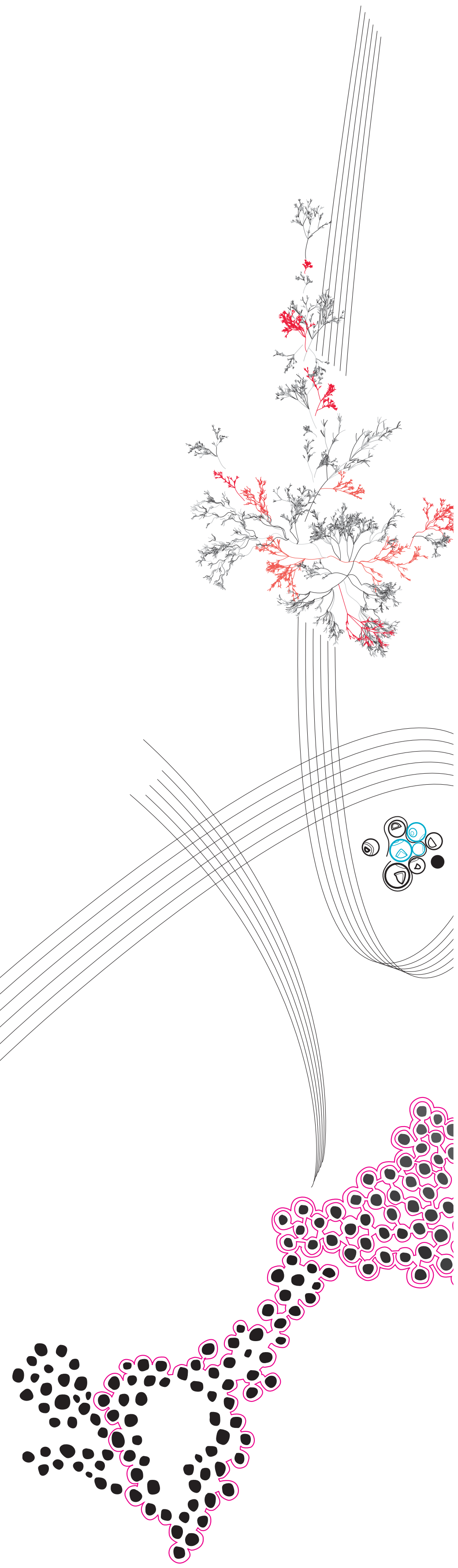
Master Thesis Applied Mathematics



A 3-stage approach to the berth allocation and quay crane specific problem in container terminals using cutting planes

Anne Meulenkamp

Supervisor:
Matthias Walter (University of Twente),
Quirijn Schevenhoven (Cofano Software Solutions)

August, 2023

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

Cofano Software Solutions

UNIVERSITY OF TWENTE.

Acknowledgements

I wish to thank Matthias Walter and Quirijn Schevenhoven for their support throughout this thesis as my supervisors. Additionally, I would like to thank all the employees working at Cofano in Twente who have helped me with debugging, suggestions and emotional support.

Abstract

The berth allocation and quay crane assignment problems are two key problems in container terminals, both maritime and inland. An improved planning with respect to the turnaround time can greatly improve the attractiveness of a terminal to ships. This report considers the berth allocation and quay crane specific problem (BACASP) for inland container terminals. We present a novel 3-stage mixed integer linear programming approach to the BACASP, with a partial integration of the first with the second stage. Time-variant and time-invariant models to the quay crane assignment are considered and solved both exactly and approximately using a rolling horizon strategy. At the heart of the first stage is a type of two-dimensional Knapsack problem. In order to improve performance, novel cutting planes for the two-dimensional geometric knapsack problem (2D-Knapsack) are presented and applied to the BACASP. For 2D-Knapsack, the cutting planes reduce the gap between the dual bound and the optimal solution by 50% on average. Numerical experiments show that our 3-stage approach obtains a good solution for the time-variant BACASP and an optimal solution to the time-invariant BACASP within 10 minutes in inland terminals with up to 30 vessels arriving per day.

Keywords: berth allocation, quay crane assignment, port operations, container transport, mixed integer programming, 2D-Knapsack, cutting planes, BACASP

Contents

1 Introduction

A large amount of consumer goods have at some point in time been transported by ship. This happens not only to things that require continental transport, like bananas from Costa Rica heading to Europe, but also to things that could theoretically have been transported by land, such as cheese from France or toys from China heading to the Netherlands. Although there are always alternative transport methods, such as by plane, train or truck, this is not done due to the costs related to bulk transport. Instead, the toy will first be brought to a big container ship in (for example) China, then that will travel to a sufficiently large port (e.g. Rotterdam) where it will be distributed amongst other transport methods to get to the specific warehouses and then eventually the store. Most of these goods will be transported in containers.

The total volume of containers handled globally per year has been steadily increasing over the past decades [1, 2], and it is expected to continue to increase with larger container ships being built as well as larger demand after the COVID-19 crisis. As a consequence, container terminals are required to increase their throughput capacity to keep up. This has led to new innovations with respect to terminal design and equipment capacity. In the last 20 years, container terminal operations have also become of interest for Operations Research. The focus of this field is to see how throughput can be increased without requiring changes to the terminal design or equipment. This requires detailed knowledge of the entire process that happens at a container terminal. This process is shown in Figure 1, with definitions for common terminology given in Appendix A.

1.1 Processes in container terminals

The first decision that a port operator needs to make is a berthing plan: which place along the quay is allocated to which incoming ship [2]. This needs to take into account that big container ships (which can only berth in sea terminals) need to be pushed and pulled through the port by tugboats. Therefore, the berthing plan cannot require more tugboats than are available. This leads to the second decision: which tugboat is doing what at which time. In other words, this necessitates a tugboat schedule. Here, it is necessary to take into account that different ships have different tugboat requirements, some may require three powerful tugboats while others only need two weaker tugboats [3]. These two decisions tie in with Subfigures a and b of Figure 1. Tugboats are not considered in this report, since we are focused on inland terminals. Inland terminals only serve barges, which do not need tugboats.

The third and fourth decisions concern Figure 1c, namely the quay crane assignment and the quay crane schedule. Quay cranes load and unload ships and place the containers on the quay. One of the decisions to make is how many quay cranes to assign to each ship, while taking into account the limitations of the ship and safety conditions. Once the quay cranes have been assigned, it is still necessary to determine which of the quay cranes is going to load/unload which containers to/from the ship and in which order this will happen, once again taking safety conditions into account [4]. In practice, the quay crane assignment is often integrated with either the quay crane scheduling or the berth allocation, since having more cranes at a berth would mean the ship could be loaded/unloaded quicker, which in turn could lead to a different optimal berthing plan. Alternatively, the berthing plan could be used as input to both assign and schedule the quay cranes [2].

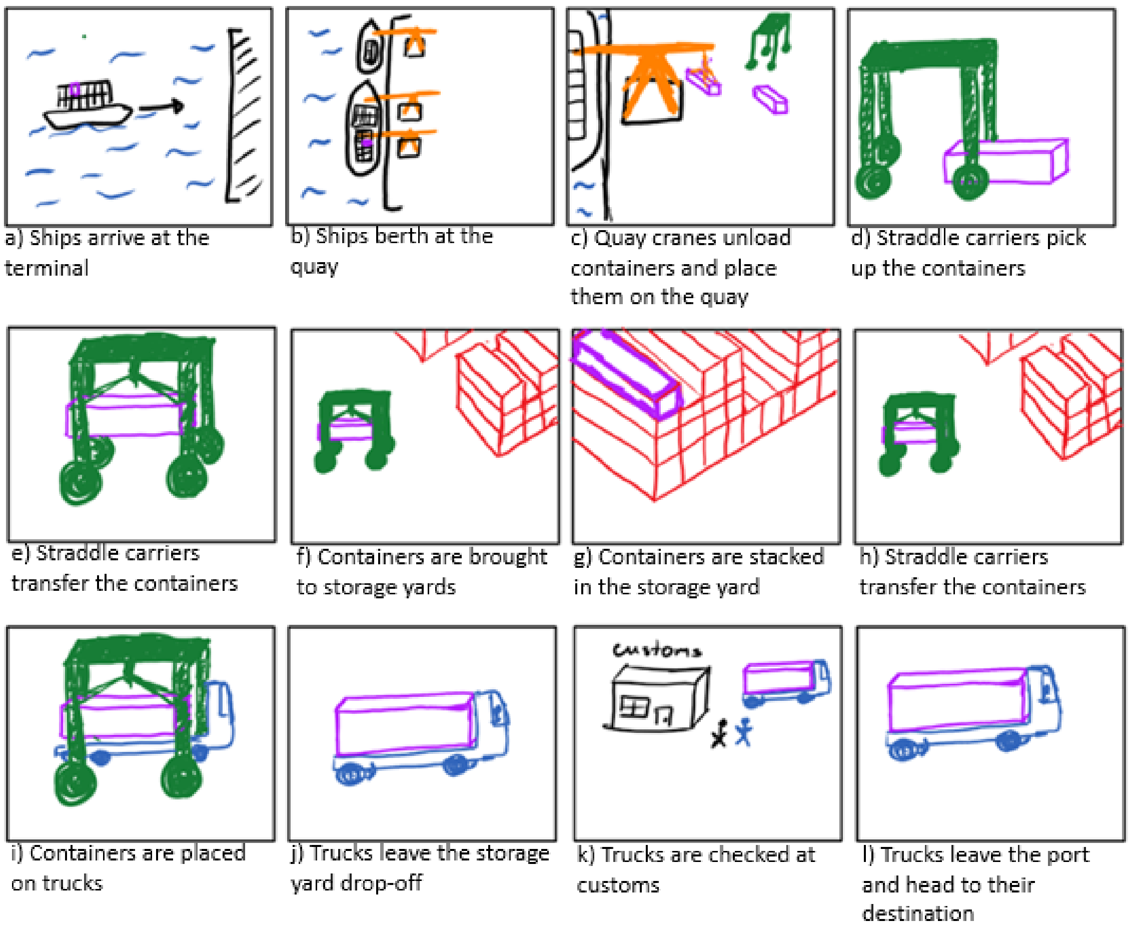

a) Ships arrive at the terminal
b) Ships berth at the quay
c) Quay cranes unload containers and place them on the quay
d) Straddle carriers pick up the containers
e) Straddle carriers transfer the containers
f) Containers are brought to storage yards
g) Containers are stacked in the storage yard
h) Straddle carriers transfer the containers
i) Containers are placed on trucks
j) Trucks leave the storage yard drop-off
customs
k) Trucks are checked at customs
l) Trucks leave the port and head to their destination

FIGURE 1: The logistical chain of a container as it moves through a container terminal. The chain happens in reverse for an arriving inland container that needs to be brought to a container ship. it is also possible that a container does not leave the terminal by truck, but instead by train or by a different ship.

The quay crane schedule also needs to allow a feasible solution to the fifth decision: which transfer vehicle will transport which container from the quay to the storage yard at what time and vice versa (Subfigures d, e, f, h and i). If there are not enough transfer vehicles to travel to and from the storage yard in time to pick up the next container, then the quay cranes would need to halt operations until the container has been removed from the quay. Naturally, this is undesirable.

Once the containers are brought to the storage yard, the next set of decisions has to be made: which container is stacked where (Subfigure g). It is highly inefficient if containers constantly need to be taken from the bottom of a stack for transport, since this requires relocating containers. These relocations each take time and require the availability of suitable equipment that does the relocating. This can be done by a variety of equipment described in Appendix A such as yard cranes, reach stackers and straddle carriers, which leads to the yard crane assignment and scheduling problems. Hence, it is better to stack the containers in a good way, and to make efficient use of yardside equipment. In practice, the quay crane schedule, transfer vehicle schedule and the yard crane schedule have a strong influence on each other, so optimizing them individually or sequentially would likely not give the same optimal solutions as the integrated problem. However, the integrated problem is much more complex [2, 4].

Some time later the containers are removed and brought to their next method of transportation: either a different ship, a train, or a truck. This once again requires efficient routing of the transfer vehicles. In some ports this is done by automated vehicles, in which case it not only requires an efficient schedule, but also has to include a route that avoids collisions with other vehicles. Containers can only be moved to their next mode of transport once that has actually arrived. In some ports it is possible to plan when trucks will arrive for specific containers, in which case this can be spread out over the day to avoid a large difference between peak and average load. This process is shown in Subfigures h, i, and j.

This also ties into the final destination a container has when it leaves the terminal for the hinterland: customs (Subfigure k). Customs serves to check whether the truck is taking/bringing the correct container and also applies random drug searches to see if there are drugs smuggled in the container. Similar to the pick-up/drop-off point in the storage yard, this is greatly aided by more spread out truck arrivals.

1.2 Related work

In this work we focus on the berth allocation and quay crane specific problem (BACASP), which is represented by subfigures a, b, and c. The goal of the BACASP is to obtain a good schedule for the first, third, and part of the fourth decision. It assigns berthing locations, start and end times and specific quay cranes to vessels. A closely related simpler problem is the berth allocation and quay crane assignment problem (BACAP), which only assigns the number of quay cranes that handle a vessel at each time, but not the specific quay cranes. Research for the BACAP can be split into two categories: the tactical BACAP and the 'normal' BACAP. The tactical BACAP is considered by Giallombardo et al. [5] and Vacca et al. [6]; it is used to investigate the costs of possible quay crane profiles to help in negotiations between the terminal and the shipping liner. It considers a long time horizon of up to one month. The 'normal' BACAP, which will henceforth be referred to as the BACAP, is generally solved for a time horizon of 1–2 weeks. Most exact formulations are based on the work of Meisel and Bierwirth [7] or Iris et al. [8] who use a generalized set partitioning model. Due to its complexity, the BACAP is often solved with a heuristic. Used heuristics include adaptive large neighborhood search (ALNS) [9, 10], relaxation rounding [11], squeaky wheel optimization [7], tabu search [7, 5], evolutionary path re-linking [12], and a rolling horizon [9, 13, 14]. All these approaches solve the berth allocation problem and quay crane assignment problem simultaneously.

The BACASP can be subdivided into several types depending on how the specific quay cranes were assigned in the model. The BACASP can either refer to models that first solve the BACAP and then assign specific cranes, models that integrate the specific cranes into BACAP, and models that either integrate or sequentially solve the quay crane scheduling problem. The BACASP was first introduced by Park and Kim [15]; they first solve the BACAP before assigning specific quay cranes. They use a subgradient method for the BACAP, and a dynamic program to obtain specific quay cranes that minimize the total number of setups. A setup occurs each time a crane starts handling a vessel. Correcher [16] also solve this version of the BACASP; they provide a MILP and test the influence of three types of cutting planes. Cheimanoff et al. [17], Correcher [16], Lalita and Murthy [18], and Thanos et al. [19] integrate the specific quay cranes into the BACASP. Lalita and Murthy only consider a subset of feasible solutions satisfying certain properties, and Thanos et al. consider a quay that consists of two disjoint pieces. The quay crane specific problem is included in several differ gradations of integration. Meisel and Bierwirth [20] solve it sequentially, while Malekahmadi et al. [21] and Abou Kasm et al. [22] fully integrate it with the BACAP. The former two group containers per bay, while the latter considers each individual container.

Both the BACAP and the BACASP have a large variation in the objective functions that are considered. Many articles search for any feasible specific crane assignment, making the objective function independent of whether they solve the BACAP or the BACASP. Objective functions that have been researched include deviation from a desired berthing location [17, 18, 23], total waiting time [24, 12, 13, 17], total time spent in port [22, 23], vessel delay [7, 23, 9, 13], and carbon emissions [24].

As far as we are aware, all previous research considers instances based on data from sea terminals, whilst we consider inland terminals. Inland terminals function the exact same way as sea terminals, except that inland terminals only handle barges. This results in three key differences. Firstly, inland terminals do not have tugboats, because tugboats are only needed when large seagoing container ships need to make delicate movements. Secondly, the turnaround time of barges is much faster than that of seagoing vessels. An inland terminal with a quay of 500m can handle up to 30 barges a day, which is a considerable difference from 20–40 vessels per week for sea terminals with a quay of more than 1km. Lastly, since barges travel on inland waterways and not the sea or ocean, they typically travel much shorter distances. Hence, inland terminals often do not know which barges will arrive more than 24 hours in advance. Making a schedule that requires this knowledge more than a week in advance thus does not make much sense.

1.3 Our contribution

This project was done in collaboration with Cofano. Cofano is a software company that aims to digitize all processes in container terminals. Many of their clients are inland terminals. As part of their aim to improve the workflow of these processes, Cofano is interested in the potential of automating some of these processes, including the berth allocation and quay crane assignment processes. During conversations within Cofano and with clients of Cofano, we noticed that the most important aspect of an automatic planner is that it needs to be fast so that it can be rerun if necessary, for instance if a vessel is delayed, if an unscheduled vessel arrives, or if more or fewer quay cranes are available than initially planned; unscheduled vessels especially are a daily occurrence at some inland terminals.

We introduce a 3-stage model for the BACASP for inland terminals that fills this gap in current research. The first stage assigns quay cranes per time step for each vessel. In this stage we minimize the total turnaround time (the total time vessel spend at the terminal). The second stage allocates a berthing position to each vessel based on the solution to the first stage. This stage only generates a feasible berthing allocation. We do not take preferred berthing locations into account, because there is little difference in the distance between berthing positions and the storage yard at an inland terminal. These two stages are solved in an iterative process, because some solutions to stage 1 are infeasible in stage 2. In this case, the solution is rejected, and a new solution to stage 1 is found. The combination of stage 1 and 2 precisely solves the BACAP. We solve the third stage sequentially to the integration of stage 1 and 2. Stage 3 assigns specific quay cranes based on the solutions to stage 1 and 2. We consider three different objective functions, one of which is the total number of setups introduced by Park and Kim [15].

All three stages are modelled using mixed integer linear programs (MILP). Stages 2 and 3 are solved exactly, while a (potentially) suboptimal solution to stage 1 is obtained using a rolling horizon strategy. Each segment of the rolling horizon is solved using a MILP solver. We obtain a solution to the BACASP that is optimal in each time segment of the rolling horizon within 10 minutes in all instances with less than 30 vessels and in 98 out of 100 considered instances with 30 vessels. All instances considered are based on data from one of Cofano's clients. All assumptions we make in this work regarding the BACASP are given below. Assumptions 1 to 3, 6 and 8 are made to simplify the model.

Assumption 1 (deterministic data)**.** The arrival times and quay crane speeds are deterministic. In practice, vessels will arrive before or after their estimated arrival time. In the case of a delay, the vessel can notify the terminal. The terminal can then decide to push back the schedule or to reschedule the ship with its new estimated arrival time. Since our model is meant to be resolved in case something happens, this is precisely one such case. Also, we are only scheduling barges at an inland terminal, the actual and estimated arrival times will not differ as much as they differ for seagoing vessels.

Assumption 2 (seamless shifts)**.** Personnel shifts connect seamlessly, causing no time delay in handling a vessel. Also, there are no breaks included in the schedule for personnel. In practice, small breaks often happen while vessels are berthing/unberthing, since the time between unloading the final container of a previous vessel until loading the first container of the current vessel takes at least 30 minutes due to assumption 9. Longer breaks could be included by reducing the crane availability for the duration of the break.

Assumption 3 (further transfer is not limiting factor)**.** The quay crane speed is the limiting factor for the unloading/loading speed of containers. This means that there are sufficient transfer vehicles, yard equipment and personnel. If the loading/unloading speed was to also depend on the transfer vehicles, yard equipment or personnel, the BACASP would depend on all other operations of a container terminal, each of which is a difficult problem to solve in its own right. In practice, the terminal knows the division of labor to maximize the quay crane efficiency, since using a crane is expensive in terms of personnel and fuel.

Assumption 4 (quay crane interference)**.** When multiple quay cranes are used to unload/load a vessel simultaneously, the loading/unloading speed does not scale linearly, since quay cranes need to maintain a safe distance to avoid collisions. The speeds are assumed to be known by the terminal. A terminal has been using their quay cranes to load and unload all the vessels that arrive there. They naturally know how many containers can be loaded per hour for each number of quay cranes. In practice, it is possible that this speed is different depending on the vessel size. If two cranes handling a vessel can be positioned far enough apart that there is no interference, the speed should increase linearly. This can easily be adapted in our model, but this is not done due to a lack of data.

Assumption 5 (non-crossing of quay cranes)**.** Quay cranes in container terminals are generally rail-mounted gantry cranes. These cranes can therefore not cross each other when moving along the quay.

Assumption 6 (no stowage plan)**.** The positions of containers on the vessel (stowage plan) are not taken into account when assigning the quay cranes. If the positions of containers need to be taken into account, then the model needs to include processing stowage plans of vessels, as well as which combinations of crane assignments are possible. We deem this better suited to a crane profile approach to solving the quay crane assignment problem, since this inclusion would reduce the set of feasible profiles, which could even make the model faster. Additionally, communication between vessels and inland terminals can be on very short notice, so the stowage plan may only be known once the vessel has already arrived.

