, pages 3–14. Springer, 2007.

K.H. Kim and G. Hong. A heuristic rule for relocating blocks. *Computers & Operations Research*, 33(4):940–954, 2006.

S. Kim, R. Pasupathy, and S. G. Henderson. A guide to Sample Average Approximation. In M.C. Fu, editor, *Handbook of Simulation Optimization*. Springer, 2015.

A.J. Kleywegt, A. Shapiro, and T. Homem De Mello. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, 12(2):479–502, 2001.

W. Klibi, F. Lasalle, A. Martel, and S. Ichoua. The stochastic multiperiod location transportation problem. *Transportation Science*, 44(2):221–237, 2010.

P. Krugman. Citigroup foundation special lecture. In *Comparative Advantage, Growth, and the Gains from Trade and Globalization: a Festschrift in Honor of Alan V Deardorff*, pages 5–15. World Scientific Publishing Company, 2011.

D. Ku and T.S. Arthanhari. Container relocation problem with time windows for

container departure. *European Journal of Operational Research*, 252:1031–1039, 2016.

A. Kulik, H. Shachnai, and T. Tamir. Maximizing submodular set functions subject to multiple linear constraints. *Mathematics of Operations Research*, 38(4):545–554, 2013.

C.-Y. Lee and D.-P. Song. Ocean container transport in global supply chains: Overview and research opportunities. *Transportation Research Part B*, 95:442–474, 2017.

J. Lee, V.S. Mirrokni, V. Nagarajan, and M. Sviridenko. Maximizing nonmonotone submodular functions under matroid or knapsack constraints. *SIAM Journal on Discrete Mathematics*, 23(4):2053–2078, 2010.

Y. Lee and N.-Y. Hsu. An optimization model for the container pre-marshalling problem. *Computers & Operations Research*, 34(11):3295–3313, 2007.

Y. Lee and Y.-J. Lee. A heuristic for retrieving containers from a yard. *Computers & Operations Research*, 37:1139–1147, 2010.

M. Levinson. *The Box: How the shipping container made the world smaller and the world economy bigger*. Princeton University Press, second edition, 2016.

S. Li, R. Negenborn, and G. Lodewijks. Distributed constraint optimization for addressing vessel rotation planning problems. *Engineering Applications of Artificial Intelligence*, 48:159–172, 2016.

D.-Y. Lin, Y.-L. Lee, and Y. Lee. The container retrieval problem with respect to relocation. *Transportation Research Part C*, 52:132–143, 2015.

A. Lodi and G. Zarpellon. On learning and branching: a survey. *TOP*, 25:207–236, 2017.

Y. Long, L.H. Lee, and E.P. Chew. The sample average approximation method for empty container repositioning with uncertainties. *European Journal of Operational Research*, 222:65–75, 2012.

L. Lozano and A.L. Medaglia. On an exact method for the constrained shortest path problem. *Computers & Operations Research*, 40:378–384, 2013.

J. Luedtke, S. Ahmed, and G.L. Nemhauser. An integer programming approach for linear programs with probabilistic constraints. *Mathematical Programming Series A*, 122:247–272, 2010.

W. Mak, D.P. Morton, and R.K. Wood. Monte Carlo bounding techniques for determining solution quality in stochastic programs. *Operations Research Letters*, 24: 47–56, 1999.

H. Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

M de Melo da Silva, S. Toulouse, and R. Wolfler Calvo. A new effective unified model for solving the pre-marshalling and block relocation problems. *European Journal of Operational Research*, 271:40–56, 2018.

M. Mes and M. Iacob. Synchromodal transport planning at a logistics service provider.

In H. Zijm, M. Klumpp, U. Claussen, and M. ten Hompel, editors, *Logistics and Supply Chain Innovation: Bridging the Gap between Theory and Practice*. Springer, 2016.

E.D. Miller-Hooks and H.S. Mahmassani. Least expected time paths in stochastic, time-varying transportation networks. *Transportation Science*, 34(2):198–215, 2000.

G.L. Nemhauser and L.A. Wolsey. An analysis of approximations for maximizing submodular set functions-I. *Mathematical Programming*, 14:265–294, 1978.

Y. Nie, X. Wu, and T. Honem de Mello. Optimal path problems with second-order stochastic dominance constraints. *Networks and Spatial Economics*, 12:561–587, 2012.