Assumption 7 (no draft)**.** The draft of vessels and water depth at the berth are not taken into account. Inland terminals almost exclusively handle barges. Barges have a small draft, so this does not need to be taken into account.

Assumption 8 (uninterrupted berthing)**.** Once a vessel is berthed, it remains berthed until all containers have been loaded/unloaded. The only time a vessel unberths before loading and unloading all containers is if the terminal planner has made a decision to first unload all containers and to later load. This could be included using a simple precedence constraint by seeing the two cases as separate vessels. Since this is done on a case by case basis based on necessity, we do not include the additional complexity in the model. Doing so without necessity only increases the amount of wasted time due to assumption 9.

Assumption 9 (berthing time)**.** Berthing and unberthing of a vessel takes a fixed amount of time each, regardless of the size of the vessel. Once a vessel arrives at a terminal, it waits some distance away from the quay until its scheduled berthing time. Travelling from this location to its berthing location and anchoring takes time; we call this set of actions berthing (unberthing) for when a vessel starts (finishes) service. Based on data from a client of Cofano, we assume the time taken is 15 minutes each for berthing and unberthing. The berthing/unberthing time depends on the distance between the waiting area and the quay, and the value is thus different for every terminal.

Assumption 10 (buffer space)**.** The space needed around a vessel to allow for berthing and unberthing is 10% of its size. This buffer space is needed to allow vessels movement space to get in or out of their berthing location, similar to cars for parallel parking. The value is based on a client of Cofano.

As far as we are aware, the way we model the quay crane speed in assumption 4 is unique, as other models either model quay crane interference using a formula or by scaling linearly. Similarly, we have found no other work where assumption 9 is included in a time-variant model. Models either include extra quay crane hours, which would translate to a different berthing time depending on the number of cranes handling the vessel over all time steps, or do not include it at all.

The bottleneck to our 3-stage model is the first stage. As part of our attempts in improving the time taken by the solver to prove optimality, we consider a 2-stage model to the two-dimensional knapsack problem as a relaxation of the BACAP. The first stage of this model can be seen as a relaxation of stage 1 of the BACASP model, while stage 2 is the same for both models. We introduce a new type of valid inequality for the two-dimensional knapsack problem, and investigate its influence on the dual bound. Several methods of lifting the inequalities are considered. The inequalities are then applied to stage 1 of our 3-stage BACASP model. The inequalities close the gap between the LP relaxation and the optimal solution by 50% on average for the 2D-Knapsack problem. There was no notable influence on the LP relaxation of stage 1 of the BACASP.

1.4 Outline

In Section 2 we discuss how we implemented our models with regard to hardware and software, as well as how we generated the instances used to test the models. The 2-stage model for the two-dimensional knapsack problem and the related inequalities are given in Section 3. This is followed by the models for stage 1, 2, and 3 in Sections 4 to 6 respectively. These sections also give a more detailed overview of the variations of their respective problems in literature. The integration of the three stages is given in Section 7.

2 Data for computational studies

In this section, we state general information about the computational experiments we conducted to investigate and improve the 2-stage approach for 2D-Knapsack (Section 3) and 3-stage approach to the BACASP (Sections 4 to 7).

All computational tests described in Sections 3 to 7 were carried out on a Lenovo Thinkpad with a 2.30 Ghz 11th Gen Intel(R) Core(TM) i7-11800H and 32 GB RAM. In each test we used a time limit of 600 seconds of CPU time for each instance in the considered test set, unless otherwise specified. Tests were done using SCIP 8.0.3, which is a framework that integrates constraint and mixed integer programming [25]. As underlying LP solver in SCIP 8.0.3 we used SoPlex 6.0.3. We used SCIP with a Python interface through PyScipOpt [26] in Python 3.10.6.

In Section 3 we introduce novel cutting planes that are included in the model through a so-called separator. The efficacy of individual cutting plane separators depends on the way they are integrated into SCIP, i.e., on the way they interact with other features of the solver, including other cutting plane separators [27]. We decided to focus on the performance of our separator when it is basically isolated by disabling features of SCIP that can interfere with its performance. Therefore, in all test runs for developing the separator, we considered only the root node of the branch-and-cut tree and called up SCIP with its default settings except for disabling primal heuristics, strong branching, other separators and restarts. The reasons are as follows. Firstly, a successful primal heuristic might cause further dual propagations, which could lead to the generation of further cuts. Secondly, every branching strategy employing strong branching can detect infeasibility of subproblems of a MIP (see [28]) and may therefore cause fixing of variables. Since this influences the dual bound, we used most infeasible branching instead (see [28], [27]). Thirdly, all other cutting plane separators must be disabled to measure only the effect of our cutting planes on the dual bound. Lastly, restarts can lead to the generation of further cuts [27].

As part of our attempts to improve the speed with which SCIP proves the optimality of a solution for the BACASP, we made several changes to the *separating* and *emphasis* parameters. The former has three settings in addition to the default: off, fast, and aggressive. The 'off' setting disables all separators, the 'fast' setting limits the time spent using separators, whilst 'aggressive' increases the usage of the separators. The *emphasis* parameter has 10 settings besides the default: counter, cpsolver, easycip, feasibility, optimality, hardlp, phasefeas, phaseimprove, phaseproof, and numerics. Each setting for the emphasis parameter changes multiple other parameters. We have considered the easycip, feasibility, optimality, and hardlp settings in addition to the default setting. These are meant to solve easy problems fast, detect feasibility fast, prove optimality fast and be capable of handling hard linear programs. The remaining settings are deemed unsuitable based on their description in the SCIP documentation.

We used two different test sets for the 2D-Knapsack problem and the BACASP. The instances for the BACASP were derived from data provided by Cofano on one of their clients. This is described in more detail in Section 2.1. The instances for the 2D-Knapsack problem were generated according to the method described by Fekete et al. [29], which is described in more detail in Section 2.2. We decided against deriving instances for 2D-Knapsack from the BACASP, because this would make it difficult to compare our results to those in literature.

2.1 Instances for the BACASP

Instances are generated based on data from the terminal considered in this report (a client of Cofano) to investigate the functionality of our 3-stage approach. The data consists of a year of operations from the terminal in question. Each instance consists of a set of vessels with their respective lengths, number of containers, and arrival times. The rest of this section discusses how the instances were generated in more detail.

Terminal specifications

The terminal consists of a single quay that is 500m long. The quay has a total of 10 quay cranes of the same type. At this terminal, it is custom to leave a buffer space of 10% of the vessel length per vessel to allow for movement at the quay. Additionally, the time it takes for a barge to berth and unberth is approximately 15 minutes each per barge [30]. A barge can use up to 3 quay cranes simultaneously. When multiple cranes are handling a vessel simultaneously, they have to take into account a safety distance between the cranes. It can happen that one crane needs to wait a bit or choose a less ideal unloading sequence in order to keep this distance. The more cranes are used, the worse the interference gets. This means that the loading/unloading speed of multiple quay cranes does not increase linearly with the number of cranes. Even in an ideal situation where there is no interference, the crane operation speed does not reach its theoretically possible value. This is because the cranes are operated by humans, which means there is a delay in attaching the container to the crane, moving the container and finally detaching the container again.

Table 1: The quay crane specifications are given per feasible number of cranes. The number of in parentheses is the theoretical speed if there is no interference and people are robots, while the other number is the actual speed [30].

number of cranes	1	2	3
loading speed	15 (20)	23 (40)	30 (60)

Vessel lengths

The vessel length refers to the length of the side of the vessel that is next to the quay; it is therefore synonymous with the section of the berth that is occupied by the vessel upon adding the 10% buffer. The barge lengths were originally provided in centimeters, but this is overly precise. There is no need to consider the barge lengths to the precision of centimeters; a discrepancy of up to 1 meter is not an issue. Therefore, the instances will be generated with the barge lengths rounded up to the nearest meter. The barge length distribution does not resemble any common discrete distribution, so it will be used directly as shown in Figure 2. The seagoing container ships have a length of 300m.

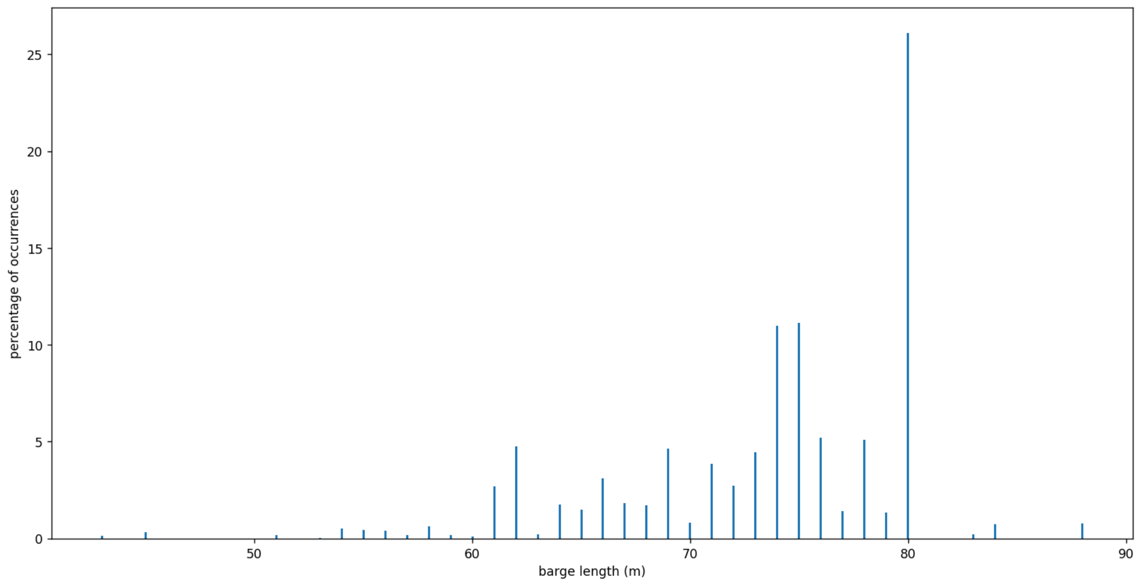


FIGURE 2: The probability mass function obtained from the data by rounding all lengths up to the nearest integer and dividing by the total number of occurrences.

Number of containers

The number of containers is heavily dependent on the vessel size. A small barge with a length of 44m has a smaller upper limit to the number of containers it can carry than a large barge with a length of 80m. Unfortunately, it is unknown to the author what this limit is for the barges that arrive at this terminal. Hence, the number of containers seen in the data will be used as a replacement.

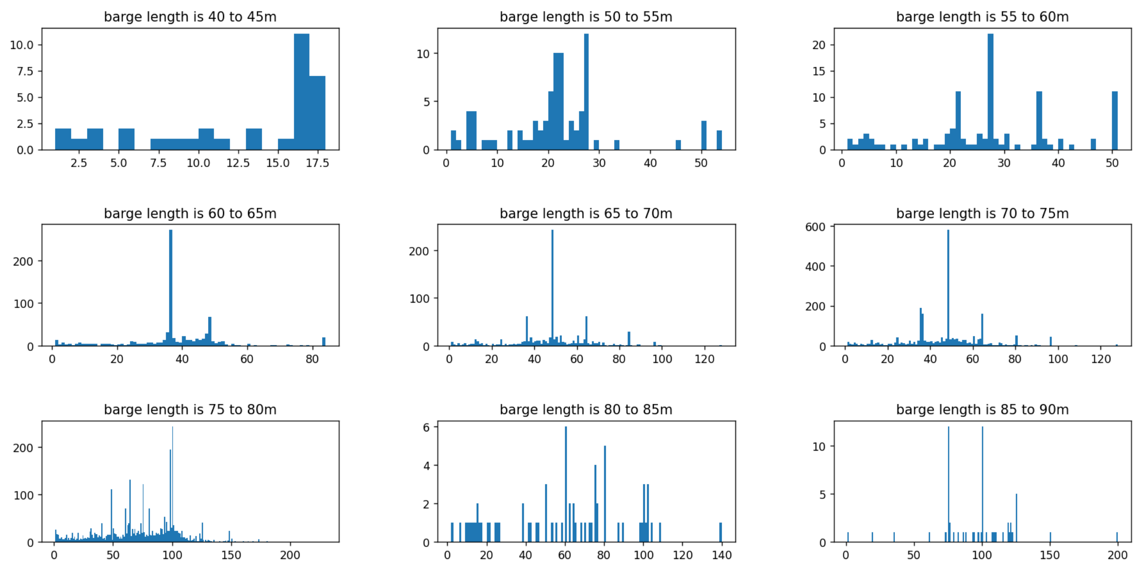


FIGURE 3: For each subplot the x-axis is the number of containers that were loaded/unloaded, and the y-axis is the number of times a specific number of containers was handled over the course of a year.

The barges are allocated to 5 intervals based on Figure 3. The distribution for each interval is a uniform discrete distribution with a minimum of 1 container and a maximum based on the maximum number of containers for that interval from the data.

TABLE 2: The maximum number of containers that were loaded/unloaded per interval. The barge lengths per interval are in meters.

group	[40, 45]	[50, 60]	(60, 65]	(65, 75]	(75, 90]
maximum	18	54	84	128	230

Although such a distribution does not truly correspond to the distributions shown in Figure 3, it fully encapsulates the feasible range of containers that need to be loaded/unloaded. Hence, the instances maintain most of their realism for this terminal.

Arrival times

Vessels can arrive and be serviced at any time during the day. Although it would be nice to assume that these arrivals are uniformly distributed, this does not need to be the case. The arrival times of barges over the course of a year has been plotted below in Figure 4, where it can easily be seen that the arrivals do not occur randomly in time.

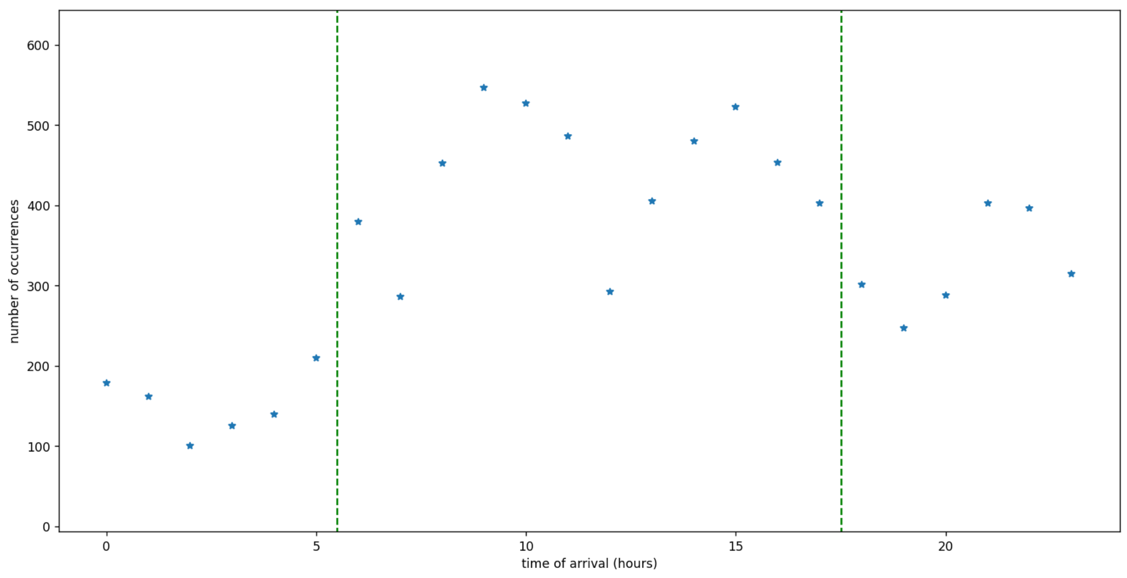


FIGURE 4: The number of times barges arrived during each hour over the course of a year. For example, there were 178 vessels that arrived between 00:00 and 01:00.

The vertical green lines divide the arrivals into three sections: the idle, moderate and busy period. The idle period refers to the time from 00:00 up to 06:00 when relatively few barges arrive at the terminal, the moderate period refers to 18:00 up to 24:00 when quite some barges arrive, and the busy period refers to 06:00 up to 18:00 when a lot of barges arrive. These periods respectively correspond to night, evening, and day hours. This division is supported by knowledge from the terminal that the barge captains do not like to travel in the dark for safety reasons.

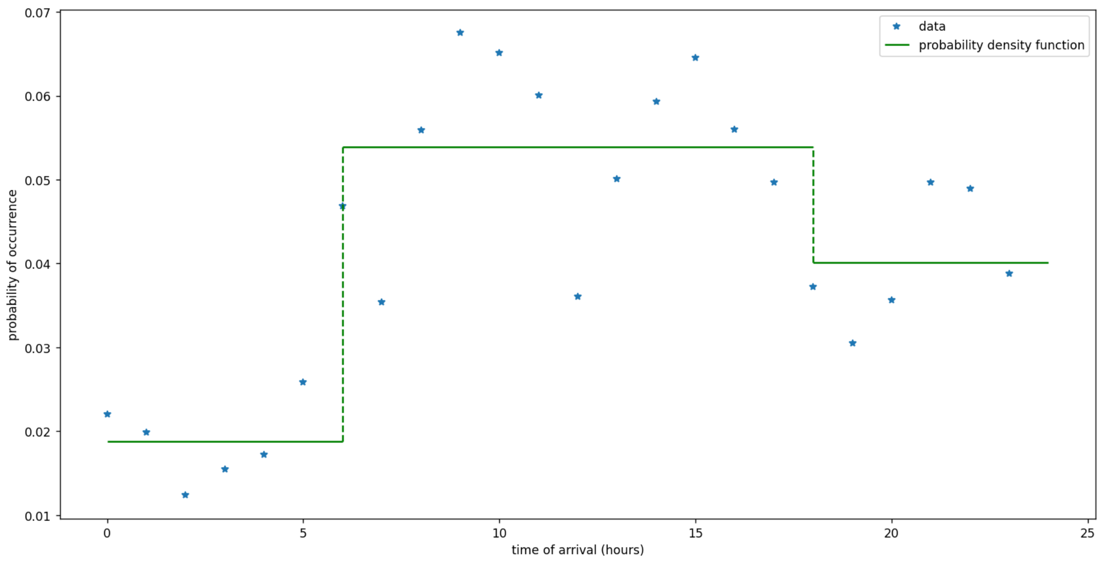


FIGURE 5: The chosen probability density function plotted with the probabilities that follow from the data.

The distribution of the arrivals times used to generate instances was chosen to consist of a piecewise combination of three continuous uniform distributions. The corresponding probability density function is shown in Figure 5, and is also given below.

$$f_a(x) = \begin{cases} 0.0188 & \text{if } x \in [0, 6) \\ 0.0539 & \text{if } x \in [6, 18) \\ 0.0401 & \text{if } x \in [18, 24) \end{cases}$$

Number of vessels

The number of vessels in an instance has a great influence on the computation time, since the state space is directly related to the number of vessels. Hence, the number of vessels of an instance is not determined based on a probability distribution. Instead, the number of vessels of an instance is fixed ahead of time.

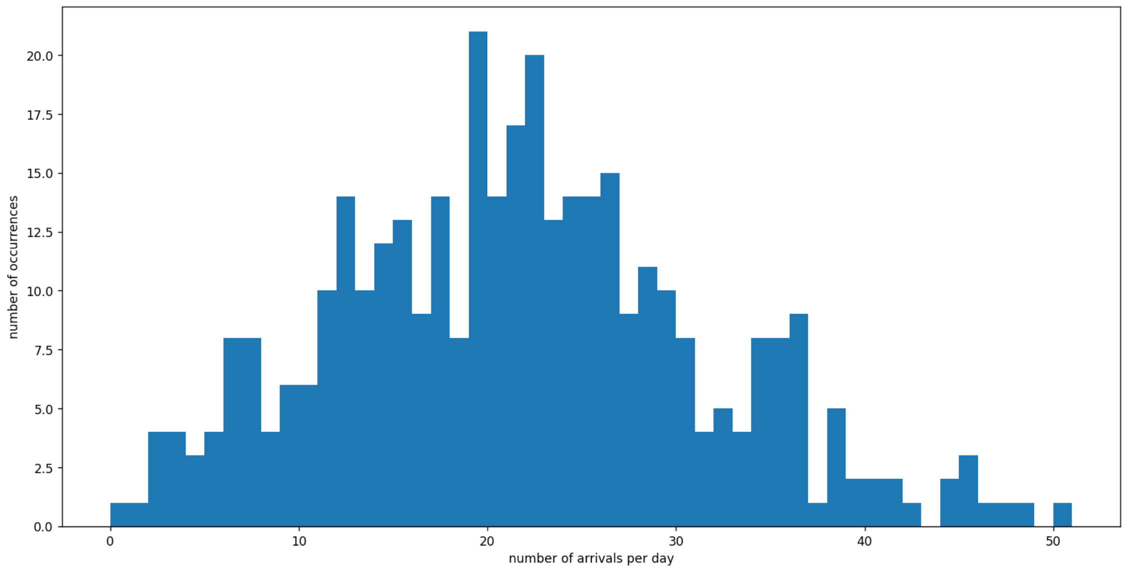


FIGURE 6: The number of times a specific number of barges arrived during one day over the course of a year.

The distributions of the number of barge arrivals in one day is shown in Figure 6, where it can be seen that the number of arrivals varies between 0 and 50. The number of arrivals is mostly between 5 and 40, with arrivals higher than 40 being very unlikely. This report considers instances with 10, 15, 20, 25 or 30 barges. Instances with 5 barges are ignored, since they would all be handled immediately upon arrival, leading to a waiting time of 0. Cases with more than 30 barges are also ignored, because handling 30 barges already takes up a full 24 hours of work. Therefore, the remaining barges would need to be considered again during the next planning horizon.

2.2 Instances for 2D-Knapsack

Recall that we investigate the two-dimensional knapsack problem as a relaxation of the BACAP, with the goal of trying to find possible methods that could improve the time taken by the solver in stage 1 of our 3-stage approach. In order to obtain results that are not restricted to the BACASP test instances, we consider state-of-the-art instances for the 2D-Knapsack problem that we describe below.

The instances for the 2D-Knapsack problem are generated in the same manner as done by Fekete et al. [29] for their OKP-2 instances. They consider 4 item types: tall, wide, big and small, and three instances types with fixed percentages of each item type. We give these in Table 3 and Table 4 for completeness. We consider the 2D-Knapsack as a relaxation of the berth allocation problem. Since we focus on only the ships arriving within 24 hours for the BACASP, having a width of 100 for the 2D-Knapsack instances is quite a discrepancy. Hence, we consider a knapsack of 100-by-25 instead of 100-by-100 as Fekete et al. [29]. To obtain a very similar distribution of items within instances, we divide the widths obtained by Table 3 by 4 and round up. The solutions to our instances can thus be seen as approximations to the original instances. We generate 100 instances with 20, 40 and 60 items of each instance type.

Table 3: The distributions of the four items types used by Fekete et al. The length and width are evenly distributed within the given range according to a discrete uniform distribution.

item type	**width**	**length**
tall	$[1, 50]$	$[75, 100]$
wide	$[75, 100]$	$[1, 50]$
big	$[50, 100]$	$[50, 100]$
small	$[1, 50]$	$[1, 50]$

Table 4: The percentage of items of each type included in an instance of type I, II, and III, as used by Fekete et al.

instance type	tall	wide	big	small
I	20	20	20	40
II	15	15	15	55
III	10	10	10	70

3 A 2-stage approach to 2D-Knapsack

The berth allocation problem in its most basic form resembles a more complicated version of the two-dimensional knapsack problem, sometimes referred to as the two-dimensional geometric knapsack problem, 2D-Knapsack, or 2D-KP. 2D-Knapsack is a fairly intuitive extension of the well-known 0-1 Knapsack problem. Items are represented as rectangles with a length and width, and the knapsack is seen as a larger rectangle. The objective is to maximize the total profit of the items that fit in the knapsack without overlapping. This problem has been extensively researched due to its prevalence in cutting stock in factories: for instance to cut a large metal plate into smaller pieces of different sizes. Due to the limitations of machinery, cutting stock research is often limited to guillotine cuts. Guillotine cuts are edge-to-edge cuts that run parallel to the edges of the material. The difference between guillotine and non-guillotine cuts is shown in Figure 7. The machines often also have restrictions on the number of cuts they can make per piece of material, leading to k-stage guillotine cuts, where k is the number of cuts [31]. Such restrictions are not present for berth allocation, and they would only serve to unnecessarily restrict possible berthing locations for vessels. Other variants, which allow for rotation of items by 90 degrees, deal with irregularly shaped items, or have restrictions on the relative positions of certain items also exist [31].

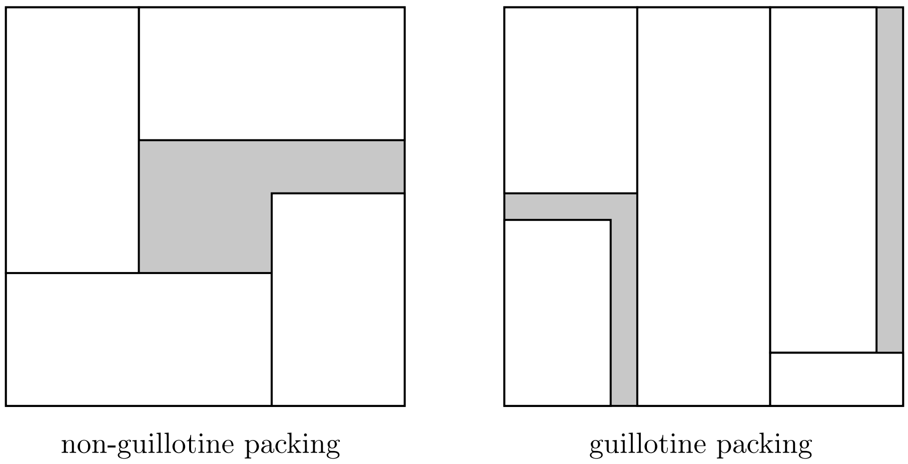


Figure 7: A visual representation of a non-guillotine (left) and guillotine (right) packing of rectangles into a square.

We consider the 2D-Knapsack problem with rectangular items that must be placed parallel to the edges of the knapsack (a larger rectangle) without rotation or additional restrictions. 2D-Knapsack is a strongly NP-hard problem due to its reduction from the bin packing problem, a well-known strongly NP-hard problem. The reduction is given in Appendix B. A lot of research focuses on approximation schemes and heuristics, especially if it is motivated from a cutting stock perspective, since businesses need good solutions, not optimal solutions. Research into the 2D-Knapsack problem (2D-KP), 2D bin packing problem (2D-BPP), strip packing problem (SPP), and 2D orthogonal packing (2D-OPP) regarding both exact and heuristic solution methods over the last 20 years has been nicely summarized by Iori et al. [31]. Similarly, Cacchiani et al. [32, 33] has summarized all new insights into many variants of the one dimensional, multidimensional, and multiple knapsack problems.

We consider three formulations for the 2D-Knapsack problem. The first formulation is adapted from the work of Chen et al. [34]. That formulation uses the relative positions of the items in a 3D-knapsack problem. We have adapted this formulation to the two-dimensional case. The advantage of this formulation is that the number of constraints and variables scale only in the number of items. The second formulation is adapted from the feasibility problem of Baldacci and Boschetti [35], which directly assigns a horizontal and vertical coordinate to the bottom-left corner of each item. We propose a novel 2-stage formulation that uses a combination of the former two as our third formulation. The use of a 2-stage model for the 2D-Knapsack is itself not novel. Baldacci and Boschetti [35] and Fekete et al. [29] also used a two-stage approach, but their first stage only selects a subset of items to place in the knapsack and does not assign any positions yet. Their second stage checks whether there exists a feasible packing with those items. Our first stage assigns a horizontal coordinate to each item, while the second stage checks whether a feasible packing exists for such an assignment. If it does not, it cuts off the solution. The first stage to our model is highly similar to that of Côté et al. [36] and Delorme et al. [37] for the strip packing problem. The parameters we use for all three formulations are given in Table 5.

Table 5: The parameters for all three formulations for 2D-Knapsack.

$\mathcal{I}$	The set of items.
l_i	The length of item i.
w_i	The width of item i.
p_i	The profit of item i.
L	The length of the knapsack.
W	The width of the knapsack.
M	A sufficiently large number ($M \geq L, W$).
H_i	The set of feasible horizontal positions for item i: $H_i := \{0, ..., W - w_i\}$.
V_i	The set of feasible vertical positions for item i: $V_i := \{1, ..., L - l_i\}$.

In order to use multi-stage approaches effectively, it is crucial to supply the earlier stage with as much information as possible, because this may reduce the number of queries to the second stage. One way of doing this is by applying inequalities that are valid for 2D-Knapsack as cuts to improve the dual bound. Baldacci and Boschetti [35] apply knapsack, extended knapsack, dominance, incompatibility, and cover inequalities to improve the dual bound. Belov and Scheithauer [38] apply Chvatal-Gomory cuts to both the one- and two-dimensional cutting stock problem. We introduce a new type of inequalities by making use of the discretized structure of one dimension of the knapsack. These inequalities are thus suitable for both the position assignment formulation and for the first stage to our 2-stage approach. As far as we are aware, no research into inequalities using this property exists.

In this section we first give the three formulations for 2D-Knapsack in Sections 3.1 to 3.3. After this, we give a brief summary of the 0-1 Knapsack problem in Section 3.3.1. This serves as an introduction to terminology for Sections 3.3.2 to 3.3.5 where we introduce the new inequalities and how to strengthen them.

3.1 Relative position formulation

Our first formulation is called the relative position formulation (RPF). It prevents items from being in the same location through the relative positions of items. The relative positions are then used in a series of non-overlapping constraints. Two items can either occupy the same vertical position at different horizontal positions, meaning that one item is to the left of the item, or two items can share the same horizontal position with different vertical positions. The parameters are given in Table 5 and the decision variables are given in Table 6.

Table 6: The decision variables for the relative position formulation for 2D-Knapsack.

x_i	Equals 1 if and only if item i is in the knapsack.
v_i	The vertical position of the bottom-left corner of item i.
h_i	The horizontal position of the bottom-left corner of item i
w_{ij}	Equals 1 if and only if item i is to the right of item j.
z_{ij}	Equals 1 if and only if item i is above item j.

$$
\begin{aligned}
\max \quad & \sum_{i \in V} p_i x_i && && \text{(1a)}\\
\text{s.t.: } & v_j + M(1 - z_{ij}) \geq v_i + l_i && \forall i,j \in \mathcal{I}, i \neq j && \text{(1b)}\\
& h_j + M(1 - w_{ij}) \geq h_i + w_i && \forall i,j \in \mathcal{I}, i \neq j && \text{(1c)}\\
& w_{ij} + w_{ji} + z_{ij} + z_{ji} \geq x_i + x_j - 1 && \forall i,j \in \mathcal{I}, i \neq j && \text{(1d)}\\
& w_{ij} + w_{ji} + z_{ij} + z_{ji} \leq 2x_i && \forall i,j \in \mathcal{I}, i \neq j && \text{(1e)}\\
& \sum_{i \in V} w_i l_i x_i \leq LW && && \text{(1f)}\\
& w_{ij}, z_{ij} \in \{0,1\} && \forall i,j \in \mathcal{I}, i \neq j && \text{(1g)}\\
& v_i \in V_i && \forall i \in \mathcal{I} && \text{(1h)}\\
& h_i \in H_i && \forall i \in \mathcal{I} && \text{(1i)}\\
& x_i \in \{0,1\} && \forall i \in \mathcal{I} && \text{(1j)}
\end{aligned}
$$

The objective is to find a packing of the knapsack that maximizes the profit. Constraint (1b) ensures z_{ij} can only equal 1 if item i is placed above item j. Constraint (1c) ensures w_{ij} can only equal 1 if item i is placed to the right of item j. Constraint (1d) requires that if items i and j are both in the knapsack, then item i must be above j (or vice versa), i must be to the right of j (or vice versa), or both. This ensures that no two items can occupy the same position in the knapsack. Constraint (1e) ensures that constraints (1b) and (1c) are always true if item i is not in the knapsack. Constraint (1f) is not necessary for a correct formulation, but it greatly improves the dual bound and is hence included; it restricts the LP relaxation from 'cheating' by preventing solutions that take up more space than the knapsack has. Constraints (1g)-(1j) define the decision variables for the vertical and horizontal overlap and the vertical and horizontal positions respectively.

3.2 Position assignment formulation

The second formulation is called the position assignment formulation (PAF). It prevents items from being in the same location through directly assigning horizontal and vertical positions to the items and requiring that at most one item can occupy each position. The formulation given here is derived from the feasibility problem presented by Baldacci and Boschetti [35].

Table 7: The decision variables for the position assignment formulation for 2D-Knapsack. The parameters can be found in Table 5

s_{ih}	Equals 1 if and only if the left side of item i is at horizontal position h.
t_{iv}	Equals 1 if and only if the bottom of item i is at vertical position v.
x_{ivh}	Equals 1 if and only if position (h, v) is occupied by item i .

$$\max \quad \sum_{i\in\mathcal{I}} p_i \sum_{h\in H_i} s_{ih} \tag{2a}$$

$$\text{s.t.:} \quad \sum_{h\in H_i} s_{ih} \leq 1 \qquad \forall i \in \mathcal{I} \tag{2b}$$

$$\sum_{v\in V_i} t_{iv} = \sum_{h\in H_i} s_{ih} \qquad \forall i \in \mathcal{I} \tag{2c}$$

$$\sum_{v=0}^{W-1} x_{ihv} = l_i \sum_{h'=h-w_i+1}^{h} s_{ih'} \qquad \forall h \in \{0,...,W-1\} \tag{2d}$$

$$\sum_{h=0}^{L-1} x_{ihv} = w_i \sum_{v'=v-l_i+1}^{v} t_{iv'} \qquad \forall h \in \{0,...,W-1\} \tag{2e}$$

$$\sum_{i\in\mathcal{I}} x_{ihv} \leq 1 \qquad \forall h \in \{0,...,W-1\}, \forall v \in \{0,...,L-1\} \tag{2f}$$

$$x_{ihv} \in \{0,1\} \qquad \forall i \in \mathcal{I}, h \in \{0,...,W-1\}, \forall v \in \{0,...,L-1\} \tag{2g}$$

$$s_{ih} \in \{0,1\} \qquad \forall i \in \mathcal{I}, \forall h \in H_i, \tag{2h}$$

$$t_{iv} \in \{0,1\} \qquad \forall i \in \mathcal{I}, \forall v \in V_i \tag{2i}$$

The objective is to find a packing of the knapsack that maximizes the profit. Constraint (2b) ensures that each item can be included in the knapsack in at most one feasible horizontal position, and constraint (2c) ensures that it is only assigned a feasible vertical position if it also gets a horizontal position. Constraints (2d) and (2e) define that x_{ihv} is equal to 1 if and only if the bottom-left corner of the item is placed such that the item covers position (h, v). Constraint (2f) ensures that every position (h, v) can be occupied by at most one item. Constraints (2g), (2h) and (2i) define the x, s and t-variables.

3.3 2-stage approach

The third formulation solves the 2D-Knapsack problem in two stages. It first assigns a horizontal position to every item, and then checks whether vertical positions exist such that no item overlaps, thus making two stages. To be more precise, the first stage assigns a horizontal position (w) to the bottom-left corner of each item, and the second stage assigns a vertical position (l). Since the second stage can be infeasible for a given solution to the first stage, the two stages need to be linked together either through a feedback loop or by checking feasibility for every new incumbent primal solution when using a mixed integer linear programming (MILP) solver.

TABLE 8: The decision variables for the formulation for stage 1 of the 2D-Knapsack problem. The parameters can be found in Table 5.

y_{iw}	Equals 1 if and only if the bottom-left corner of item i is at width w.
x_{iw}	Equals 1 if and only if some part of item i is at width w.

The first stage for 2D-Knapsack is modelled in (3), with the objective to maximize the total profit. Constraints (3b) and (3c) allow each item to be assigned to at most one feasible horizontal position. Constraint (3d) defines the auxiliary variable x. It is used in constraint (3e) to prevent the total length of items that occupy slice w from exceeding L, because any solution that does not satisfy this condition is infeasible in stage 2.

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{n} p_i \sum_{w=0}^{W-w_i} y_{iw} & & \text{(3a)}\\
& \sum_{w=0}^{W-w_i} y_{iw} \leq 1 & \forall i \in \mathcal{I} & \quad \text{(3b)}\\
& \sum_{w=W-w_i+1}^{W-1} y_{iw} = 0 & \forall i \in \mathcal{I} & \quad \text{(3c)}\\
& \sum_{v=\max(0,w-w_i+1)}^{w} y_{iv} = x_{iw} & \forall i \in \mathcal{I}, \forall w \in \{0, ..., W-1\} & \quad \text{(3d)}\\
& \sum_{i \in \mathcal{I}} l_i x_{iw} \leq L & \forall w \in \{0, ..., W-1\} & \quad \text{(3e)}\\
& y_{iw}, x_{iw} \in \{0,1\} & \forall i \in \mathcal{I}, \forall w \in \{0, ..., W-1\} & \quad \text{(3f)}
\end{aligned}
$$

The second stage checks whether a feasible packing exists for a given solution y to stage 1. The decision variables and additional parameters are given in Table 9.

TABLE 9: The additional parameters (above) and decision variables (below) for the formulation for stage 2 of the 2D-Knapsack problem

$\mathcal{I}_1$	The set of items that are included in the knapsack in solution y.
b_{ij}	Equals 1 if and only if item i is placed entirely to the left of item j in solution y.
c_j	The vertical position assigned to item i.
a_{ij}	Equals 1 if and only if item i is placed entirely above item j.

The output from the first stage needs to be processed in order to serve as an input to the second stage. Given a solution y to stage 1, $\mathcal{I}_1$ and b_{ij} can be found. An item i is in $\mathcal{I}_1$ if and only if $y_{iw} = 1$ for some w. Moreover, we have $b_{ij} := 1$ if and only if

$$w_i + \sum_{w=0}^{W-w_i} w y_{iw} \leq \sum_{w=0}^{W-w_j} w y_{jw},$$

so if and only if item i is placed entirely to the left of item j. Items that are not in the knapsack can be ignored in stage 2. The formulation for stage 2 is given in (4).

$$
\begin{aligned}
\max \quad & 0 & & \text{(4a)}\\
a_{ij} + a_{ji} + b_{ij} + b_{ji} & \geq 1 & \forall i \neq j \in \mathcal{I}_1 & \quad \text{(4b)}\\
c_j + L(1 - a_{ij}) & \geq c_i + l_i & \forall i \neq j \in \mathcal{I}_1 & \quad \text{(4c)}\\
c_i & \in \mathbb{Z}_{\geq 0} & \forall i \in \mathcal{I}_1 & \quad \text{(4d)}\\
a_{ij} & \in \{0,1\} & \forall i \neq j \in \mathcal{I}_1 & \quad \text{(4e)}
\end{aligned}
$$

Stage 2 only needs to find a feasible solution, so there is no objective function. Constraint (4b) ensures that for every pair of items $i, j \in \mathcal{I}_1$ one of the items must be either entirely above or to the left of the other (or both). Thus, if neither item is to the left of the other, then one must be above the other. This is enforced through constraint (4c), which limits feasible vertical positions if one item is above the other.

The two stages are then linked through the following process:

1. The solver obtains a feasible solution to stage 1.
2. The solution to stage 1 is taken as an input to stage 2.
3. If stage two is feasible, we have a new incumbent solution to stage 1. Continue solving until optimality of the current solution is proven, or a new solution is found.
4. If stage 2 is infeasible, reject the solution y' by adding

$$\sum_{(i,w):y'_{iw}=1} y_{iw} \leq |\{(i,w) : y'_{iw} = 1\}| - 1 \tag{5}$$

as a constraint to stage 1. Continue solving until optimality of the current solution is proven, or a new solution is found. Note that formally, (5) is part of (3).

3.3.1 The 0-1 Knapsack problem

Stage 1 can be seen as a series of connected 0-1 Knapsack problems, namely one such problem for every w. Therefore, we consider some aspects of the 0-1 Knapsack problem as part of our investigation.

The 0-1 Knapsack problem is a well-known weakly NP-hard problem. The goal of the problem is to maximize the profit gained by placing items in a knapsack with a weight restriction. Hence, not all items can be included in the knapsack. The 0-1 Knapsack problem is the oldest of the many knapsack problems, and has been known for over a century [32]. The 0-1 Knapsack problem can be formulated as an integer linear problem according to (6), where W is the weight capacity of the knapsack, p_i are the profits and w_i are the weights for each item $i \in \{1, ..., n\}$. An item i is included in the knapsack if and only is $x_i = 1$.

$$\max \sum_{i=1}^{n} p_i x_i \tag{6a}$$

$$\text{s.t.:} \quad \sum_{i=1}^{n} w_i x_i \leq W \tag{6b}$$

Many methods have been studied in relation to the 0-1 knapsack problem, but the most common method of solving it is still the dynamic programming approach COMBO introduced by Martello et al. [32, 39]. Another method is the discovery of valid inequalities that can be used as cutting planes in a branch-bound-and-cut approach. One type of valid inequalities for 0-1 Knapsack are *cover inequalities* which were introduced independently by Balas [40], Hammer et al. [41], and Wolsley [42]. A *cover* is a subset of items such that not all items in the cover can fit in the knapsack simultaneously. Hence, if $C \subseteq \mathcal{I}$ is a cover, at most $|C| - 1$ items in C could fit in the knapsack. If C is a *minimal cover*, then any $|C| - 1$ items in C fit in the knapsack. This leads to the so-called *cover inequalities* of the form

$$\sum_{i\in C} d_i x_i \leq |C| - 1, \tag{7}$$

where C is a cover and $d_i = 1$ for all items $i \in C$. Cover inequalities can be quite weak, so they are often strengthened through coefficient lifting [40, 42]. *Lifting* is a procedure by which inequalities that are not facet-inducing can be strengthened until they are facet-inducing. It does this by looking at the x-variables that are not yet in the inequality on a one-by-one basis, and then finding the maximum coefficient for which each variable could be added while maintaining the validity of the inequality for the polytope. This is called the *lifting coefficient.* If the items are lifted sequentially, this process is called *sequential lifting.* Other type of lifting also exist for cover inequalities [43], but these will not be considered in this report. For a cover inequality of 0-1 Knapsack with capacity W, items $i = 1, ..., n$, weights w_i and some cover C, this would mean the lifting coefficient of the first lifted item is given by

$$d_j := |C| - 1 - \max\{\sum_{i\in C} d_i x_i : \sum_{i\in C} w_i x_i \leq W - w_j\}, \tag{8}$$

where d_i are the coefficients of the cover items and $x_i := 1$ if and only if item i is included in the knapsack. The maximization finds the maximum value the left-hand side of the cover inequality can attain; subtracting this from the right-hand side results in the maximum coefficient that maintains validity. For the second and other lifted items, the equation changes slightly, since the inequality now also includes a (possibly) non-zero coefficient for an item not in the cover. Let $\mathcal{L} := C \cup \{i : i \text{ is lifted}\}$, then the lifting coefficient for item $j \notin \mathcal{L}$ is given by

$$d_j := |C| - 1 - \max\{\sum_{i\in \mathcal{L}} d_i x_i : \sum_{i\in \mathcal{L}} w_i x_i \leq W - w_j\}. \tag{9}$$

An example of lifting a cover inequality for 0-1 Knapsack is given below.