K. Nip and Z. Wang. Approximation algorithms for a two-phase knapsack problem. In *International computing and combinatorics conference*, volume 10976, of *Lecture Notes in Computer Science*, pages 63–75, 2018.

C. Parreño-Torres, R. Alvarez-Valdes, and R. Ruiz. Integer programming models for the pre-marshalling problem. *European Journal of Operational Research*, 274: 142–154, 2019.

A. Pérez Rivera and M. Mes. Service and transfer selection for freights in a synchromodal network. In A. Paias, M. Ruthmair, and S. Voß, editors, *Computational Logistics*, volume 9855 of *Lecture Notes in Computer Science*. Springer, 2016.

A. Pérez Rivera and M. Mes. Anticipatory freight selection in intermodal long-haul round-trips. *Transportation Research Part E*, 105:176–194, 2017.

M.E.H. Petering and M.I. Hussein. A new mixed integer program and extended look-ahead heuristic algorithm for the block relocation problem. *European Journal of Operational Research*, 231:120–130, 2013.

Port of Rotterdam Authority. Port vision 2030: Port compass, 2011.

Port of Rotterdam Authority. Facts and figures, 2020a.

Port of Rotterdam Authority. Barge performance monitor. `www.portofrotterdam.com/en/doing-business/logistics/connections/barge-performance-monitor`, 2020b. Accessed: 5 August 2020.

L. Di Puglia Pugliese and F. Guerriero. A survey of resource constrained shortest path problems: Exact solution approaches. *Networks*, 62(3):183–200, 2013.

A. Rendl and M. Prandtstetter. Constraint models for the container pre-marshalling problem. In G. Katsirelos and C.-G. Quimper, editors, *ModRef 2013: 12th International Workshop on Constraint Modelling and Reformulation*, pages 44–56, 2013.

B. van Riessen, R. Negenborn, and R. Dekker. Synchromodal container transportation: an overview of current topics and research opportunities. In F. Corman, S. Voß, and R. Negenborn, editors, *Computational Logistics*, volume 9335 of *Lecture Notes in Computer Science*. Springer, 2015.

B. van Riessen, R. Negenborn, and R. Dekker. Real-time container transport planning

with decision trees based on offline obtained optimal solutions. *Decision Support Systems*, 89:1–16, 2016.

J. Scholl, D. Boywitz, and N. Boysen. On the quality of simple measures predicting block relocations in container yards. *International Journal of Production Research*, 56(1-2):60–71, 2018.

G. Schäfer and B.G. Zweers. Maximum coverage with cluster constraints: an LP-based approximation technique. In *Proceedings of Workshop on Approximation and Online Algorithms*, Lecture Notes in Computer Science, 2020. To appear.

A. Shapiro, D. Dentcheva, and A. Ruszczyński. *Lectures on Stochastic Programming*. SIAM, 2009.

K. Sharyapova. *Optimization of Hinterland Intermodal Container Transportation*. PhD thesis, Eindhoven University of Technology, 2014.

D.B. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming A*, 62:461–474, 1993.

A. da Silva Firmino, R.M. de Abreu Silva, and V.C. Times. A reactive grasp metaheuristic for the container retrieval problem to reduce crane's working time. *Journal of Heuristics*, 25(2):141–173, 2019.

R. Stahlbock and S. Voß. Operations research at container terminals: a literature update. *OR Spectrum*, 30(1):1–52, 2008.

Statista. International seaborne trade carried by container ships from 1980 to 2017. `www.statista.com/statistics/253987/international-seaborne-trade-carried-by-containers/`, 2020. Accessed: 14 July 2020.

M. SteadieSeifi, N. Dellaert, W. Nuijten, T. van Woensel, and R. Raoufi. Multimodal freight transportation planning: a literature review. *European Journal of Operational Research*, 233(1):1–15, 2014.

D. Steenken, S. Voß, and R. Stahlbock. Container terminal operation and operations research - a classification and literature review. *OR Spectrum*, 26(1):3–49, 2004.

X. Sun, J. Zhang, and Z. Zhang. Deterministic algorithms for the submodular multiple knapsack problem. *arXiv*, 2020. 2003.11450.

M. Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32:41–43, 2004.

S. Tanaka and F. Mizuno. An exact algorithm for the unrestricted block relocation problem. *Computers & Operations Research*, 95:12–31, 2018.