Example 3.1. Consider a 0-1 Knapsack instance with 5 items and capacity 10, with the item weights given in Table 10.

TABLE 10: An instance of 0-1 Knapsack with capacity 10 and 5 items with corresponding weights.

items	1	2	3	4	5
weights	4	4	6	3	3

A minimal cover for this instance is $C = \{1, 2, 3\}$, since $4 + 4 = 8 \leq 10$ and $4 + 6 \leq 10$, but $4 + 4 + 6 = 14$ exceeds the capacity. The cover inequality would then be given by

$$x_1 + x_2 + x_3 \leq |C| - 1 = 2. \tag{10}$$

To lift this inequality, we can only look at the remaining items. Suppose we first lift item 4, and then lift item 5. If item 4 is in the knapsack, a capacity of $10 - 3 = 7$ remains. This means that among items 1, 2 and 3, either item 1 or 2 still fits in the knapsack. This means that the coefficient d_4 for item 4 is given by

$$d_4 = 3 - 1 - \max\{x_1 + x_2 + x_3 : 4x_1 + 4x_2 + 6x_3 \leq 7\} = 2 - 1 = 1. \tag{11}$$

The lifting coefficient for item 5 can then be determined by

$$d_5 = 2 - \max\{x_1 + x_2 + x_3 + x_4 : 4x_1 + 4x_2 + 6x_3 + 3x_4 \leq 10 - 3 = 7\} \tag{12}$$
$$= 2 - 2 = 0, \tag{13}$$

because items 1 and 4 (or 2 and 4) still fit in the knapsack if item 5 is included. This also immediately shows that the lifting sequence influences the final lifted inequality. If items 4 and 5 had been lifted in the reverse order (first 5 then 4), item 5 would get $d_5 = 1$, and item 4 would have $d_4 = 0$.

It can easily be seen that computing a lifting coefficient is again a 0-1 Knapsack problem. The dynamic programming approach for 0-1 Knapsack runs in pseudo-polynomial time in the profits. Fortunately, it computes lifting coefficients in polynomial time, because the maximum 'profit' in the lifting problem is the maximum lifting coefficient, which satisfies

$$\max_i d_i \leq |C| - 1 \leq n - 1 < n. \tag{14}$$

Hence, obtaining lifting a cover inequality can be done in $\mathcal{O}(n^2)$. Note that this is only true for simple lifting, as lifting including sequential *up-lifting* and *down-lifting* was proven NP-hard by Chen and Dai [43].

3.3.2 Adjacent cover inequalities

In this subsection, we introduce a novel type of inequality for the first stage of our 2-stage approach (3). Recall that stage 1 can be seen as a series of interconnected 0-1 Knapsack problems. An item i is included in the 0-1 Knapsack problem of slice w if $x_{iw} = 1$, and its weight is simply l_i (with knapsack capacity L). Since cover inequalities are valid for the 0-1 Knapsack problem created for every w, they are also valid for (3).

The new inequalities make use of the connected aspects of these 0-1 Knapsack problems, and can also be generated using the covers of the 0-1 Knapsack problems. We denote these new inequalities as *adjacent cover inequalities.* An adjacent cover inequality looks at two consecutive slices w and $w+1$, which allows for three possibilities: an item is in both slices, in only one of the slices or in neither slice.

Proposition 3.1. *Let C be a cover of the 0-1 Knapsack problem given by profits p_i, weights l_i and capacity L. Let*

$$P^{C,w} := \{(i,v) : i \in C, \max(0, w - w_i + 2) \leq v \leq min(w, W - w_j)\} \tag{15}$$

be the set of all placements such that the item occupies both slice w and $w+1$, and let

$$P_j^{C,w} := \{(j, max(0, w - w_j + 1)), (j, min(w + 1, W - w_j))\} \backslash P^{C,w} \tag{16}$$

be the set of placements of item j such that it occupies only slice w or $w+1$, where $w \in \{0, ..., W-2\}$ and $j \in C$ is a special side item. *Then the* adjacent cover inequalities *given by*

$$\sum_{(i,v)\in P^{C,w}} y_{iv} + \sum_{(j,v)\in P_j^{C,w}} y_{jv} \leq |C| - 1, \tag{17}$$

are valid for the 2D-Knapsack problem.

Proof. If $w - w_j + 1 < 0$ then $(j, 0)$ is already in $P^{C,w}$, because the position that would allow it to only occupy the left slice (indexed by w) is outside the 2D-Knapsack. Similarly, if $w + 1 > W - w_j$ then $(j, W - w_j)$ is in $P^{C,w}$. If either of these is the case, the inequality is a weakened version of the normal cover inequality given in (7) for w or $w+1$ respectively. Hence, we will assume that both positions are feasible; that is, $|P_j^{C,w}| = 2$ holds.

Each item can be included in the knapsack at most once, so at most one y_{iv} can equal 1 for each item i. At most $|C|-1$ items can be included in the Knapsack through the inequality, since any assignment of all items in the cover would exceed the capacity L of slice w or $w + 1$. It is easily seen that any assignment of $|C|$ items purely from $P^{C,w}$ is impossible, because both slices would exceed the capacity. Switching out the assignment for item j to an assignment where it only occupies a single slice instead of both would still exceed the capacity in that slice. Hence, at most $|C| - 1$ of the assignments in the inequality can be set to 1.

□

A small example with 3 items and $W = 5$ is shown in Figure 8. All possible placements occupying at least slice w or $w+1$ are shown. Each placement is characterized by the item and the leftmost slice occupied by the item. For example, item 1 occupies only the left slice w if its leftmost position is 1, both slices if it is 2 and only the right slice $w+1$ if its leftmost position is 3; these placements are referred to as $(1, 1), (1, 2)$, and $(1, 3)$ respectively. It is also possible that it is impossible for an item to be in only the left or right slices, depending on the width of the item. This is the case for item 4, which occupies both slices for all its feasible placements.

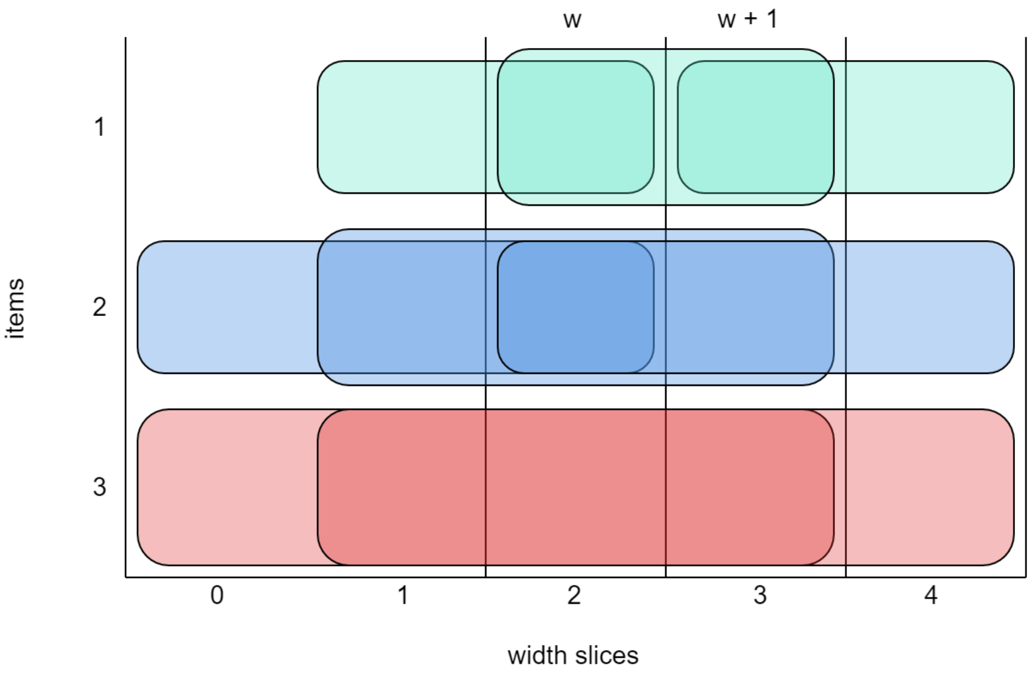


FIGURE 8: A visual representation of a 2D-Knapsack instance with a total width $W = 5$ and 3 items with widths of 2, 3, and 4 respectively. All feasible placements that occupy slice w or $w + 1$ (or both) are displayed as rectangles with the width of the respective item.

Suppose these 3 items make up a cover C and item 1 is the side item. In terms of the notation introduced in Proposition 3.1, we would have $P^{C,2} = \{(1,2),(2,1),(2,2),(3,0),(3,1)\}$ and $P_1^{C,2} = \{(1,1),(1,3)\}$. The corresponding adjacent cover inequality would then be given by

$$y_{12} + y_{21} + y_{22} + y_{30} + y_{31} + y_{11} + y_{13} \leq 2. \tag{18}$$

A property of cover inequalities in 0-1 Knapsack is that the cover inequalities for non-minimal covers are redundant, because they are implied by the minimal cover inequalities [40]. A similar result holds for adjacent cover inequalities.

Proposition 3.2. *Let C be a (non-minimal) cover and let C' be a minimal cover contained in C with side item $j \in C'$. Then adjacent cover inequalities* (17) *generated by C are implied by those generated by C'.*

Proof. The inequalities generated by C' are of the form

$$\sum_{(i,v)\in P^{C',w}} y_{iv} + \sum_{(j,v)\in P_j^{C',w}} y_{jv} \leq |C'| - 1. \tag{19}$$

Since y must be a feasible solution to (3), it must satisfy (3b), which implies that

$$\sum_{v=\max(0,w-w_i+2)}^{\min(w,W-w_i)} y_{iv} \leq 1 \qquad \forall i \in \mathcal{I}, \tag{20}$$

and

$$\sum_{(i,v)\in P^{C\backslash C',w}} y_{iv} \leq |C\backslash C'|, \tag{21}$$

also hold, where (21) is obtained from (20) by summing it over all $i \in C\backslash C'$. Adding this to (19) gives

$$\sum_{(i,v)\in P^{C',w}} y_{iv} + \sum_{(i,v)\in P^{C\backslash C',w}} y_{iv} + \sum_{(j,v)\in P_j^{C',w}} y_{jv} \leq |C'| - 1 + |C\backslash C'|, \tag{22}$$

which is simply the same as the inequality generated by C. Hence, cutting planes for non-minimal covers are redundant, because they are implied by those for minimal covers. □

Cutting planes generated by inequalities can be said to be strong if they are facet-inducing for the convex hull of all (integer) vectors satisfying (3). We only care about the projection onto the y-variables, because the x-variables follow directly from them via (3d). Thus, we consider polytope $P = conv(Y)$, where

$$Y = \{y \in \{0,1\}^{N\times W} : y \text{ is feasible for (3)}\}. \tag{23}$$

The cutting planes generated by(17) are only facet-inducing for P under limited conditions. This is shown in Proposition 3.4, the proof of which uses Proposition 3.3.

Proposition 3.3. *The adjacent cover inequalities generated by a minimal cover C are facet-inducing for $conv(Y^C)$, where Y^C is the set of $y \in \{0,1\}^{|C|\times W}$ such that*

$$\sum_{w=1}^{W-w_i} y_{iw} \leq 1 \qquad \forall i \in C \tag{3b}$$

$$\sum_{i\in C} l_i \sum_{v=\max(0,w-w_i+1)}^{w} y_{iv} \leq L \qquad \forall w \in \{0,...,W-1\}. \tag{3e}$$

Proof. Let F be the face induced by (17) for some $j \in C$ and $w \in \{0,...,W-2\}$. Let H be a facet defined by $d^T y \leq \gamma$ such that $F \subseteq H$.
We first show that the coefficients d_{iv} for $(i,v) \in P^{C,w} \cup P_j^{C,w}$ cannot be increased. This follows immediately, because any assignment of $|C|-1$ items in C is feasible for Y^C, and any increase would result in a feasible y to violate (17).
It remains to show that $d_{iv} = 0, i \in C, i \neq j$ for $(i,v) \notin P^{C,w}$. These are item-position combinations where item i occupies neither slice w nor $w+1$, or combinations where only one of the two is occupied if $i \neq j$. Let $\hat{y}^1,...,\hat{y}^4$ be solutions to the LP relaxation that lie in F constructed in the following way, where e_{iv} is the unit vector with its 1-entry at (i,v):

- $\hat{y}^1$ is a solution where exactly $|C|-1$ items item-position combinations in $P^{C,w} \cup P_j^{C,w}$ are selected, and no other items are included in the knapsack. Let the set of items in the knapsack be called $C_{\hat{y}}$
- $\hat{y}^2 := \hat{y}^1 + e_{i'v}$ is solution $\hat{y}^1$ where the remaining item i' is assigned to any feasible location v in the knapsack.

Since these solutions are all in F (and thus also in H), we have that $d^T\hat{y}^k = \gamma$ for $k = 1,2$. Taking the difference of these equations gives us

$$0 = d^T(\hat{y}^1 - \hat{y}^2) = d^T(-e_{i'v}) = -d_{i'v} \qquad \forall i' \notin C_{\hat{y}}, \quad \forall \text{ feasible } v. \tag{24}$$

If side item j is only in slice $w+1$ for some $\hat{y}$, then positions $v = 0,...,w-w_{i'}+1$ are feasible for the remaining item i'. Similarly, if j is only in slice w, positions $v = w+1,...,W-w_{i'}$ are feasible. This gives us that $d_{iv} = 0$ for $(i,v) \notin P^{C,w} \cup P_j^{C,w}$, since all those positions are feasible if the other $|C|-1$ items are assigned positions for which $(i,v) \in P^{C,w}, i \neq j$ and $(j,v) \in P_j^{C,w}$.

□

Proposition 3.4. *The adjacent cover inequalities generated by a minimal cover with side item j are facet-inducing for P if and only if for any item $i \notin C$ with $w_i > 1$ there is a set $C_i \subset C$ with $|C_i| = |C|-1$ such that*

$$l_i + \sum_{k\in C_i} l_k \leq L, \tag{25}$$

and if for any $i \notin C$ with $w_i = 1$ there is a set $C_i \subset C$ with $j \notin C_i, |C_i| = |C|-2$ such that

$$l_i + \sum_{k\in C_i} l_k \leq L, \tag{26}$$

Proof. The proof is given in Appendix C □

3.3.3 Lifting

Due to the very limited conditions for which the inequalities given in (17) are facet-inducing, it is important to consider lifting the inequalities to make them facet-inducing. The normal cover inequalities (7) can be lifted in $\mathcal{O}(n^2)$ through dynamic programming according to (9). Unfortunately, we could not find a polynomial time algorithm for the lifting of the adjacent cover inequalities.

The adjacent cover inequalities that are not facet-inducing can be lifted to become facet-inducing in a similar manner as cover inequalities with pseudo-polynomial time dynamic programming algorithms. Since the lifting only deals with whether an item occupies the left, right or both slices, all y_{iw} assignments that would cause item i to occupy both slices would get the same lifting coefficient. Hence, they can be grouped together as

$$y_{i,both} := \sum_{v=w-w_i+2}^{w} y_{iv}. \tag{27}$$

The left and right slices correspond to slice w and $w+1$ respectively. This means that

$$y_{i,left} := y_{i(w-w_i+1)} \tag{28}$$

and

$$y_{i,right} := y_{i(w+1)}. \tag{29}$$

Contrary to the lifting problem for 0-1 Knapsack where only a single capacity constraint needs to be satisfied to maintain validity of the cover inequality, the lifting problem for the adjacent cover inequalities needs to satisfy two capacity constraints and the requirement that each item can only be assigned to a single position. Let $\mathcal{L}$ be the set of items that are already lifted (this includes the items in the cover), and let j be the next item to lift. Then the lifting coefficients for item j can be found by solving

$$\max \quad \sum_{i\in\mathcal{L}} d_i^{left} y_{i,left} + d_i^{right} y_{i,right} + d_i^{both} y_{i,both} \tag{30a}$$

$$\text{s.t.:} \quad y_{i,left} + y_{i,right} + y_{i,both} \leq 1 \qquad \forall i \in \mathcal{L} \tag{30b}$$

$$\sum_{i\in\mathcal{L}} l_i(y_{i,left} + y_{i,both}) \leq L_{left} \tag{30c}$$

$$\sum_{i\in\mathcal{L}} l_i(y_{i,right} + y_{i,both}) \leq L_{right} \tag{30d}$$

$$y_{i,left}, y_{i,right}, y_{i,both} \in \{0,1\}^{|\mathcal{L}|}, \tag{30e}$$

where l_i is the length of item i and $d_{i,*}$ are the lifting coefficients for if item i is placed at a position where it only occupies the left slice, only the right slice, or both slices. L_{right} and L_{left} represent the total amount of space left in the right and left slice respectively after item j has been included. If item j is only in the left slice, $L_{left} = L - l_j$ while $L_{right} = L$. If it is only in the right slice, $L_{left} = L$ while $L_{right} = L - l_j$, and if it is in both then $L_{left} = L - l_j$ and $L_{right} = L - l_j$. The lifting coefficients for item j are then given by

$$\begin{aligned} d_j^{left} &= |C| - 1 - OPT_{left} \\ d_j^{right} &= |C| - 1 - OPT_{right} \\ d_j^{both} &= |C| - 1 - OPT_{both} \end{aligned}$$

for the optimal objective values to their respective versions of (30). The d-values obtained in such a manner are restricted in several ways. Firstly, $d_i^{left}, d_i^{right} \leq d_i^{both}$, because any solution that is feasible for lifting $y_{i,both}$ is also feasible for lifting $y_{i,left}$ and $y_{i,right}$. Secondly, the objective of (30) is always between 0 and $|C| - 1$, so the same holds for the d-values. Lastly, every lifting coefficient is obtained through consecutive iterations of (30), and is thus dependent on all previously lifted variables. An example is given below.

Example 3.2. Consider the 2D-Knapsack problem with 5 items, a total width of 5 and a length of 10, with the item dimensions given in Table 11.

TABLE 11: An instance of 2D-Knapsack with $W = 5$ and $L = 10$.

items	1	2	3	4	5
lengths	4	4	4	7	3
widths	2	2	2	2	2

A minimal cover for this instance is $\{1, 2, 3\}$, since $4 + 4 \leq 10$ and $4 + 4 + 4 > 10$. Let item 1 be the side item, and let $w = 2$. Suppose we first lift all variables related to item 4 ($y_{4,left}, y_{4,both}, y_{4,right}$) for which item 4 occupies slice w or $w + 1$. If item 4 is placed such that it only occupies the right (left) slice, we get that we can place item 1 in the left (right) slice. No other items fit, since $7 + 4 > 10$. This gives us $d_4^{left} = d_4^{right} = 2 - 1 = 1$. No items fit if item 4 occupies both slices, resulting in $d_4^{both} = 2$. We now update $\mathcal{L} := C \cup \{i : i \text{ is lifted}\} = \{1, 2, 3, 4\}$. The only remaining item to lift is item 5. If it is placed in both slices, we can still add item 4 in both slices. Hence, $d_5^{both} = 2 - 2 = 0$. It then immediately follows that $d_5^{left} = d_5^{right} = 0$.

Recall that the lifting problem for the 0-1 Knapsack problem is itself a 0-1 Knapsack problem. It was solvable in polynomial time due to a restriction on the maximum profit. We suspect that (30) is not solvable in polynomial time. A proof is given for a generalization of the problem in Proposition 3.5.

Proposition 3.5. *The mixed integer linear program* (30) *is NP-hard when the lifting coefficients are only restricted to*

$$d_i^{left}, d_i^{right} \leq d_i^{both} \qquad \forall i, \qquad (31)$$

$$0 \leq d_i^{both} \leq |C| - 1 \qquad \forall i, \qquad (32)$$

the objective is at most $|C| - 1$, and C is a non-minimal cover.

Proof. The problem is NP-hard, because a polynomial time reduction from Subset Sum exists, which is NP-hard. Let $\mathcal{I}$ be an instance of Subset Sum with $a_i \in \mathbb{Z}^+$ for $i = 1, ..., n$ and $0 < S \leq \frac{1}{2} \sum_{i \in V} a_i$, then $\mathcal{I}$ is a "Yes" instance if there exists a subset $J \subseteq \{1, ..., n\}$ such that $\sum_{i \in J} a_i = S$. Then the corresponding instance $\phi(\mathcal{I})$ for (30) is given by

$$\begin{aligned}
l_i &= a_i && \forall i \in \{1, ..., n\}, \\
d_i^{both} &= 1 && \forall i \in \{1, ..., n\}, \\
d_i^{left} &= 1 && \forall i \in \{1, ..., n\}, \\
d_i^{right} &= 1 && \forall i \in \{1, ..., n\}, \\
L_{left} &= S, \\
L_{right} &= \sum_{i \in V} a_i,
\end{aligned}$$

and an additional $k \geq n$ items with $l_{j+n} \geq \max_i a_i$ for $j = 1,..,k$ and $d_{j,*}$ according to (17). The adjacent cover inequality is satisfied for the additional items, because

$$\sum_{j=1}^{k} l_{j+n} \geq \sum_{j=1}^{n} l_{j+n} \geq \sum_{i \in V} a_i > L_{right} \geq L_{left}.$$

Suppose $\mathcal{I}$ is a "Yes" instance for Subset Sum, then there exists a J such that $\sum_{i \in J} a_i = S$. Choosing $y_{i,left} = 1$ for $i \in J$, $y_{i,right} = 1$ for $i \in \{1,..,n\} \backslash J$, and setting all other variables to 0 is a feasible solution for the lifting problem with an objective of n.

Suppose $\phi(\mathcal{I})$ has an objective of n, but $\mathcal{I}$ is not a "Yes" instance for Subset Sum. Then all optimal solutions to the lifting problem must use at least one item $j + n$ with $j = 1, ..., k$. This also means that there is an equal number of items not used with $i = 1, ..., n$. However, $l_{j+n} \geq l_i$ for $j = 1, ..., k, i = 1, ..., n$, so any used item $j + n$ can be replaced by an unused item i and maintain feasibility. Replacing all items $j + n$ in such a manner gives a solution using exactly all items $i = 1, ..., n$ with an objective of n, making it an optimal solution too. This is a contradiction, so $\mathcal{I}$ must be a "Yes" instance for Subset Sum.

Thus, an instance $\mathcal{I}$ is a "Yes" instance for Subset Sum if and only if the lifting problem for $\phi(\mathcal{I})$ has an objective of n, which means that the lifting problem is at least as hard as Subset Sum. Thus, the lifting problem is NP-hard.

□

It is important to note that the lifting problem in Proposition 3.5 is different from the actual lifting procedure in two ways. Firstly, the inequalities lifted by the lifting procedure are only facet-defining if the initial cover is a minimal cover, whereas the proof of Proposition 3.5 uses a non-minimal cover. Secondly, lifting an inequality requires repetitive application of the lifting problem to find all the lifting coefficients. The chosen d and l in Proposition 3.5 do not satisfy this property. Hence, even if $P \neq NP$, a polynomial time algorithm for the actual lifting procedure might exist. However, it would have to make use of properties of either a minimal cover or the relation between d and l.

3.3.4 Lifting sequences

The sequence in which the items are lifted influences the resulting lifted inequality, just like what happens with lifted cover inequalities. If all variables for item 5 were lifted before item 4 in Example 3.2, we would get $d_5^{both} = 1, d_5^{left} = d_5^{right} = 0$ and $d_4^{both} = 1, d_4^{left} = d_4^{right} = 0$, because item 5 can occupy both slices together with item 4. If item 4 only occupies 1 slice, item 1 can be placed in the other slice. Hence, it is important to consider in which sequence the variables are lifted. We consider 2 types of lifting sequences.

1. Lift per item: lift all variables of an item before lifting the next item.
2. Lift per variable: lift *both* variables for all items before lifting the *left/right* variables or vice versa.

Within each type of lifting sequence, the order in which the items are lifted is also part of the lifting sequence. We consider three item orders: random, by increasing length, and by decreasing length. This results in a total of 9 lifting sequences.

Not only the lifting sequence, but also how the lifting procedure is incorporated into the solver has influence on the result. One method is to find minimal covers, determine the corresponding adjacent cover inequalities, and lift the inequalities before solving the MILP. The lifted inequalities can then be added directly as cuts or constraints, or through a separator that adds (a subset of) violated lifted inequalities. This method works well for small instances, but for larger instances the number of minimal covers grows exponentially, so lifting all the adjacent cover inequalities would take too much time. Also, this method only allows lifting sequences that are independent of the LP relaxation solution. The second method does not have these drawbacks. For normal cover inequalities, it relies on a separation procedure to find a violated cover inequality, reduces it to a minimal cover, and then lifts the cover inequality related to the minimal cover [27]. This method does not lift or find unnecessary inequalities, but it does require the additional step of finding a violated inequality. Additionally, it may miss covers for which the original cover inequality is not violated, but the lifted cover inequality is violated. In this report, we apply both methods to see the influence of the drawbacks and advantages on the LP relaxation in the root node when added through a separator, and when all inequalities are added directly.

Lift per item

We have found three pseudo-polynomial time algorithms with different time complexities for 'per item' lifting sequences. Which algorithm is better depends on the knapsack size and instance size.

The first dynamic program ALGORITHM1 runs in $\mathcal{O}(nL^2)$. The time complexity comes from the fact that an $(L+1)$-by-$(L+1)$ matrix needs to be updated n times. Each update requires constant time, so the time complexity is $\mathcal{O}(nL^2)$.

The first algorithm works by making a series of packing decisions. The algorithm takes an adjacent cover inequality as input in the form of a minimal cover and its side item, as well as the set of items, their lengths, and the total length of the knapsack. Let $l, r \in \{0, ..., L\}$ denote the amount of occupied space in the left and right slice respectively for some packing of items in $\mathcal{L}$. First the algorithm finds the maximum value k that can be obtained for each l, r pair using the items in the cover. This can be done by ordering the items in cover by increasing length; the maximum achievable value is then given by the maximum k for which the k smallest cover items fit in l, r.

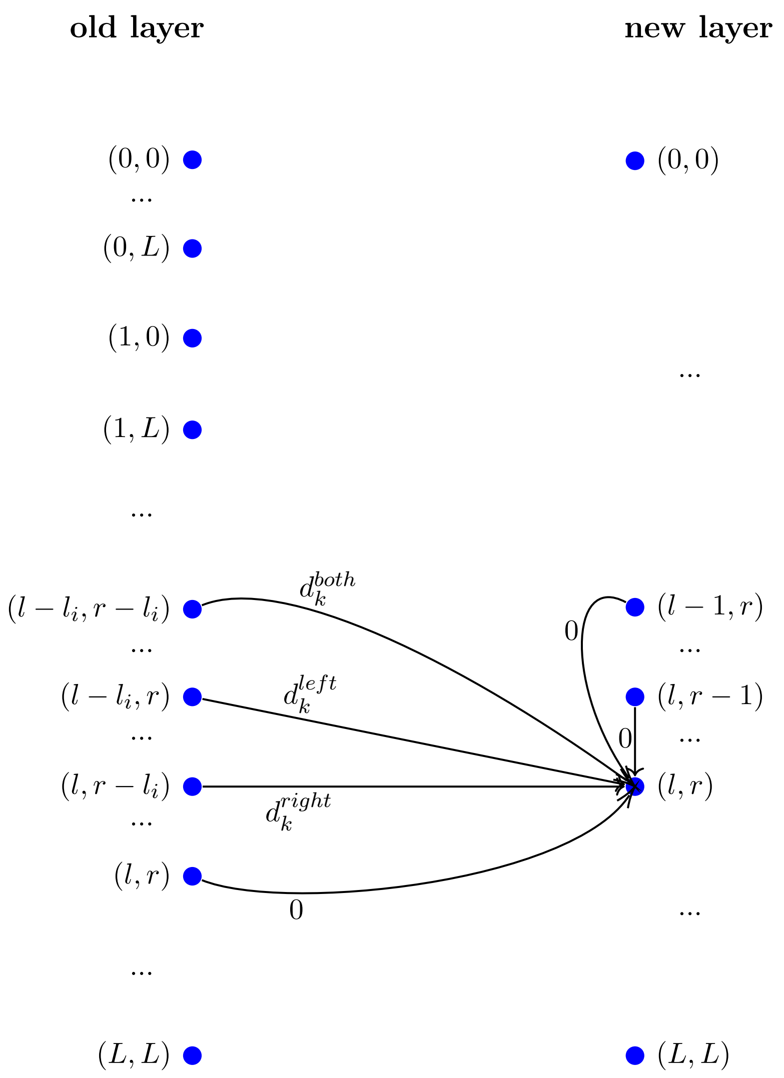


FIGURE 9: A graphical representation of Line 25 in ALGORITHM1 where the old layer corresponds to V^{old} and the new layer to V^{new} where the item i is lifted. The arrows are only drawn in for updating (l,r), but the same arrows exist for all other nodes in the new layer (provided that the needed nodes in the old layer exist).

For the items that need to be lifted, the d-value follows directly from the maximum value achievable when using all previously lifted items and the cover if the lifted item occupies the left side, right side of both sides respectively. The maximum values are then updated for each l, r if the newly lifted item can be freely chosen to be included or excluded. This is done in line 25. The new value is a maximum over 6 options, The first four options correspond to not including j in the knapsack, including it only in the left slice, including it only in the right slice, and including it in both slices respectively. The latter two ensure that the formula for $d_{j,*}$ are correct. If $l - l_j$ or $r - l_j$ is less than 0, then these options must be ignored. If $l_j = 1$, then it obviously cannot be in both slices, so this option can be ignored. A graphical representation is shown in Figure 9, where it can be seen that ALGORITHM1 solves the longest path problem from $(0,0)$ to any other node if arcs are according to Figure 9 for all nodes and all n layers.

```
Function: ALGORITHM 1($C, j, l, \mathcal{I}, L$)
Input: Minimal cover $C$, special item $j$, item lengths $l = (l_1, ..., l_n)$, items $\mathcal{I}$, and
       knapsack length $L$
Output: The coefficient vector $d$ of the lifted cover inequality
1  Let $\mathcal{L} = \{0, 1, ..., L\}$.
2  Let $V^{old}_{l,r} = 0 \ \forall l, r \in \mathcal{L}$ be the maximum value achievable when using 0 items.
3  $V^{new} \leftarrow V^{old}$
4  $\mathcal{I} \leftarrow \mathcal{I} \backslash C$
5  for $i \in C$ do
6      if $i = j$ then
7          $d^{left}_i = d^{right}_i = d^{both}_i = 1$
8      else
9          $d^{both}_i = 1$
10         $d^{left}_i = d^{right}_i = 0$
11     end
12     for $l \in \mathcal{L}$ do
13         for $r \in \mathcal{L}$ do
14             $V^{new}_{l,r} \leftarrow$
               $\max\left(V^{old}_{l,r}, V^{old}_{l-l_i,r} + d^{left}_i, V^{old}_{l,r-l_i} + d^{right}_i, V^{old}_{l-l_i,r-l_i} + d^{both}_i, V^{new}_{l-1,r}, V^{new}_{l,r-1}\right)$
               // ignore term if index is negative
15         end
16     end
17     $V^{old} \leftarrow V^{new}$
18 end
19 for $i \in \mathcal{I}$ do
20     $d^{left}_i = |C| - 1 - V^{old}_{L-l_i,L}$
21     $d^{right}_i = |C| - 1 - V^{old}_{L,L-l_i}$
22     $d^{both}_i = |C| - 1 - V^{old}_{L-l_i,L-l_i}$
23     for $l \in \mathcal{L}$ do
24         for $r \in \mathcal{L}$ do
25             $V^{new}_{l,r} \leftarrow$
               $\max\left(V^{old}_{l,r}, V^{old}_{l-l_i,r} + d^{left}_i, V^{old}_{l,r-l_i} + d^{right}_i, V^{old}_{l-l_i,r-l_i} + d^{both}_i, V^{new}_{l-1,r}, V^{new}_{l,r-1}\right)$
               // ignore term if index is negative
26         end
27     end
28     $V^{old} \leftarrow V^{new}$
29 end
30 return $d$
```

The second dynamic program runs in $\mathcal{O}(n^2 L)$. It works the same way as the first dynamic program, except that it interchanges the value stored with the amount of space occupied in the right slice. This changes the matrix into an L by $|C| - 1$ matrix, which still requires n updates, thus resulting in a time complexity of $\mathcal{O}(n^2 L)$.

Let l, r, j be as in the first dynamic program and let k be a value in $0, ..., |C| - 1$. The algorithm has the same inputs as ALGORITHM1. First, the algorithm finds the minimal

r for which each k value can be achieved for a given l using only the items in the cover. Then, the algorithm lifts the remaining items. The d-values for the item to lift follow from the maximum k for which the items needed to reach it still fit in the knapsack. In other words, the maximum k for which $r \leq L$ or $r \leq L - l_i$ depending on whether the lifted item is in the right slice only or not.

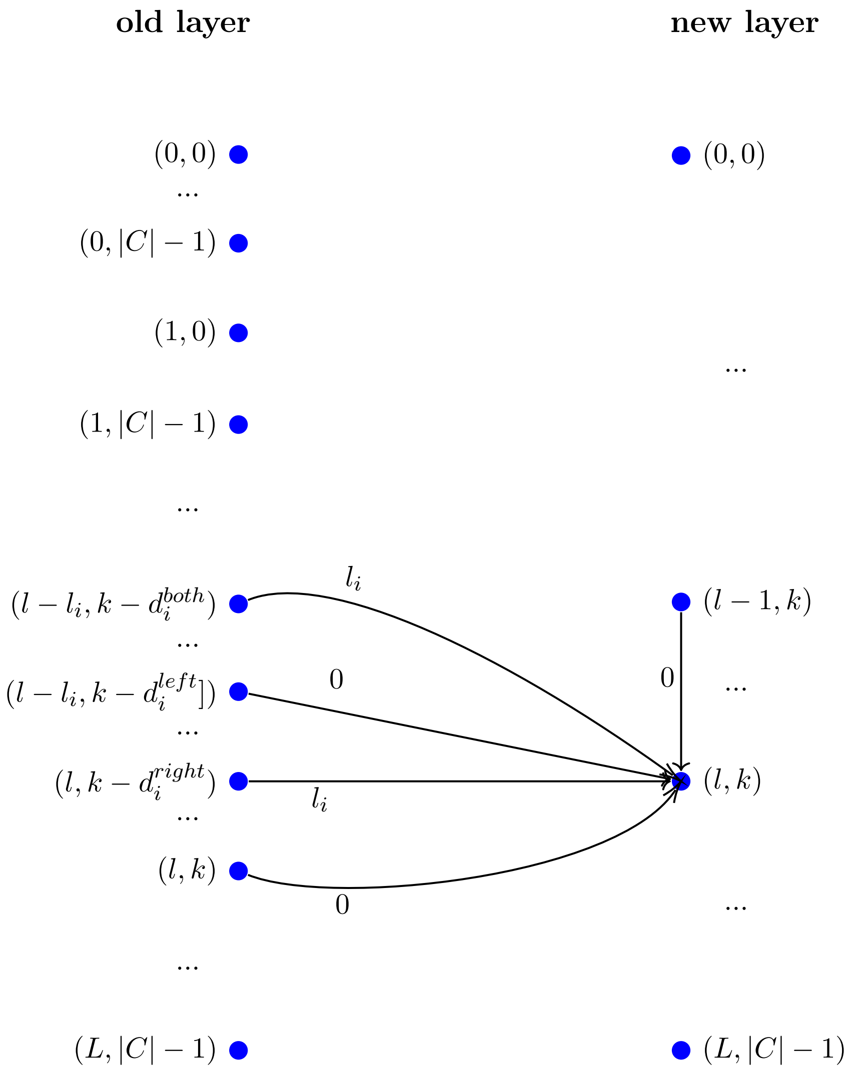


FIGURE 10: A graphical representation of line 25-26 in ALGORITHM2 where the old layer corresponds to V^{old} and the new layer to V^{new} where the item i is lifted. The arrows are only drawn in for updating (l,k), but the same arrows exist for all other nodes in the new layer (provided that the needed nodes in the old layer exist).

The minimum r needed to be able to reach a value of k for a given l is then updated if the lifted item can be freely chosen to be included or excluded. The first four options correspond to not including j in the knapsack, including it only in the left slice, including it only in the right slice, and including it in both slices. The last option simply ensures that if $V_{l',k} = r$ for some l', k, then this r is also a feasible option for $l > l'$. This allows for determining the lifting coefficients directly from the largest feasible l. If $l - l_i$ or $k - d_{i,*}$ is less than 0, then these V must be ignored. If $l_i = 1$, then it cannot be in both slices, so this option can be disregarded. A graphical representation is given in Figure 10. ALGORITHM2 can be seen as a shortest path from $(0,0)$ to any other node in the graph given by Figure 10 with all arcs and all n layers.

```
    **Function:** ALGORITHM 2$(C, j, l, \mathcal{I}, L)$
    **Input:** Minimal cover $C$, special item $j$, item lengths $l = (l_1, ..., l_n)$, items $\mathcal{I}$, and
            knapsack length $L$
    **Output:** The coefficient vector $d$ of the lifted cover inequality
 1  Let $\mathcal{L} = \{0, 1, ..., L\}$.
 2  Let $V^{old}_{l,k} = \infty \ \forall l \in \mathcal{L}, k \in \{1, ..., |C| - 1\}$ be the minimum $r$ for which the value $k$
      can be achieved.
 3  Let $V^{old}_{l,0} = 0 \ \forall l \in \mathcal{L}$
 4  $V^{new} \leftarrow V^{old}$
 5  $\mathcal{I} \leftarrow \mathcal{I} \backslash C$
 6  **for** $i \in C$ **do**
 7      **if** $i = j$ **then**
 8          $d^{left}_i = d^{right}_i = d^{both}_i = 1$
 9      **else**
10          $d^{both}_i = 1$
11          $d^{left}_i = d^{right}_i = 0$
12      **end**
13      **for** $l \in \mathcal{L}$ **do**
14          **for** $k \in \{1, ..., |C| - 1\}$ **do**
15              $V^{old}_{l,k} = \min\left(V^{old}_{l,k}, V^{old}_{l-l_i,k-d^{left}_i}, V^{old}_{l,k-d^{right}_i} + l_i, V^{old}_{l-l_i,k-d^{both}_i} + l_i, V^{new}_{l-1,k}\right)$
                 // ignore term if index is negative
16          **end**
17      **end**
18      $V^{old} \leftarrow V^{new}$
19  **end**
20  **for** $i \in \mathcal{I}$ **do**
21      $d^{left}_i = |C| - 1 - \max\{k : V^{old}_{L-l_i,k} \leq L\}$
22      $d^{right}_i = |C| - 1 - \max\{k : V^{old}_{L,k} \leq L - l_i\}$
23      $d^{both}_j = |C| - 1 - \max\{k : V^{old}_{L-l_i,k} \leq L - l_i\}$
24      **for** $l \in \mathcal{L}$ **do**
25          **for** $k \in \{1, ..., |C| - 1\}$ **do**
26              $V^{old}_{l,k} = \min\left(V^{old}_{l,k}, V^{old}_{l-l_i,k-d^{left}_i}, V^{old}_{l,k-d^{right}_i} + l_i, V^{old}_{l-l_i,k-d^{both}_i} + l_i, V^{new}_{l-1,k}\right)$
                 // ignore term if index is negative
27          **end**
28      **end**
29      $V^{old} \leftarrow V^{new}$
30  **end**
31  **return** $d$
```

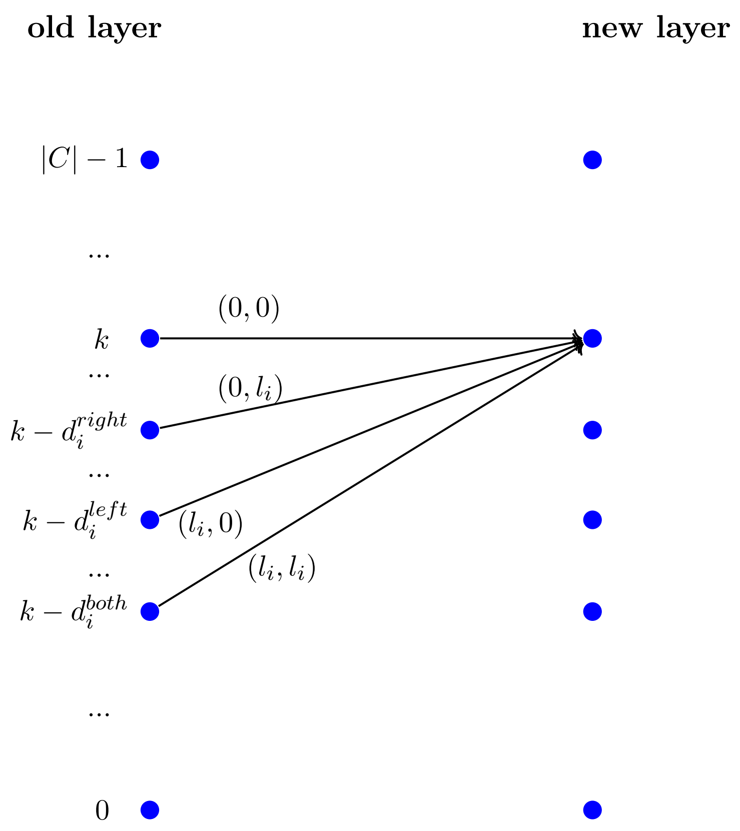


FIGURE 11: A graphical representation of lines 15-23 in ALGORITHM3 where the old layer corresponds to V^{old} and the new layer to V^{new} where the item i is lifted. The arrows are only drawn in for updating k, but the same arrows exist for all other nodes in the new layer (provided that the needed nodes in the old layer exist). The values next to an arc indicate the additional space that will be occupied in the left or right slice. The leftmost number is the increase in the left slice, and the other is the increase in the right slice.