S. Tanaka and K. Tierney. Solving real-world sized container pre-marshalling problems with an iterative deepening branch-and-bound algorithm. *European Journal of Operational Research*, 264:165–180, 2018.

S. Tanaka, K. Tierney, C. Parreño-Torres, and R. Alvarez-Valdes. A branch and bound approach for large pre-marshalling problems. *European Journal of Operational Research*, 278:211–225, 2019.

K. Tierney and S. Voß. Solving the robust container pre-marshalling problem. In

A. Paias, M. Ruthmair, and S. Voß, editors, *Computational Logistics*, volume 9588 of *Lecture Notes in Computer Science*, pages 131–145. Springer, 2016.

K. Tierney, S. Voß, and R. Stahlbock. A mathematical model of inter-terminal transportation. *European Journal of Operational Research*, 235:448–460, 2014.

K. Tierney, D. Pacino, and S. Voß. Solving the pre-marshalling problem to optimality with A* and IDA*. *Flexible Service and Manufacturing Journal*, 29:223–259, 2017.

B. Toktas, J.W. Yen, and Z.B. Zabinsky. Addressing capacity uncertainty in resource-constrained assignment problem. *Computers & Operations Research*, 33:724–745, 2006.

F. Tricoire, J. Scagnetti, and A. Beham. New insights on the block relocation problem. *Computers & Operations Research*, 89:127–139, 2018.

B. Verweij, S. Ahmed, A.J. Kleywegt, G. Nemhauser, and A. Shapiro. The sample average approximation method applied to stochastic routing problems: a computational study. *Computational Optimization and Applications*, 24(2-3):289–333, 2003.

I.F.A. Vis and R. de Koster. Transshipment of containers at a container terminal: an overview. *European Journal of Operational Research*, 147:1–16, 2003.

J. Vondrák. Symmetry and approximability of submodular maximization problems. *SIAM Journal on Computing*, 42(1):265–304, 2013.

S. Voß and S. Schwarze. A note on alternative objectives for the blocks relocation problem. In C. Paternina-Arboleda and S. Voß, editors, *Computational Logistics*, volume 11756 of *Lecture Notes in Computer Science*. Springer, 2019.

Y. Wang, I. Bilegan, and T. Crainic. A revenue management approach for network capacity allocation of an intermodal barge transportation system. In A. Paias, M. Ruthmair, and S. Voß, editors, *Computational Logistics*, volume 9588 of *Lecture Notes in Computer Science*. Springer, 2016.

D.P. Williamson and D.B. Shmoys. *The design of approximation algorithms*. Cambridge University Press, 2010.

World Shipping Counsel. Top 50 world container ports. `www.worldshipping.org/about-the-industry/global-trade/top-50-world-container-ports`, 2020. Accessed: 14 July 2020.

K.C. Wu and C.J. Ting. A beam search algorithm for minimizing reshuffle operations at container yards. In *Proceedings of the 2010 International Conference on Logistics and Maritime Systems*, 2012.

E.C. Xavier and F.K. Miyazawa. Approximation schemes for knapsack problems with shelf divisions. *Theoretical Computer Science*, 352:71–84, 2006.

X. Yang, J.M.W. Low, and L.C. Tang. Analysis of intermodal freight from China to Indian ocean: a goal programming approach. *Journal of Transport Geography*, 19: 515–527, 2011.

E. Zehender, M. Caserta, D. Feillet, S. Schwarze, and S. Voß. An improved mathematical formulation for the blocks relocation problem. *European Journal of Operational*

Research, 245:415–422, 2015.

H. Zhang, C.-Y. Lee, and T. Li. The value of specific cargo information for substitutable modes of inland transport. *Transportation Research Part E*, 85:23–39, 2016.

L. Zhang and T. Homen-de-Mello. An optimal path model for the risk-averse traveler. *Transportation Science*, 51(2):518–535, 2017.

Y. Zhang and J. Tang. Itinerary planning with time budget for risk-averse travelers. *European Journal of Operational Research*, 267:288–303, 2018.

W. Zhao and A.V. Goodchild. The impact of truck arrival information on container terminal rehandling. *Transportation Research Part E*, 46:327–343, 2010.

A. Ziliaskopoulos and W. Wardell. An intermodal optimum path algorithm for multimodal networks with dynamic arc travel times and switching delays. *European Journal of Operational Research*, 125(3):486–502, 2000.

R. Zuidwijk and Albert Veenstra. The value of information in container transport. *Transportation Science*, 49(3):675–685, 2015.

B.G. Zweers and R.D. van der Mei. Minimum costs paths in intermodal transportation networks with stochastic travel times and overbookings. *Submitted*, 2020.

B.G. Zweers, S. Bhulai, and R.D. van der Mei. Optimizing barge utilization in hinterland container transportation. *Naval Research Logistics*, 66:253–271, 2019.

B.G. Zweers, S. Bhulai, and R.D. van der Mei. Optimizing pre-processing and relocation moves in the stochastic container relocation problem. *European Journal of Operational Research*, 283:954–971, 2020a.

B.G. Zweers, S. Bhulai, and R.D. van der Mei. Planning hinterland container transportation in congested deep-sea terminals. *Flexible Services and Manufacturing Journal*, 2020b. doi: 10.1007/s10696-020-09387-3.

B.G. Zweers, S. Bhulai, and R.D. van der Mei. Pre-processing a container yard under limited available time. *Computers & Operations Research*, 123-105045, 2020c.

Summary

Since the introduction of container transportation in the 1960s, the number of containers being transported worldwide has grown enormously. Moreover, the size of the vessels on which the containers are transported is increasing. Nowadays, the largest deep-sea vessels can carry more than 20,000 TEU, a standard size for a container. Consequently, a large number of containers are transhipped via deep-sea ports. For instance, almost 25,000 containers were being loaded and unloaded every day in the port of Rotterdam in 2019. The transshipment of that many containers is only possible if the containers are efficiently delivered to and picked up from the deep-sea port. This inland transportation, also called *hinterland transportation*, is the focus of this dissertation.

For hinterland transportation three modalities are available: train, barge, and truck. If containers are shipped by a truck, they can be delivered directly to their final destination. Since most companies do not have a rail or water connection, *inland terminals* are essential for the use of trains and barges. The barges and trains bring containers from the deep-sea port to the inland terminal, at which the containers are temporarily stored. Afterward, a truck delivers the container to its final destination. The use of trains and barges has many advantages over transportation by truck. First of all, it is considerably cheaper, and second, the CO_2-emission is much lower. A final advantage is that the use of fewer trucks could lead to a reduction in traffic jams. Therefore, the European Commission aims for a modal shift from trucks to barges and trains.

To utilize the advantages of barge and train transportation, several *operational planning problems* need to be solved. The problems studied in this dissertation are based on challenges faced by an inland container terminal in the port of Amsterdam. However, the methods that are developed are generally applicable. First of all, containers must be on the 'best possible' barge. To determine which barge is optimal, not only the transportation costs are taken into account but also other factors, such as the delay probability or the possibility that a barge cannot be unloaded. These types of problems are discussed in Chapters 2, 3, and 4. An efficient transshipment of containers at an inland terminal is a second aspect that is important to transport more containers per barge and train. Ideally, a container is moved as little as possible, and if it is moved, then only when the workload at the terminal is low. In Chapters 5, 6, and 7, these types of problems are solved.

In this dissertation, these two types of problems are studied from the perspective of an inland terminal. We use the same approach for every problem. First of all, the problem is modeled as a *mathematical optimization problem*. Afterward, an *exact algorithm* is developed that produces the optimal solution. However, calculating the optimal solution often takes too long to apply these algorithms in practice. Therefore, *heuristics* that produce fast solutions that are close to the optimal solution are also developed.

In Chapter 2, the best plan for the transportation of a set of containers located at multiple deep-sea terminals to the inland terminal is determined. The objective of this problem is to ship as many containers as possible per barge. However, at the same time, the barge must not visit too many terminals because that increases the probability of a delay for the barge. A third important aspect that is taken into account are the storage costs at both the deep-sea and inland terminal. This problem is modeled as an *integer linear optimization problem*. When using commercial solvers for this formulation, the optimal solution is obtained, but the running time can be more than a few hours for larger instances. We show that when this formulation is solved in two steps the running time is only a few seconds. The first step determines which terminals are visited, and which containers are shipped on which barge is decided in the second step. The solutions of this are nearly optimal. Finally, a heuristic is developed that simulates the behavior of a human planner. With that algorithm, we show that there are strong improvements possible by implementing the other two algorithms.