The third dynamic program can be seen as an extension to the normal 0-1 Knapsack cover lifting dynamic program. Instead of just finding the smallest amount of weight needed to reach a value of k with previously lifted items, this dynamic program finds all *non-dominated pairs* (l, r). A pair (l, r) is dominated if there exists a pair (l', r') with $l' \leq l, r' \leq r$ such that node i, k can be reached by occupying l' and r' space in the left and right slices respectively.

Non-dominated pairs can be found either with NONDOMINATED(A, V_k^{new}) or with NONDOMINATED2$(A \cup V_k^{new})$. The former compares each candidate $(a, b) \in A$ with each of the (l, r) pairs already in V_k^{new}. If (a, b) is dominated, it is discarded. If instead it dominates (l, r), then (l, r) is removed from V_k^{new} and (a, b) is added to V_k^{new}. The latter method is a recursive method similar to merge-sort [44]. Since the starting set A is sorted by decreasing l, no (l_i, r_i) for $i = 1, .., p$ can dominate a pair with $i = p+1, ..., m$. Hence, we need to find which pairs in the former set are dominated by a pair in the latter set. We know that r_i strictly increases with i, because both sets consist of sorted non-dominated pairs. This means that any pair with $i = 1, ..., p$ that is dominated by a pair with $i = p+1, ..., m$ must be dominated by (l_{p+1}, r_{p+1}). Hence, as soon as $r_i \geq r_{p+1}$ for $i \leq p$, we know that all the remaining pairs in Q are dominated by (l_{p+1}, r_{p+1}).

```
Function: NONDOMINATED($A, V_k^{new}$)
Input: New set of candidate $(l, r)$ pairs $A$, and the current set of pairs $V_k^{new}$
Output: The updated set of non-dominated $(l, r)$ pairs
for $(a, b) \in A$ do
    for $(l, r) \in V_k^{new}$ do
        if $a >= l$ and $b >= r$ then
            break
        else if $a \leq l$ and $b \leq r$ then
            $V_k^{new} \leftarrow V_k^{new} \backslash (l, r)$
        end
    end
    if $V_k^{new}$ was set then
        $V_k^{new} \leftarrow V_k^{new} \cup (a, b)$
    end
end
return $V_k^{new}$
```

```
Function: NONDOMINATED2($A$)
Input: Complete set of candidate $(l, r)$ pairs $A$ sorted by decreasing $l$.
Output: The updated set of non-dominated $(l, r)$ pairs
if $|A| = 1$ then
    return $A$
else
    $\{(l_1, r_1), (l_2, r_2), ..., (l_p, r_p)\} \leftarrow$ NONDOMINATED2$\{A[1], ..., A[\lfloor \frac{n}{2} \rfloor]\}$
    $\{(l_{p+1}, r_{p+1}), ..., (l_m, r_m))\} \leftarrow$ NONDOMINATED2$\{A[\lfloor \frac{n}{2} \rfloor + 1], ..., A[n]\}$
    while $i \leq p$ and $r_i < r_{p+1}$ do
        $i \leftarrow i + 1$
    end
    return $\{(l_1, r_1), ..., (l_{i-1}, r_{i-1}), (l_{p+1}, r_{p+1}), ..., (l_m, r_m)\}$
end
```

```
Function: ALGORITHM 3$(C, j, l, \mathcal{I}, L)$
Input: Minimal cover $C$, special item $j$, item lengths $l = (l_1, ..., l_n)$, items $\mathcal{I}$, and
       knapsack length $L$
Output: The coefficient vector $d$ of the lifted cover inequality
Let $\mathcal{L} = \{0, 1, ..., L\}$.
Let $V_k = \emptyset\ \forall k \in \{1, ..., |C| - 1\}$.
Let $V_0 = \{(0, 0)\}$.                    // Only $k = 0$ is possible with no items.
$\mathcal{I} \leftarrow \mathcal{I} \backslash C$
for $i \in C$ do
    if $i = j$ then $d_i^{left} = d_i^{right} = d_i^{both} = 1$
    else $d_i^{both} = 1$, $d_i^{left} = d_i^{right} = 0$

    for $k \in \{|C| - 1, ..., 1\}$ do
        $A = \emptyset$
        if $k - d_i^{left} \geq 0$ and $d_i^{left} > 0$ then $A \leftarrow A \cup (l + l_i, r)$ for $(l, r) \in V_{k-d_i^{left}}$
        if $k - d_i^{right} \geq 0$ and $d_i^{right} > 0$ then $A \leftarrow A \cup (l, r + l_i)$ for $(l, r) \in V_{k-d_i^{right}}$
        if $k - d_i^{both} \geq 0$ and $d_i^{both} > 0$ then
          $A \leftarrow A \cup (l + l_i, r + l_i)$ for $(l, r) \in V_{k-d_i^{both}}$
        $V_k \leftarrow$ NONDOMINATED$(A, V_k)$
    end
end
for $i \in \mathcal{I}$ do
    $d_i^{left} = |C| - 1 - \max\{k : \exists (l, r) \in V_k$ s.t. $l \leq L - l_i\}$
    $d_i^{right} = |C| - 1 - \max\{k : \exists (l, r) \in V_k$ s.t. $r \leq L - l_i\}$
    $d_i^{both} = |C| - 1 - \max\{k : \exists (l, r) \in V_k$ s.t. $r, l \leq L - l_i\}$
    for $k \in \{|C| - 1, ..., 1\}$ do
        $A = \emptyset$
        if $k - d_i^{left} \geq 0$ and $d_i^{left} > 0$ then $A \leftarrow A \cup (l + l_i, r)$ for $(l, r) \in V_{k-d_i^{left}}$
        if $k - d_i^{right} \geq 0$ and $d_i^{right} > 0$ then $A \leftarrow A \cup (l, r + l_i)$ for $(l, r) \in V_{k-d_i^{right}}$
        if $k - d_i^{both} \geq 0$ and $d_i^{both} > 0$ then
          $A \leftarrow A \cup (l + l_i, r + l_i)$ for $(l, r) \in V_{k-d_i^{both}}$
        $V_k \leftarrow$ NONDOMINATED$(A, V_k)$
    end
end
return $d$
```

The fourth algorithm is the same as ALGORITHM3, except it uses NONDOMINATED2$(A \cup V_k)$ instead of NONDOMINATED(A) to find the non-dominated pairs.

Lift per variable

Contrary to algorithms that lift per item, algorithms that lift per variable cannot use the concept of the 'old' and 'new' layer to easily update a matrix that can be used for the next item to lift, because not all variables of an item are considered at the same time. The first time a variable of item i is lifted can be done in the same manner as 'per item' algorithms, but the second time cannot. This is because such an algorithm does not store what specific items were placed where in the packing that obtained value k with l, r space occupied; it only stores l, r and k. Using this same strategy for lifting 'per variable' could lead to using an item twice, thus achieving a higher k than is actually possible for l, r.

We investigate two types of lifting 'per variable': lifting $y_{i,both}$ first for all items, and then lifting $y_{i,right}$ and $y_{i,left}$, and vice versa. The algorithms AlgorithmBoth and AlgorithmSides presented in this section are based on Algorithm3, and are identical up to line 19 of Algorithm3. After this, the algorithms split the for loop over $\mathcal{I}$ into two parts. In AlgorithmBoth the $y_{i,both}$ are lifted in the same manner as in Algorithm3. After $y_{i,both}$ has been lifted for all items, $y_{i,left}$ and $y_{i,right}$ are lifted. To do so, the maximum value k that can be obtained using all items except for i with their currently known lifting coefficients is needed. This is done through the auxiliary function FindMax, which finds all non-dominated pairs l, r for which a value of k can be obtained for a given set of items. The lifting coefficients are then found in the same manner as in Algorithm3.

```
   Function: AlgorithmBoth($C, j, l, \mathcal{I}, L$)
   Input: Minimal cover $C$, special item $j$, item lengths $l = (l_1, ..., l_n)$, items $\mathcal{I}$, and
          knapsack length $L$
   Output: The coefficient vector $d$ of the lifted cover inequality
 1 Let $\mathcal{L} = \{0, 1, ..., L\}$.
 2 Let $V_k = \emptyset \ \forall k \in \{1, ..., |C| - 1\}$.
 3 Let $V_0 = \{(0, 0)\}$.                  // Only $k = 0$ is possible with no items.
 4 $\mathcal{I} \leftarrow \mathcal{I} \backslash C$
 5 for $i \in C$ do
 6     if $i = j$ then $d_i^{left} = d_i^{right} = d_i^{both} = 1$
 7     else $d_i^{both} = 1$, $d_i^{left} = d_i^{right} = 0$
 8 end
 9 $V \leftarrow$ FindMax($C, l, d, C, V, L$)
10 $V^{cover} \leftarrow V$
11 for $i \in \mathcal{I}$ do
12     $d_i^{both} = |C| - 1 - \max\{k : \exists (l, r) \in V_k \text{ s.t. } r, l \leq L - l_i\}$
13     for $k \in \{|C| - 1, ..., 1\}$ do
14         $A = \emptyset$
15         if $k - d_i^{both} \geq 0$ and $d_i^{both} > 0$ then
16             $A \leftarrow A \cup (l + l_i, r + l_i)$ for $(l, r) \in V_{k - d_i^{both}}$
17         end
18         $V_k \leftarrow$ Nondominated($A, V_k$)
19     end
20 end
21 for $i \in \mathcal{I}$ do
22     $V \leftarrow$ FindMax($C, l, d, \mathcal{I} \backslash \{i\}, V^{cover}, L$)
        $d_i^{left} = |C| - 1 - \max\{k : \exists (l, r) \in V_k \text{ s.t. } l \leq L - l_i\}$
23     $d_i^{right} = |C| - 1 - \max\{k : \exists (l, r) \in V_k \text{ s.t. } r \leq L - l_i\}$
24 end
25 return $d$
```

```
Function: AlgorithmSides($C, j, l, \mathcal{I}, L$)
Input: Minimal cover $C$, special item $j$, item lengths $l = (l_1, ..., l_n)$, items $\mathcal{I}$, and
       knapsack length $L$
Output: The coefficient vector $d$ of the lifted cover inequality
1  Let $\mathcal{L} = \{0, 1, ..., L\}$.
2  Let $V_k = \emptyset \ \forall k \in \{1, ..., |C| - 1\}$.
3  Let $V_0 = \{(0, 0)\}$.                // Only $k = 0$ is possible with no items.
4  $\mathcal{I} \leftarrow \mathcal{I} \backslash C$
5  for $i \in C$ do
6      if $i = j$ then $d_i^{left} = d_i^{right} = d_i^{both} = 1$
7      else $d_i^{both} = 1$, $d_i^{left} = d_i^{right} = 0$
8  end
9  $V \leftarrow$ FindMax($C, l, d, C, V, L$)
10 $V^{cover} \leftarrow V$
11 for $i \in \mathcal{I}$ do
12     $d_i^{left} = |C| - 1 - \max\{k : \exists(l, r) \in V_k \text{ s.t. } l \leq L - l_i\}$
13     $d_i^{right} = |C| - 1 - \max\{k : \exists(l, r) \in V_k \text{ s.t. } r \leq L - l_i\}$
14     for $k \in \{|C| - 1, ..., 1\}$ do
15         $A = \emptyset$
16         if $k - d_i^{left} \geq 0$ and $d_i^{left} > 0$ then $A \leftarrow A \cup (l + l_i, r)$ for $(l, r) \in V_{k - d_i^{left}}$
17         if $k - d_i^{right} \geq 0$ and $d_i^{right} > 0$ then $A \leftarrow A \cup (l, r + l_i)$ for $(l, r) \in V_{k - d_i^{right}}$
18         $V_k \leftarrow$ Nondominated($A, V_k$)
19     end
20 end
21 for $i \in \mathcal{I}$ do
22     $V \leftarrow$ FindMax($C, l, d, \mathcal{I} \backslash \{i\}, V^{cover}, L$)
         $d_i^{both} = |C| - 1 - \max\{k : \exists(l, r) \in V_k \text{ s.t. } r, l \leq L - l_i\}$
23 end
24 return $d$
```

```
Function: FindMax($C, l, d, I, V, L$)
Input: Minimal cover $C$, item lengths $l = (l_1, ..., l_n)$, lifting coefficients $d$, relevant
       items $I$, non-dominated pairs using cover items $V$, and knapsack length $L$
Output: The set $W$ of non-dominated pairs $l, r$ for each value $k = 0, ..., |C| - 1$
        when using only items in $I \cup C$.
1  for $i \in I$ do
2      for $k \in \{|C| - 1, ..., 0\}$ do
3          $A \leftarrow \emptyset$
4          if $k - d_i^{left} \geq 0$ and $d_i^{left} > 0$ then $A \leftarrow A \cup (l + l_i, r)$ for $(l, r) \in V_{k-d_i^{left}}$
5          if $k - d_i^{right} \geq 0$ and $d_i^{right} > 0$ then $A \leftarrow A \cup (l, r + l_i)$ for $(l, r) \in V_{k-d_i^{right}}$
6          if $k - d_i^{both} \geq 0$ and $d_i^{both} > 0$ then
            $A \leftarrow A \cup (l + l_i, r + l_i)$ for $(l, r) \in V_{k-d_i^{both}}$
7          $V_k \leftarrow$ Nondominated($A, V_k$)
8      end
9  end
10 return $V$
```

3.3.5 Cover generation

In order to be able to lift a cover inequality or an adjacent cover inequality, it is necessary to first find a minimal cover. All minimal covers of a set of items can be found using Covers. It calls FindCover which works recursively. In each iteration FindCover either finds a cover and adds it to the set, adds the item but does not find a cover and calls itself, or cannot find a cover because there are no more items to add. Since the items are sorted in order of decreasing length, if adding an item creates a cover it must be a minimal cover, because the last item added is the smallest item in the cover. Algorithm FindCover is called a total of n times in Covers, once for every item. Each time, the item in question is forced to be in the cover, and only items with a smaller length can be added to it. Hence, all minimal covers are found while preventing the possibility of finding the same cover twice.

```
Function: Covers($\mathcal{I}, l, L$)
Input: Set of items sorted in decreasing length $\mathcal{I}$, items lengths $l$, and total
       length $L$
Output: The set of all minimal covers.
1 Let $Covers = \emptyset$ be the set of minimal covers.
2 for $cc \in \{1, ..., |\mathcal{I}|\}$ do
3     $i = \mathcal{I}(i)$
4     $C = \{i\}$
5     $Covers \leftarrow$ FindCover($cc, C, Covers$)
6 end
7 return $Covers$
```

```
Function: FindCover($cc, \mathcal{I}, C, Covers, l, L$)
Input: Current index $cc$, set of items sorted in decreasing length $\mathcal{I}$, current set $C$,
       set of found minimal covers $Covers$, items lengths $l$, and total length $L$
Output: The set of minimal covers where each minimal cover includes $C$ and
        some subset of items with smaller length than the $cc$'th item in $\mathcal{I}$.
1  if $cc = |\mathcal{I}|$ then
2      return $Covers$
3  end
4  for $aa \in \{cc+1, ..., |\mathcal{I}|\}$ do
5      $i = \mathcal{I}(aa)$
6      if $\sum_{x \in C} l_x + l_i > L$ then
7          $Covers \leftarrow Covers \cup \{C \cup \{i\}\}$
8      else
9          $Covers \leftarrow$ FindCover($aa, C \cup \{i\}, Covers$)
10     end
11 end
12 return $Covers$
```

3.4 Computational results

In this subsection we present computational results for the three formulations and the impact of the adjacent cover inequalities on our 2-stage approach. Before starting a more thorough investigation into our 2-stage approach, it is necessary to compare the 3 models for 2D-Knapsack to see why we, in addition to Fekete et al. [29] and Baldacci and Boschetti [35], opt for a 2-stage approach instead of solving the problem in a single MILP. The computational results are shown in Table 12.

Table 12: The average and maximum computation time taken over 100 instances of each instance type for the position assignment formulation (PAF), relative position formulation (RPF) and 2-stage approach. The solver was cut off after 1 hour. (*) The average and maximum for PAF were taken over only 10 instances.

model	**type I**		**type II**		**type III**	
	mean (s)	max (s)	mean (s)	max (s)	mean (s)	max (s)
PAF*	3600	3600	3600	3600	3600	3600
RPF	0.76	6.09	3.8	144.73	34.61	1115.43
2-stage	1.83	7.29	2.46	16.0	5.27	37.74

It can clearly be seen that the position assignment formulation is significantly worse than the other two models. The reason for this is twofold. Firstly, the LP relaxation for the position assignment formulation leads to a far higher dual bound than the other two formulations. Since 2D-Knapsack is a maximization problem, the dual bound gives an upper bound. A higher upper bound is thus worse. Secondly, the formulation contains more variables and constraints, making the solver slower. The relative position formulation is solved slightly faster than the two-stage model for type I instances on average, but this switches for type II and III instances. This suggests that it is more difficult to solve the relative position formulation when there are more small items. More importantly, the maximum time taken to prove optimality for the relative position formulation increases greatly for type II and III instances, needing almost 20 minutes in the worst case. In practice a more stable model is often preferable to a model that solves easy instances quickly but difficult instance slowly. Hence, our 2-stage model is better than both single stage models.

The computational results related to the adjacent cover inequalities can be divided into five parts: the time it takes to generate the minimal covers, the time it takes to lift an inequality, the improvement to the LP relaxation for the various lifting sequences, the differences in improvement between the ways the inequalities were included in the model, and a comparison with normal cover inequalities.

Cover generation

The time it takes to generate all minimal covers using COVERS is given in Table 13. It can clearly be seen that these computation times are negligible compared to the time taken to lift those same inequalities, because finding a single cover takes $8 \cdot 10^{-6}$ seconds on average, while lifting it takes a factor of 1000 more (Table 14).

TABLE 13: The average and maximum time (in milliseconds) taken to find all minimal covers over 100 instances with 20, 40 or 60 items and instance types I, II, or III.

	20			40			60		
instance type	I	II	III	I	II	III	I	II	III
average time (ms)	1	2	4	53	127	432	1260	4550	19 854
maximum time (ms)	3	11	22	280	873	2739	22 975	48 467	300 307

Lifting speed

We discussed two types of lifting sequences (per item, per variable type) with 3 item orders (random, increasing length, decreasing length) in Section 3.3.4. The 'per variable' lifting sequence was further subdivided into lifting variables where the item covers both slices first or lifting variables where the item only covers one slice first. Four algorithms to lift adjacent cover inequalities per item were proposed. A comparison in computation time per lifted inequality of these algorithms is given in Table 14. It can be seen that the third method (used in ALGORITHM3 and ALGORITHM4) is significantly faster than the other two methods. This was expected, since while this method has the same worst case time complexity as ALGORITHM2, it is unlikely to occur in practice. Contrary to expectations, ALGORITHM3 is faster than ALGORITHM4, with the difference becoming more notable as N increases. We suspect this is due to the fact that $|V_k|$ tends to be small in practice, making a very small inner loop. The two algorithms ALGORITHMBOTH and ALGORITHMSIDES are based on ALGORITHM3, but they have an additional dependency on N due to the necessity of FINDMAX.

TABLE 14: The average time taken to lift an adjacent cover inequality for each algorithm in milliseconds. The average was taken over 15 instances for each number of items in the instance N. The instances were evenly divided over type I, II, and III.