The problem of Chapter 3 is similar to that of Chapter 2, but there are two main differences. First, in Chapter 3 containers are transported both to and from the deep-sea terminal. Second, the number of containers that can be loaded and unloaded is unknown at the moment when the planning is done in Chapter 3. As a result, it might be that more containers are loaded on the barge than can be unloaded at the terminal. We treat the number of containers that can be loaded and unloaded at a single terminal as a stochastic variable. Afterward, the problem is modeled as a *stochastic problem with recourse*. Subsequently, this problem is solved using a technique called *Sample Average Approximation*. This method converges to the optimal solution if sufficiently many samples of the stochastic variable are used. However, it is also possible to generate faster solutions. We develop a heuristic in which the original problem is simplified, such that the optimal solution can be calculated using standard techniques of *stochastic programming*. This optimal solution can then be used to replace the stochastic variable by a deterministic value. This heuristic produces solutions that are better than those of other heuristics in which a deterministic value replaces the stochastic variables.

In Chapter 4, the best transportation plan is not determined for a set of

containers but a single container. This container is transported through a network in which the travel times are stochastic. Moreover, there is a probability that a leg is overbooked. The goal of the problem is to find the cheapest route for which the shipment arrives at the final destination before a specific deadline. Each route has a certain on-time arrival probability because of the stochasticity. Since determining an acceptable on-time arrival probability beforehand is hard, *Pareto-optimal* solutions are constructed. In these solutions, the costs of a route are compared with the probability of arriving before the deadline. Besides an optimal algorithm based on *dynamic programming*, we also give a heuristic in which a *risk measure* replaces all stochastic variables. This risk measure is a deterministic value that is used in an integer linear optimization problem. By varying the risk acceptance in the risk measure, this heuristic can also be used to construct Pareto-optimal solutions.

In Chapter 5, we change our focus to stacking problems for containers at terminals. When a container needs to leave the terminal, it frequently occurs that other containers are stacked on top of it. These containers need to be relocated to other stacks while a truck is waiting at the terminal. These moves are called *relocation moves*. An alternative is that containers are already positioned in the right order when it is less busy at the terminal, known as *pre-marshalling moves*. The problem of pre-marshalling moves is that more moves are needed than relocation moves. In this chapter, we introduce a new type of movement, namely the *pre-processing moves*. These moves are performed when the terminal equipment is idle, but in contrast to the pre-marshalling moves, the containers do not need to be positioned entirely in the right order. Consequently, fewer pre-processing moves are performed than pre-marshalling moves, and the moment the pre-processing moves are performed is better than for the relocation moves. We formulate two new problems in this chapter, and in Chapters 6 and 7, solution methods are presented for these problems.

In the problem studied in Chapter 6, the weighted sum of the number of pre-processing and relocation moves is minimized. We derive a heuristic for this problem in which the first containers to be positioned correctly are those that leave the terminal the latest. We need to calculate the expected number of relocation moves for the resulting bay to know if these moves result in an improvement. In this chapter, a method is developed that uses a few decision rules to determine the expected number of relocation moves. The optimal solution is calculated using a *branch-and-bound algorithm*, but for problem instances consisting of many containers, this method needs a couple of hours.

In Chapter 7, the number of pre-processing moves that can be performed is limited. The optimal algorithm of the previous chapter can be used for this problem with a few adjustments. Nevertheless, in this chapter, we use the branch-and-bound method also in a heuristic in which the remaining relocation

moves are estimated. Another heuristic presented in this chapter tries to position every stack's top container in a correct position. Eight different ways in which this can be done are derived, and the one resulting in the largest decrease in the objective function is chosen. In the problem studied in Chapter 6, we assume the crane that handles the containers to move during the pre-processing phase. Nevertheless, in this chapter we relax this assumption. The solutions for the original problem can be used in an integer linear optimization problem to determine how many moves need to be performed in which part of the terminal.

Finally, we study problems from a different perspective in Chapter 8. In the previous chapters, numerical experiments are used to evaluate the quality of heuristics. In contrast, in this chapter, we derive algorithms for which it can theoretically be proven that a given factor bounds the difference between their solution and the optimal solution. We develop these so-called *approximation algorithms* for optimization problems with two levels of capacity. Well-known problems with a single tier of capacity are the *Multiple Knapsack Problem*, the *Maximum Coverage Problem with Knapsack Constraint*, and the *Capacitated Facility Location Problem*. We extend these problems by partitioning the knapsacks in clusters, which also impose a capacity constraint. We introduce a decision variable that determines how much capacity of the cluster is dedicated to which knapsack. This decision variable can be applied to extend existing approximation algorithms based on linear programming. For specific problems, the approximation algorithm's performance guarantee remains the same for the extensions with clusters, and for other problems, it only becomes slightly worse.

Samenvatting

Sinds het begin van het containervervoer in de jaren zestig van de vorige eeuw is het aantal containers dat wereldwijd vervoerd wordt enorm gegroeid. Om dit stijgende aantal containers te vervoeren worden ook de schepen steeds groter: de grootste schepen ter wereld hebben vandaag de dag een capaciteit van meer dan 20,000 TEU, een standaardmaat voor een container. Dit heeft ook tot gevolg dat een groot aantal containers wordt overgeslagen in een zeehaven. Zo werden er in 2019, bijvoorbeeld, bijna 25.000 containers per dag geladen en gelost in de haven van Rotterdam. Deze aantallen zijn alleen mogelijk als de containers efficiënt worden aangeleverd en opgehaald. Dit binnenlandse vervoer van containers is de focus van dit proefschrit.

Voor binnenlands vervoer bestaan drie mogelijkheden: treinen, binnenvaartschepen en vrachtwagens. Als containers met een vrachtwagen worden vervoerd kunnen ze direct naar de eindbestemming worden gebracht. Aangezien de meeste bedrijven geen spoor- of waterwegverbinding hebben, worden binnenlandse containerterminals gebruikt voor schepen en treinen. De binnenvaartschepen en treinen brengen containers vanuit de zeehaven naar de binnenlandse terminal en op deze terminals worden de containers tijdelijk opgeslagen. Daarna worden ze met de vrachtwagen naar hun eindbestemming vervoerd. Het gebruik van treinen en binnenvaartschepen heeft veel voordelen ten opzichte van vervoer met vrachtwagens. Allereerst is het goedkoper en ten tweede levert het ook minder CO_2-uitstoot op. Een laatste voordeel is dat de capaciteit op vaar- en spoorwegen vaak minder schaars is dan op autowegen. Hierdoor kan een afname van het gebruik van vrachtwagens leiden tot minder files. Daarom heeft de Europese Commissie de ambitie uitgesproken om minder containers per vrachtwagen te vervoeren en meer per binnenvaartschip en trein.

Om de voordelen van vervoer via binnenvaartschepen en treinen ten volle te benutten is, zijn er verschillende operationele planningsvraagstukken die opgelost moeten worden. De problemen die bestudeerd worden in dit proefschrift zijn gebaseerd op uitdagingen die een binnenlandse containerterminal in de haven van Amsterdam heeft. Echter zijn de methoden die ontwikkeld worden algemeen toepasbaar. Voor een effectief gebruik van binnenvaartschepen is het ten eerste noodzakelijk dat de containers op het 'best mogelijke schip' worden vervoerd. Om te bepalen wat het beste schip is, worden niet alleen de kosten van transport meegenomen, maar ook andere zaken zoals de kans op vertraging of

de mogelijkheid dat een schip niet gelost kan worden. Dit type probleem wordt behandeld in Hoofdstukken 2, 3 en 4. Een efficiënte overslag van containers op de binnenlandse terminal is een tweede aspect dat belangrijk is om meer containers per vaar- en spoorwegen te vervoeren. Idealiter wordt een container zo min mogelijk verplaatst en als dat dan toch moet gebeuren, dan op een rustig moment. In Hoofdstukken 5, 6 en 7 worden dit soort problemen opgelost.

In dit proefschrift worden deze twee type problemen bestudeerd vanuit het perspectief van een binnenlandse containerterminal. Voor elk probleem gebruiken we eenzelfde aanpak: allereerst wordt het probleem als een wiskundig optimalisatieprobleem gemodelleerd. Vervolgens wordt een exact algoritme gegeven dat de optimale oplossing voor dit probleem uitrekent. Echter duurt het bepalen van de optimale oplossing voor deze problemen vaak te lang om praktisch toepasbaar te zijn. Vandaar dat ook een heuristiek wordt ontwikkeld die snel een oplossing geeft die niet per se optimaal is, maar wel goed is.