N	20	40	60
ALGORITHM1	4540	10500	16100
ALGORITHM2	115	321	555
ALGORITHM3	1	3	4
ALGORITHM4	2	9	17
ALGORITHMBOTH	2	11	21
ALGORITHMSIDES	2	10	20

Lifting sequences

The goal of trying various lifting sequences was to see if one sequence was strictly better than another, or if all sequences had instances where they obtained a better LP-bound than the other sequences. We measure the impact of the adjacent cover inequalities in terms of the relative improvement to the LP relaxation. Let z_{LP} be the original value obtained by the LP relaxation, let OPT be the optimal objective value, and let z_{adj} be the LP-bound obtained after adding the inequalities. The relative improvement Δ, which indicates how much of the gap between the original LP relaxation and the optimal solution is closed by adding the adjacent cover inequalities, is then given by

$$\Delta := 100\frac{z_{LP} - z_{adj}}{z_{LP} - OPT} \tag{33}$$

The results of the three item orders are shown in Figure 12 for each sequence type. It is clear that there is no real difference between the different item orders when lifting per item or lifting using AlgorithmBoth, while sorting the items by decreasing length is better on average when lifting using AlgorithmSides. The improvement to the LP bound for the best item order of the three sequence types is comparable. It is noteworthy that it is only better on average, because none of the nine sequence type and item order combinations dominates another.

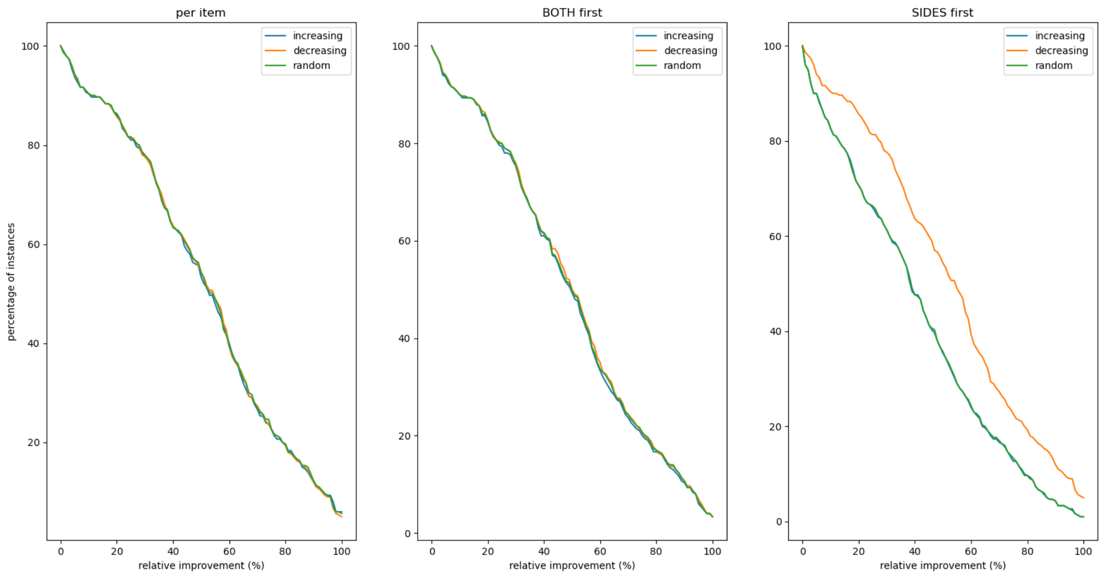


Figure 12: The percentage of instances where adding the lifted inequalities improves the LP-bound by more than a relative improvement r when the inequalities are lifted with Algorithm3 (left), AlgorithmBoth (center) and AlgorithmSides (right) with the items sorted randomly, by increasing length or by decreasing length. A total of 300 instances with 20 items, split evenly per instance type, were used.

Inclusion method

In practice, it would be unrealistic to directly add all lifted inequalities to the model as constraints due to the number of inequalities, which becomes especially notable as the number of items increases. It is also unnecessary to add the inequalities as constraints (model inequalities), since they only influence the dual bound and not the feasibility of a solution to the primal. Hence, we considered adding the inequalities as cuts (inequalities whose only purpose is to strengthen the LP relaxation) through a separator with the SCIP settings set as described in Section 2. This enables us to add a portion of the inequalities, depending on which were violated by the LP relaxation solution. This also allows us to choose how many violated inequalities to add each time the solver is called; this number can have an impact on the dual bound obtained at the root node, and on the time it takes to obtain this dual bound. We only compare the value of the dual bound and not the computation time, because the separator is written in Python which greatly slows down SCIP due to its poor integration. Hence, including more inequalities at a time is better simply because the separator gets called fewer times.

Table 15: The average relative improvement obtained when adding 50, 100, 200, 500, 1000 or all violated lifted adjacent cover inequalities as cuts in a separator. The average is taken over a total of 300 instances, 100 for each instance type, with 20 items per instance.

number of inequalities	50	100	200	500	1000	all
average relative improvement (%)	52.88	52.98	53.01	53.01	53.01	53.01

The average relative improvement obtained by varying the number of lifted adjacent cover inequalities added each time the separator is called is given in Table 15. The improvement is only a little lower if only 50 or 100 inequalities are added; the other four have the same average improvement. Once again, we note that none of the variations is strictly better than any other, since there exist instances such that one is better than the other and vice versa for all of them.

If we do not lift the adjacent cover inequalities beforehand, it also becomes impossible to look for violated lifted adjacent cover inequalities. Instead, we can only look for (nearly) violated adjacent cover inequalities, lift those, and then use the ones that became violated. Given a leniency of 1, an adjacent cover inequality (17) is violated if the left side is within 1 of the right side. The impact of the leniency on the improvement of the LP-relaxation is shown in Table 16, where it can be seen that the impact tapers off as the leniency increases. However, choosing to increase the leniency just because more violated lifted inequalities are found is not a proper method. For example, we restricted the leniency to 2, since the smallest cover size is 2. This already means that all adjacent cover inequalities with minimal covers of size 2 were lifted, even if the left side of the inequality was originally 0. This naturally defeats the purpose of selecting the inequalities based on how close they are to being violated, and even in this case the improvement to the LP-bound is lower than that when lifting the inequalities beforehand. Therefore, we suggest lifting the adjacent cover inequalities before solving the model. If this is too expensive (in computation time) we suggest using a procedure such as Wolter's [27] by finding a non-minimal violated adjacent cover inequality, reducing it to a minimal cover, and then lifting it.

Table 16: The average relative improvement obtained by adding all violated lifted adjacent cover inequalities found with a leniency of 0, 0.5, 1, 1.5, and 2. The average is taken over a total of 300 instances, 100 for each instance type, with 20 items per instance.

amount of leniency	0	0.5	1	1.5	2
average relative improvement (%)	37.06	42.41	45.98	49.02	50.79

Cover inequalities

While decreasing the gap between the original LP relaxation solution and the optimal solution by a little more than half (53%) sounds good, it is important to compare it to the impact of other inequalities. Since adjacent cover inequalities are closely related to cover inequalities, we compare the impact they have on the LP relaxation when added separately and together. It is shown in Figure 13 that the adjacent cover inequalities have a stronger impact on the LP relaxation than the cover inequalities, and the combination of the two improves it even more.

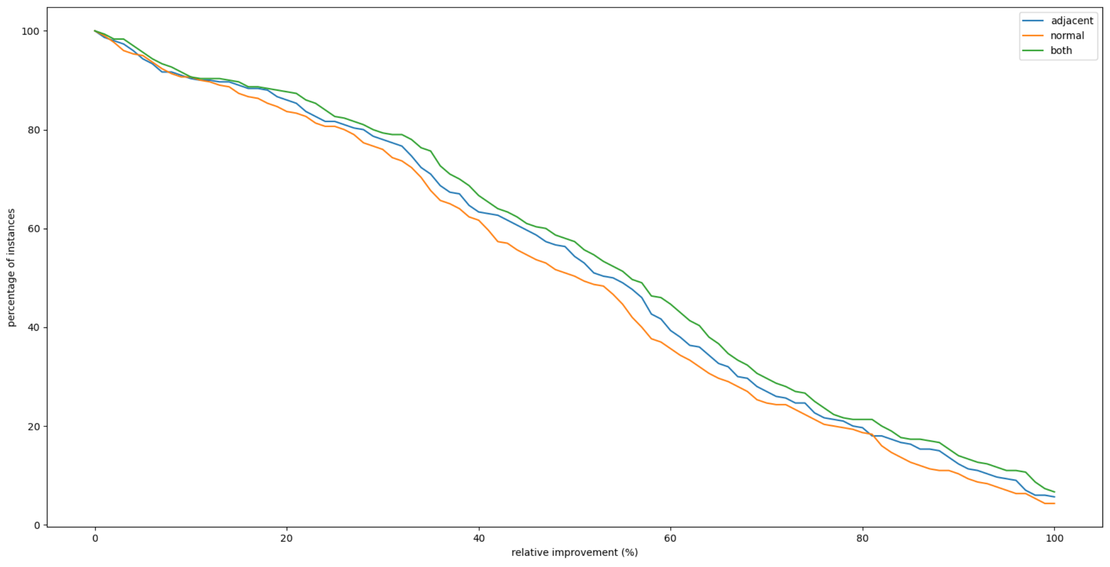


FIGURE 13: The percentage of instances where the added lifted inequalities improve the LP-bound by more than a relative improvement r, when adding only cover inequalities, only adjacent cover inequalities, or both. The percentage is taken over a total of 300 instances, 100 for each instance type, with 20 items per instance.

4 Quay crane assignment (stage 1)

Recall that we considered 2D-Knapsack as a relaxation to the BACAP, and that we solve the BACASP in a 3-stage approach where the first two stages correspond to the BACAP. The first stage assigns the number of quay cranes that will handle each vessel at each time step, and thus also determines the period of time the ship is berthed. The second stage checks if a given solution to the first stage has a feasible berthing allocation, and the third stage assigns specific quay cranes to each vessel at each time step. The BACAP is more complicated than 2D-Knapsack in four aspects: vessels have arrival times and thus cannot be scheduled anywhere, there is an additional restriction based on the number of available quay cranes, the handling times of vessels depends on the quay cranes, and the objective depends not on if a vessel is scheduled but on when it is scheduled. Hence, a 2D-Knapsack instance would correspond to a BACAP instance where each vessel arrives at time 0 and must be served by exactly one crane while berthed. The only remaining difference is then the objective. Similar to our approach to 2D-Knapsack, we consider a 2-stage approach where the first stage assigns the horizontal positions (berthing times) and the second stage checks if a feasible assignment of vertical positions (berthing locations) exists. If it does not, the solution to stage 1 is rejected. In this section, we discuss two versions of the quay crane assignment problem and several methods of improving the time taken by the solver to prove optimality.

There are three different versions of the quay crane assignment problem (QCAP): the time-variant, restricted time-variant and time-invariant versions. The time-variant version allows the number of quay cranes handling a vessel to differ during service, whilst the time-invariant version keeps the number of cranes handling a vessel constant during service [45]. The time-invariant version restricts the possibilities, which can lead to a worse optimal solution than the time-variant version, but restricting the solution space result in a considerably smaller computation time [45]. The restricted time-variant version allows the number of cranes handling a vessel to change according to a certain rule also exists (e.g. cranes can only move between shifts [6]). This results in more feasible solutions than the time-invariant version, but less than the time-variant version.

The quay crane assignment problem (QCAP) is rarely solved independently, and research often combines it with the berth allocation problem. The time-variant QCAP has been approached in several different ways. Meisel and Bierwirth [20] use a MILP that directly assigns a number of cranes to vessels at each time step. Giallombardo et al. [5] and Zhen [46] use quay crane profiles, each of which consists of a set of feasible crane assignments for a single vessel. Iris et al. [8, 10] apply a set partitioning model to both the time-variant and time-invariant versions. Due to the model complexity, many articles focus on developing metaheuristics such as genetic algorithms instead of solving the model exactly [8]. The first to apply a rolling horizon strategy were Xiao and Hu [13]. Liu et al. [9] apply it in conjunction with adaptive large neighborhood search.

The time-invariant QCAP is solved both exactly and approximately. Chen et al. [47] derive a benders' cut algorithm that was more efficient than the branch-and-cut algorithm embedded in CPLEX at the time. Later, Correcher [16] present a branch-and-cut algorithm to obtain an exact solution, while Cheimanoff et al. [17] present both an exact method and a heuristic using variable neighborhood search. Some articles provide an initial solution from a heuristic to the solver (warm start) to reduce the time it takes to solve the model [8]. An unconventional approach using scheduling techniques is implemented by Blazewicz et al. [11] to obtain a suboptimal solution, while Ji et al. [23] construct a rolling horizon strategy.

We consider both a time-variant and time-invariant version based on the work of Meisel and Bierwirth [20] and the work of Correcher [16] respectively. In order to improve the computation time of the solver, we consider both a warm start and a rolling horizon strategy. The time-variant version is discussed in Section 4.1, and the time-invariant version is discussed in Section 4.2.

4.1 Time-variant quay crane assignment

Six variations of a time-variant model are considered, and the version with the best worst-case and average-case behavior was selected to be used in the 3-stage approach. Formulation A_1 is given in (34) and can be considered the default formulation. Four types of changes to formulation A_1 were considered; these are discussed in Sections 4.1.1 to 4.1.4. Each formulation is categorized based on how it differs from A_1 in these four ways. The specific differences are shown in Table 18.

TABLE 17: The parameters (above) and decision variables (below) for formulation A_1 of the time-variant quay crane assignment problem.

L	The total berth length.
Q_{max}	The total number of QC.
Q	The set of possible quay crane assignments: $Q = \{1, ..., Q_{max}\}$.
T_{max}	The end of the time horizon.
T	The set of time steps: $T = \{1, ..., T_{max}\}$.
N	The number of vessels.
V	The set of vessels: $V = \{1, ..., N\}$.
u_i	The maximum number of QC that can handle vessel i simultaneously.
l_i	The minimum number of QC used to handle vessel i when berthed.
v_q	The speed in containers per time step Δt if q cranes are used simultaneously.
Con_i	The number of containers that need to be loaded/unloaded for vessel i.
a_i	The arrival time of vessel i.
s_i	The size of vessel i.
y_{iqt}	Equals 1 if and only if vessel i is handled by q cranes at time t.
b_{it}	Equals 1 if and only if vessel i is berthed at time t.
$start_{it}$	Equals 1 if and only if vessel i starts berthing at time t.
c_i	The completion time of vessel i (the time it leaves the berth).
QC_{iq}^{start}	Equals 1 if and only if q cranes are used when vessel i starts service.
QC_{iq}^{end}	Equals 1 if and only if q cranes are used when vessel i finishes service.

$$\min \quad \sum_{i \in V} c_i - a_i \tag{34a}$$

$$\text{s.t.:} \quad \sum_{q \in Q} y_{iqt} \leq 1 \qquad \forall i \in V, \forall t \in T \tag{34b}$$

$$\sum_{q \in Q} q y_{iqt} \leq u_i \qquad \forall i \in V, \forall t \in T \tag{34c}$$

$$\sum_{q \in Q} q y_{iqt} \geq l_i b_{it} \qquad \forall i \in V, \forall t \in T \tag{34d}$$

$$\sum_{i \in V} \sum_{q \in Q} q y_{iqt} \leq Q_{max} \qquad \forall t \in T \tag{34e}$$

$$\sum_{q \in Q} y_{iqt} \leq b_{it} \qquad \forall i \in V, \forall t \in T \tag{34f}$$

$$1.1 \sum_{i \in V} s_i b_{it} \leq L \qquad \forall t \in T \tag{34g}$$

$$\sum_{t \in T} \sum_{q \in Q} v_q y_{iqt} \geq Con_i + \frac{1}{4} \sum_{q \in Q} v_q (QC_{iq}^{start} + QC_{iq}^{end}) \qquad \forall i \in V \tag{34h}$$

$$\sum_{q \in Q} QC_{iq}^{start} = 1 \qquad \forall i \in V \tag{34i}$$

$$\sum_{q \in Q} QC_{iq}^{end} = 1 \qquad \forall i \in V \tag{34j}$$

$$QC_{iq}^{start} \geq y_{iqt} - b_{i(t-1)} \qquad \forall q \in Q, \forall i \in V, \forall t \geq 2 \in T \tag{34k}$$

$$Q_{iq}^{start} \geq y_{iq1} \qquad \forall q \in Q, \forall i \in V \tag{34l}$$

$$QC_{iq}^{end} \geq y_{iqt} - b_{i(t+1)} \qquad \forall q \in Q, \forall i \in V, \forall t \leq T_{max} - 1 \in T \tag{34m}$$

$$Q_{iq}^{end} \geq y_{iqT_{max}} \qquad \forall q \in Q, \forall i \in V \tag{34n}$$

$$(t+1) b_{it} \leq c_i \qquad \forall i \in V, \forall t \in T \tag{34o}$$

$$\sum_{t \in T} b_{it} + t \cdot start_{it} = c_i \qquad \forall i \in V \tag{34p}$$

$$\sum_{t \in T} t \cdot start_{it} \geq a_i \qquad \forall i \in V \tag{34q}$$

$$\sum_{t \in T} start_{it} = 1 \qquad \forall i \in V \tag{34r}$$

$$\sum_{n=1}^{t} start_{in} \geq b_{it} \qquad \forall i \in V, \forall t \in T \tag{34s}$$

$$start_{it} \leq b_{it} \qquad \forall i \in V, \forall t \in T \tag{34t}$$

$$y_{iqt} \in \{0, 1\} \qquad \forall i \in V, \forall q \in Q, \forall t \in T \tag{34u}$$

$$b_{it}, start_{it} \in \{0, 1\} \qquad \forall i \in V, \forall t \in T \tag{34v}$$

$$QC_{iq}^{start}, QC_{iq}^{end} \in \{0, 1\} \qquad \forall i \in V, \forall q \in Q \tag{34w}$$

$$c_i \in \{1, ..., T_{max} + 1\} \qquad \forall i \in V \tag{34x}$$

Formulation A_1 correctly models the quay crane assignment problem. The objective is to minimize the total dwell time. Constraint (34b) ensures that only a single number of cranes can be assigned to each vessel at each time step. Constraints (34c) and (34d) ensure that the assigned number of quay cranes is between its upper and lower bounds if it is berthed. Constraints (34e) and (34g) ensure that the total number of cranes in use and the berthing space occupied do not exceed the availability at each time step. Constraint (34f) ensures that if a vessel is being handled by a crane, then it must also be berthed. In the case that the lower bound of all vessels is at least 1, this inequality can be replaced by an equation.

Constraint (34h) ensures that all containers of a vessel are loaded/unloaded, given the assumption that berthing/unberthing takes 15 minutes each. Since the berthing (unberthing) takes up 15 minutes of the first and last time step a vessel is berthed, the cranes assigned to the vessel at these times can only perform 45 minutes of work instead of 60. Hence, the 15 minutes added in excess need to be subtracted. The number of cranes used during the first and last time step the vessel is berthed are assigned by constraints (34i)–(34n). Since only a single QC_{iq}^{start} and QC_{iq}^{end} can be set to 1 per vessel, and because there must exist a time step where a ship starts berthing and ends berthing due to constraints (34h) and (34r), these six constraints define QC_{iq}^{start} and QC_{iq}^{end} as in Table 17.

Constraints (34o) together with the objective ensures that the completion time is the time step after the ship unberths. Constraint (34p) ensures that a vessel remains berthed from when it starts berthing until it unberths. If (34f) is set to equality instead, this constraint is superfluous. Constraints (34q)-(34t) ensure that each ship is assigned a single start time after it arrives, and that its start time is when it starts berthing. Constraint (34r) forces each ship to be handled within the time horizon. Constraints (34u)-(34x) define the decision variables to be binary variables for $y, b, start, QC^{start}, QC^{end}$, and integer for c.

Table 18: The differences between the 6 time-variant formulations

variations	A_1	A_2	B_1	B_2	C_1	D_1
start time binary	✓	✓	✓	✓		
start time integer					✓	✓
completion time binary		✓		✓		
completion time integer	✓		✓		✓	✓
quay crane bounds			✓	✓	✓	
additional constraints		✓		✓		

The six versions differ in 4 aspects, namely the way the start and completion time are modelled, the way the lower and upper bounds on the quay cranes are modelled, and the inclusion of extra constraints to aid cut generation. Formulation A_1 in (34) models the start time with a binary variable ($start_{it}$), the completion with an integer variable (c_i), models the upper and lower bounds for quay cranes by adding them as constraints, and does not include extra constraints. All variations considered have some aspect that could potentially lower the computation time for the solver.