In Hoofdstuk 2 wordt voor een groep containers, die op meerdere terminals in een zeehaven staan, bepaald hoe ze het beste vervoerd kunnen worden. Het doel van dit probleem is om zoveel mogelijk containers per binnenvaartschip te vervoeren, maar tegelijkertijd mag een schip ook niet te veel terminals bezoeken, omdat dit de kans vergroot dat het schip vertraging oploopt. Een derde belangrijk aspect dat wordt meegenomen in het bepalen van het transportplan zijn de opslagkosten op zowel de zeeterminal als de binnenlandse terminal. Dit probleem wordt gemodelleerd als een geheeltallig lineair optimalisatieprobleem. Als standaard oplossingsmethoden worden gebruikt voor deze formulering, dan wordt de optimale oplossing verkregen, maar kan de rekentijd een paar uur of meer zijn voor grotere probleeminstanties. We laten zien dat wanneer deze formulering in twee stappen wordt opgelost, waarin in de eerste stap wordt bepaald welke terminals worden bezocht en in de tweede welke containers op welk schip gaan, dan is de rekentijd slechts enkele seconden. Bovendien zijn met deze heuristiek de oplossingen bijna gelijk aan de optimale oplossing. Ten slotte is ook een algoritme ontwikkeld dat het gedrag van een menselijke planner simuleert en daarmee wordt aangetoond dat er veel potentie zit in het implementeren van de andere twee methoden.

Het probleem dat wordt behandeld in Hoofdstuk 3 lijkt erg op dat van Hoofdstuk 2. De twee grote verschillen zijn dat containers nu zowel van als naar de zeeterminals gebracht moeten worden en dat het aantal containers dat geladen en gelost kan worden bij een zeeterminal nog onbekend is als de planning gemaakt moet worden. Hierdoor kan het zijn dat er meer containers op een schip staan dan gelost kunnen worden bij een terminal. We behandelen het aantal containers dat bij een enkele terminal geladen en gelost kunnen worden als een stochastische variabele en modelleren het probleem als een stochastisch probleem met *recourse*. Dit probleem wordt vervolgens opgelost met een techniek

die *Sample Average Approximation* heet. De oplossing van deze methode convergeert naar de optimale oplossing als genoeg trekkingen van de stochastische variabelen worden gebruikt. Het is echter ook mogelijk om sneller oplossingen te genereren. Hiervoor wordt het oorspronkelijke probleem zodanig gesimplificeerd dat de optimale oplossing bepaald kan worden met standaard technieken van stochastisch programmeren. Deze optimale oplossing van het versimpelde probleem kan vervolgens gebruikt worden om de stochastische variabele te vervangen door een deterministische waarde. Deze heuristiek die gebaseerd is op stochastisch programmeren werkt beter dan bestaande technieken voor het vervangen van een stochastische variabele door een deterministische waarde.

In Hoofdstuk 4 wordt niet het beste transportplan voor een groep containers bepaald, maar voor één enkele container. Deze container moet vervoerd worden door een netwerk waarin de reistijden stochastisch zijn. Bovendien bestaat de kans dat een gepland vertrek gemist wordt, omdat deze overboekt is. Het doel is om de goedkoopste route te vinden waarvoor de container vóór een zekere deadline arriveert bij zijn eindbestemming. Vanwege de stochasticiteit heeft elke route een bepaalde kans waarvoor de container te laat aankomt. Aangezien het moeilijk van tevoren te bepalen is welke kans men acceptabel vindt, worden Pareto-optimale oplossingen gegeven, waarin de kosten van een route worden afgezet tegen de kans op een aankomst na de deadline. Naast een optimaal algoritme dat gebaseerd is op dynamisch programmeren, geven we ook een heuristiek waarin alle stochastische variabelen worden vervangen door een risicomaat. Deze risicomaat is een deterministische waarde die vervolgens in een geheeltallig lineair optimalisatieprobleem gebruikt kan worden. Door het variëren van het risico in de risicomaat kunnen ook Pareto-optimale oplossingen gevonden worden.

In Hoofdstuk 5 verleggen we onze focus naar het stapelen van containers op een terminal. Het gebeurt regelmatig dat op een moment dat een container de terminal moet verlaten, er andere containers bovenop hem staan. Deze containers moeten dan naar een andere plek verplaatst worden, terwijl een vrachtwagen staat te wachten op de container die hij moet ophalen. Deze verplaatsingen worden *relocation moves* genoemd. Een alternatief is om de containers al in de juiste volgorde klaar te zetten als het rustiger is op de terminal, wat ook bekend staat als *pre-marshalling moves*. Het probleem met de *pre-marshalling moves* is dat veel meer verplaatsingen van containers nodig zijn dan voor de *relocation moves*. In dit hoofdstuk introduceren we een nieuw soort verplaatsing, namelijk de *pre-processing move*. Deze verplaatsingen worden uitgevoerd als het rustig is op de terminal, maar in tegenstelling tot de *pre-marshalling moves* hoeven de containers niet volledig in de juiste volgorde te staan. We formuleren twee nieuwe problemen in dit hoofdstuk en in Hoofdstukken 6 en 7 worden oplossingmethoden gepresenteerd voor deze problemen.

In het probleem dat we bestuderen in Hoofdstuk 6 wordt de gewogen som van het aantal *pre-processing* en *relocation moves* geminimaliseerd. Voor dit probleem ontwikkelen we een heuristiek waarin containers die het laatst de terminal verlaten, het eerst op de juiste plek moeten worden gezet. Om te weten of deze verplaatsing een verbetering oplevert, moet ook het verwachte aantal *relocation moves* bepaald worden. Een methode die op basis van enkele beslissingsregels een schatting maakt wordt hiervoor gegeven in dit hoofdstuk. De optimale oplossing wordt bepaald met een *branch-and-bound* algoritme, maar voor probleeminstanties met veel containers kost deze methode vaak meerdere uren aan rekentijd.

In Hoofdstuk 7 is het aantal *pre-processing moves* dat uitgevoerd kan worden beperkt. Het optimale algoritme van het vorige hoofdstuk kan met een paar kleine aanpassingen ook gebruikt worden voor dit probleem. Echter in dit hoofdstuk gebruiken we de *branch-and-bound* methode ook in een heuristiek waarin het aantal resterende *relocation moves* niet exact berekend wordt maar wordt geschat. Een andere heurstiek die gepresenteerd wordt in dit hoofdstuk probeert de bovenste container van elke stapel in de juiste positie te zetten. Dit kan op acht verschillende manieren en de manier die de grootste daling in het aantal *relocation moves* oplevert, wordt uitgevoerd. We hebben voor het probleem dat we behandelen in Hoofdstuk 6 aangenomen dat de kraan niet beweegt tijdens het uitvoeren van de *pre-processing moves*. In dit hoofdstuk laten we deze aanname los en gebruiken we de oplossing voor het oorsprokelijke probleem om met behulp van een een geheeltallig lineair optimalisatieprobleem te bepalen hoeveel verplaatsingen in welk deel van de terminal moeten worden uitgevoerd.

Ten slotte bekijken we in Hoofdstuk 8 problemen via een andere blik. Waar in voorgaande hoofdstukken numerieke experimenten gebruikt zijn om de kwaliteit van de heuristieken te evalueren, geven we in dit hoofdstuk algoritmes waarvoor bewezen is dat hun oplossing gegarandeerd binnen een bepaalde marge van de optimale oplossing zit. We ontwikkelen deze zogeheten *approximation algorithms* voor optimalisatieproblemen met twee niveaus van capaciteit. Bekende problemen met een enkel type capaciteit zijn het *Multiple Knapsack Problem*, *Maximum Coverage Problem with Knapsack Constraint* en het *Capacitated Facility Location Problem*. Deze problemen breiden we uit door *knapsacks* onder te verdelen in clusters, die elk ook weer een eigen capaciteit hebben. Met behulp van een beslissingsvariabele die bepaalt hoeveel capaciteit van het cluster naar ondergelegen knapsacks gaat, kunnen bestaande *approximation algorithms* gebaseerd op lineair programmeren worden uitgebreid. Voor bepaalde problemen blijft de marge tussen de optimale oplossing en de oplossing gegeven door het *approximation algorithm* gelijk als de clusters worden toegevoegd en voor anderen wordt ze slechts een fractie groter.