4.1.1 Integer start time

The first change is to represent the time the vessel starts berthing with an integer variable. This means that all constraints including $start_{it}$ will change, so constraints (34p)-(34t). This was done by replacing the binary variable $start_{it}$ by an integer variable $start_i$ such that

$$start_i := \sum_{t \in T} t \cdot start_{it} \qquad \forall i \in V. \tag{35}$$

This can be directly substituted in constraints (34p) and (34q). Constraint (34q) can also be removed by setting $start_i \in \{a_i, ..., T\}$ instead of $\{1, ..., T\}$. Constraint (34r) becomes unnecessary, and constraint (34s) needs to be rewritten. Its purpose was to ensure that the ship could not be berthed before it starts berthing, so the new constraint needs to do the same. This is done through a big-M constraint in (36c), which ensures that $start_i$ is less than or equal to the smallest t for which $b_{it} = 1$. It is set equal to the smallest t due to constraint (36d), which is 0 for all t except the smallest t for which $b_{it} = 1$. Hence, the replacement for constraints (34p)–(34t) given by (36) maintains a correct model for the quay crane assignment problem.

$$\sum_{t \in T} b_{it} = c_i - start_i \qquad \forall i \in V \tag{36a}$$
$$start_i \geq a_i \qquad \forall i \in V \tag{36b}$$
$$start_i \leq t + T_{max}(1 - b_{it}) \qquad \forall i \in V, \forall t \in T \tag{36c}$$
$$start_i \geq t(b_{it} - b_{i(t-1)}) \qquad \forall i \in V, \forall t \in T \backslash \{1\} \tag{36d}$$
$$start_i \geq b_{i1} \qquad \forall i \in V \tag{36e}$$
$$start_i \in T \qquad \forall i \in V \tag{36f}$$

4.1.2 Quay crane bounds

The second change is to model the upper and lower bounds for the number of quay cranes used by a vessel by restricting the variables instead of through constraints (34c) and (34d). There are three variables that depend on the number of cranes: QC_{iq}^{start}, QC_{iq}^{end}, and y_{iqt}. Constraints (34c) and (34d) force $y_{iqt} = 0$ for $q < l_i$ or $q > u_i$. Hence, these variables do not contribute and can be removed. The number of cranes used at the start and end of berthing are naturally also limited by l_i and u_i, so the same can be done for those. This results in

$$y_{iqt} \in \{0, 1\} \qquad \forall i \in V, \forall q \in Q_i, \forall t \in T \tag{37a}$$
$$QC_{iq}^{start}, QC_{iq}^{end} \in \{0, 1\} \qquad \forall i \in V, \forall q \in Q_i \tag{37b}$$

where $Q_i := \{\max(1, l_i), l_i + 1, ..., u_i\}$ is the set of cranes that are feasible for vessel i. It is possible that $l_i = 0$, but it is not possible to assign 0 cranes, so the smallest term in Q_i is replaced by 1 if this is the case. However, just replacing (34c), (34d), (34u), and (34w) by (37) does not make a valid model, because many constraints in (34) would then include non-existing y-variables. This is resolved by replacing Q by Q_i in all constraints in (34).

4.1.3 Binary completion time

The third change is to model the completion time using a binary variable end_{it} instead of the integer variable c_i. The two are related through

$$\sum_{t \in T} t \cdot end_{it} = c_i \qquad \forall i \in V. \tag{38}$$

In combination with (39a), this ensures that $end_{it} = 1$ if and only if $c_i = t$. Hence, replacing (34o), (34p), and (34x) by (39), as well as replacing c_i by $\sum_{t \in T} end_{it}$ in the objective, is a valid model.

$$\sum_{t \in T} end_{it} = 1 \qquad \forall i \in V \tag{39a}$$

$$\sum_{t \in T} b_{it} + t(start_{it} - end_{it}) = 0 \qquad \forall i \in V \tag{39b}$$

$$(t+1)b_{it} \leq \sum_{t \in T} t \cdot end_{it} \qquad \forall i \in V, \forall t \in T \tag{39c}$$

$$end_{i(T_{max}+1)} = b_{iT_{max}} \qquad \forall i \in V \tag{39d}$$

$$end_{it} \in \{0, 1\} \qquad \forall i \in V, \forall t \in T \cup \{T_{max} + 1\} \tag{39e}$$

4.1.4 Additional constraints

The fourth change does not remove or replace anything. Instead, it solely adds constraints to aid cut generation for the solver. The additional constraints can be placed into two groups: constraints related to the start time and constraints related to the completion time. These constraints are given in [10], and are stated here in (40a)-(40e) for the start time constraint and in (40f)-(40j) for the completion time constraints. Some constraints from [10] are not given here, because they are already in (34) or (39).

$$b_{i1} = start_{i1} \qquad \forall i \in V \tag{40a}$$

$$start_{it} = 0 \qquad \forall i \in V, \forall t \in \{1, ..., a_i - 1\} \tag{40b}$$

$$b_{i(t-1)} + start_{it} \leq 1 \qquad \forall i \in V, \forall t \in \{a_i + 1, ..., T_{max}\} \tag{40c}$$

$$\sum_{n=a_i}^{t-1} b_{in} + T_{max}(start_{it} - 1) \leq 0 \qquad \forall i \in V, \forall t \in \{a_i + 1, ..., T_{max}\} \tag{40d}$$

$$b_{i(t-1)} + start_{it} \geq b_{it} \qquad \forall i \in V, \forall t \in \{a_i + 1, ..., T_{max}\} \tag{40e}$$

$$b_{it} + end_{it} \leq 1 \qquad \forall i \in V, \forall t \in T \tag{40f}$$

$$end_{i(t+1)} \leq b_{it} \qquad \forall i \in V, \forall t \in T \tag{40g}$$

$$end_{it} = 0 \qquad \forall i \in V, \forall t \in \{1, ..., a_i + p_i^{min} - 1\} \tag{40h}$$

$$\sum_{n=t}^{T_{max}} b_{in} + T_{max}(end_{it} - 1) \leq 0 \qquad \forall i \in V, \forall t \in \{a_i + p_i^{min}, ..., T_{max}\} \tag{40i}$$

$$b_{i(t-1)} \leq b_{it} + end_{it} \qquad \forall i \in V, \forall t \in \{2, ..., T_{max}\} \tag{40j}$$

4.2 Time-invariant formulations

Five variations of a time-invariant model are considered, and the version with the best worst-case and average-case behavior is selected to be used in the time-invariant version of the 3-stage approach. Formulation INV_1 is given in (42). Two types of changes to INV_1 are considered; these are discussed in Section 4.2.1 and Section 4.2.2. The specific differences between the formulations are shown in Table 20.

TABLE 19: The parameters (above) and decision variables (below) for the time-invariant quay crane assignment problem.

L	The total berth length.
Q_{max}	The total number of QC.
T_{max}	The end of the time horizon.
T	The set of time steps: $T = \{1, ..., T_{max}\}$.
N	The number of vessels.
V	The set of vessels: $V = \{1, ..., N\}$.
u_i	The maximum number of QC that can handle vessel i simultaneously.
l_i	The minimum number of QC used to handle vessel i when berthed.
Q_i	The set of feasible crane allocations for vessel i.
v_q	The speed in containers per time step if q cranes are used simultaneously.
P_{iq}	The service time of vessel i is handled by q cranes.
T_{iq}	The set of feasible start times: $\{a_i, ..., T_{max} - P_{iq} + 1\}$ if vessel i is handled by q cranes.
Con_i	The number of containers that need to be loaded/unloaded for vessel i.
a_i	The arrival time of vessel i.
s_i	The size of vessel i.
y_{iqt}	Equals 1 if and only if vessel i starts berthing at time t and is handled by q cranes.
x_{iqt}	Equals 1 if and only if vessel i is handled by q cranes at time t.
p_i	The processing time of vessel i (the time it occupies part of the berth).
$start_i$	The start time of vessel i (the time it berths).
c_i	The completion time of vessel i (the time it unberths).

The time-invariant approach has several advantages over the time-variant approach. Since each vessel will use the same number of cranes during its service, the processing time is directly related to the number of cranes used. In the time-variant version, many different crane assignments can lead to the same handling time, but just changing the order in which the cranes are assigned can also cause an increase in the processing time. This requires (34h) as a constraint to ensure that all containers get loaded/unloaded. Such a constraint is unnecessary for the time-invariant approach, since selecting the number of quay cranes directly determines how long the vessel will remain berthed. This also removes the need for the QC_{iq}^{start} and QC_{iq}^{end} variables. The processing time P_{iq} of a vessel i given that it is handled by a number of cranes q is determined according to (41) prior to implementing the model.

$$P_{iq} = \left\lceil \frac{Con_i}{v_q} + 0.5 \right\rceil \tag{41}$$

We did an additional refinement based on P_{iq} to reduce assigning wasted cranes. If there are $q < q'$ such that $P_{iq} = P_{iq'}$, then using q' cranes 'wastes' $q' - q$ cranes each time step, since the vessel remains berthed for the same amount of time. These q' are thus excluded from the set of feasible crane allocations Q_i.

$$
\begin{aligned}
\min \quad & \sum_{i\in V} c_i - a_i && && \text{(42a)}\\
\text{s.t.:} \quad & \sum_{q\in Q_i}\sum_{t\in T_{iq}} y_{iqt} = 1 && \forall i \in V && \text{(42b)}\\
& \sum_{q\in Q_i} P_{iq} \sum_{t\in T_{iq}} y_{iqt} = p_i && \forall i \in V && \text{(42c)}\\
& \sum_{q\in Q_i}\sum_{t\in T_{iq}} t \cdot y_{iqt} = c_i - p_i && \forall i \in V && \text{(42d)}\\
& \sum_{n=t-P_{iq}+1}^{t} y_{iqt} = x_{iqt} && \forall i \in V, \forall q \in Q_i, \forall t \in T && \text{(42e)}\\
& \sum_{i\in V}\sum_{q\in Q_i} q x_{iqt} \leq Q_{max} && \forall t \in T && \text{(42f)}\\
& 1.1 \sum_{i\in V} s_i \sum_{q\in Q_i} x_{iqt} \leq L && \forall t \in T && \text{(42g)}\\
& y_{iqt} \in \{0,1\} && \forall i \in V, \forall q \in Q_i, \forall t \in T_{iq} && \text{(42h)}\\
& x_{iqt} \in \{0,1\} && \forall i \in V, \forall q \in Q_i, \forall t \in T && \text{(42i)}\\
& c_i, p_i \in \{0,1\} && \forall i \in V && \text{(42j)}
\end{aligned}
$$

Formulation INV$_1$ correctly models the time-invariant quay crane assignment problem. The objective is the total dwell time. Constraint (42b) ensures that each vessel is handled between its arrival and T_{max}. Constraint (42c) sets the processing time equal to P_{iq} if and only if $y_{iqt} = 1$ for some $t \in T_{iq}$. The completion time is defined as the start time plus the processing time in constraint (42d). Constraint (42e) defines the auxiliary decision variable x_{iqt} as a sum of y-variables. Due to the equality, the entire formulation can also be written solely with $p, c,$ and y- variables. We prefer to include x_{iqt}. It makes the MILP more legible, since it is an intuitive representation of the number of cranes occupies by each vessel at each time step. Constraint (42f) restricts the total number of cranes used at each time to Q_{max}, and constraint (42g) ensures that the space occupied by berthed vessels does not exceed the total berth length L at each time step. Constraints (42h)–(42j) define the decisions variables as binary variables.

TABLE 20: The differences between the 5 time-invariant formulations

variations	INV$_1$	INV$_2$	INV$_3$	INV$_4$	INV$_5$
equality	✓		✓		✓
big-M		✓	✓		
big-M dependent				✓	✓

The five versions differ in which constraint is used to link the x- and y- variables, namely using constraint an equality constraint as in (42e), using a big-M type constraint, or using a big-M type constraint with M dependent on i and q. The differences between the versions are shown in Table 20.

4.2.1 Big-M constraint

Formulations INV$_2$ and INV$_3$ replace constraint (42e) by (43). This replacement maintains validity of the model. If $y_{iqt} = 1$ then the ship remains berthed for the next P_{iq} time steps, so $x_{iqn} = 1$ for $n = t, ..., t + P_{iq} - 1$. This is enforced by (43a), because it requires the sum of these x-variables to be at least P_{iq}, which forces each of the variables to 1. If $y_{iqt} = 0$, then (43a) is always true so long as $M \geq \max_{i \in V} P_{i1}$. In this case there must be some other t' such that $y_{iqt'} = 1$ by (42b), and (43b) forces all x-variables not in (43a) for $t = t'$ to 0.

$$\sum_{n=t}^{t+P_{iq}-1} x_{iqn} - P_{iq} - M(y_{iqt} - 1) \geq 0 \qquad \forall i \in V, \forall t \in T_{iq}, \forall q \in Q_i \tag{43a}$$

$$\sum_{t \in T} \sum_{q \in Q_i} x_{iqt} = p_i \qquad \forall i \in V \tag{43b}$$

4.2.2 Big-M dependent on processing time

Formulations INV$_4$ and INV$_5$ are the same as INV$_2$ and INV$_3$ except constraint (43a) is replaced by (44). This can be seen as a tightening of the constraint, since it sets the right side to exactly 0 if $y_{iqt} = 0$, instead of to $P_{iq} - M$ which can be below 0. This has no influence on integer-valued solutions, but it does influence the LP relaxation. A fractional value for y_{iqt} now forces some x_{iqn} for $n = t, ..., t + P_{iq} - 1$ to a positive value, while this is not the case for (43a).

$$\sum_{n=t}^{t+P_{iq}-1} x_{iqn} - P_{iq} - P_{iq}(y_{iqt} - 1) \geq 0 \qquad \forall i \in V, \forall t \in T_{iq}, \forall q \in Q_i \tag{44}$$

4.3 Warm start

A method of speeding up a MILP solver is by providing it with a good initial primal solution. Often times, the solver uses built in heuristics to find its initial solution, but it is possible that a custom-built heuristic provides a better solution. We call providing this solution to the solver a warm start.

We tried four heuristics all based on a greedy approach. A greedy approach sorts the vessels in increasing or decreasing order of some aspect, and then always adds the first vessel in the list that still fits. For instance for 0-1 Knapsack, a greedy approach would be to sort the items by decreasing profit to weight ratio, and then to go through the list in order and every item that still fits in the knapsack. While this is a very intuitive and easy heuristic, greedy algorithms tend to have very bad worst-case results. For the algorithm described above, the solution provided by the greedy algorithm can be arbitrarily far from the optimal solution. This is illustrated in the example below.

Example 4.1. Consider a 0-1 Knapsack instance with 2 items and capacity $C \gg 0$, with item weights and profits given by Example 4.1 where $1 \gg \varepsilon > 0$.

items	1	2
weights	1	C
profits	$1+\varepsilon$	C
ratio	$1+\varepsilon$	1

In this case, the greedy algorithm will first place item 1 in the knapsack, and then conclude that item 2 does not fit. Hence, it obtains an objective value of $1+\varepsilon$. The optimal solution is to put item 2 in knapsack, which gives an objective value of C. Hence, the greedy algorithm is a factor of $\frac{C}{1+\varepsilon}$ away from the optimal value. Since C can be arbitrarily large, and ε is very close to zero, this ratio can get arbitrarily bad.

Instances that cause this worst-case behavior in the greedy algorithm are very unlikely to occur in practice, making the greedy approach quite useful. In our greedy approach, we find a solution to the time-invariant quay crane assignment problem. Since any solution to the time-invariant version is also a solution to the time-variant version, our greedy approach works as a warm start for both versions. The four sorting orders we used were increasing vessel size, decreasing vessel size, increasing number of containers, and decreasing number of containers. Once the set of vessels is sorted accordingly, algorithm WARMSTART finds a solution by scheduling the first vessel in the list that has arrived, fits at the berth and satisfies that the number of available quay cranes meets its minimum requirement. This continues until all vessels are scheduled.

```
Function: WARMSTART($\mathcal{I}, L, T_{max}, Q_{max}, Q, s, a, P$)
Input: Sorted set of vessels $\mathcal{I}$, berth length $L$, time horizon $T_{max}$, total quay
       cranes $Q_{max}$, feasible crane allocations $Q = \{Q_i$ s.t. $i \in \mathcal{I}\}$, vessel sizes $s$,
       arrival times $a$, and handling times $P = \{P_{iq}$ s.t. $i \in \mathcal{I}, q \in Q_i\}$
Output: A solution to the quay crane assignment problem, where $start_i, c_i$, and
        $QC_i$ are the start time, completion time, and used number of quay
        cranes for $i \in \mathcal{I}$ respectively, and $V$ is the objective value of the solution.
Let $L_t = L$ for $t \in \{1, ..., T\}$.
Let $Q_t = Q_{max}$ for $t \in \{1, ..., T\}$.
Let $V = 0$.
while $\mathcal{I} \neq \emptyset$ do
    for $t \in \{1, ..., T_{max}\}$ do
        for $i \in \mathcal{I}$ do
            if $a_i \leq t$ and $s_i \leq L_t$ and $\min\{q : q \in Q_i\} \leq Q_t$ then
                $q = \max\{q \in Q_i : q \leq Q_t\}$
                for $\tau \in \{t, ..., t + P_{iq} - 1\}$ do
                    $Q_t \leftarrow Q_t - q$
                    $L_t \leftarrow L_t - s_i$
                end
                $start_i = t$, $c_i = t + P_{iq}$, $QC_i = q$
                $V \leftarrow V + c_i - a_i$, $\mathcal{I} \leftarrow \mathcal{I} \setminus \{i\}$
            end
        end
    end
end
return $start, c, QC, V$
```

It can be seen in Table 21 that each of the four heuristics has instances where it obtains a better objective value than the other heuristics. Ordering the vessels by increasing number of containers is the best in the majority of the instances, with the ordering by increasing size in second place. The other two are only better in rare instances. Since computing these heuristic solutions takes an insignificant amount of time compared to the computation time of the optimal solution to the MILP, we used a combined heuristic that takes the best solution among these four heuristics and adds it to the solver as a warm start solution.

Table 21: The number of instances for which each heuristic obtained the best objective value. The heuristics are compared over 100 instances each for 10, 15, 20, 25, and 30 items, so a total of 500 instances.

sorting order	increasing Con_i	decreasing Con_i	increasing s_i	decreasing s_i
best	370	14	92	24

4.4 Rolling horizon strategy

Although proving optimality for a solution is interesting mathematically, it is not important for a company focused on functionality. A terminal just wants to get a good quay crane assignment, and thus the priority is to get a good solution as quick as possible, rather than the optimal solution in considerably more time. This section considers speeding up the process of finding a good solution by using a rolling horizon approach. Instead of trying to schedule all vessels for the full time horizon of 48 hours immediately, this approach divides the time horizon into multiple segments that partially overlap. For instance, dividing the time horizon into three segments, each with a length of $T_{max}/2$ and an overlap of $T_{max}/4$, not only halves the number of y_{iqt} and b_{it}-variables, but it also greatly decreases the number of constraints. A smaller instance is generally solved more quickly, so this can greatly improve the computation time.

The formulations in Section 4.1 need to be adapted for the rolling horizon strategy to work. Firstly, the requirement that all vessels are scheduled needs to be changed, because not all vessels have to be scheduled in a single segment. This is done by adding a binary variable z_i that equals 1 if and only if vessel i is scheduled during this segment. Constraints (34r), (34j), (34i), and (39a) (if included) need to be changed to equal 1 if and only if vessel i is actually scheduled. Hence, the right-hand side of these constraints is changed from 1 to z_i. Secondly, T changes to $\{1 + (T - overlap) \cdot segment, ..., (T - overlap) \cdot segment + T\}$ to represent the time frame of each segment. Thirdly, the set of vessels V becomes the set of unscheduled vessels. In the first segment this is still $\{1, ..., N\}$, but in other segments some vessels have already been scheduled. Since each vessel must be scheduled exactly once, these vessels need to be excluded from V after they are scheduled. Fourthly, the total berth length L and quay crane availability Q_{max} change to depend on the time step. Consecutive time segments overlap to ensure that a vessel that could not be scheduled in the previous segment because it could not complete service in this segment, can be scheduled with the same start time in the current segment. However, this overlap also causes part of the berth and quay cranes to already be occupied by vessels scheduled in the previous segment. Lastly, constraints that either sum over or only hold for specific time steps need to be changed. These changed constraints are given in (45).

It is possible that some vessel scheduled in the previous segment starts berthing after the overlap with the current segment. In this case, setting this vessel as unscheduled instead allows for a larger solution space. This can improve the optimal objective value. However, it also increases the number of vessels that need to be scheduled in later time segments.

$$\sum_{n=\min\{t:t\in T\}}^{t} start_{it} \geq b_{it} \qquad \forall i \in V, \forall t \in T \tag{34s}$$

$$c_i \in T \cup \{\max(T)+1\} \qquad \forall i \in V \tag{34x}$$

$$start_i \geq t(b_{it} - b_{i(t-1)}) \qquad \forall i \in V, \forall t \in T\setminus \min(T) \tag{36d}$$

$$start_i \geq b_{i(\min(T))} \qquad \forall i \in V \tag{36e}$$

$$end_{i(\max(T)+1)} = b_{i(\max(T))} \qquad \forall i \in V \tag{39d}$$

$$end_{it} \in \{0,1\} \qquad \forall i \in V, \forall t \in T \cup \{\max(T)+1\} \tag{39e}$$

$$b_{i(\min(T))} = start_{i(\min(T))} \qquad \forall i \in V \tag{40a}$$

$$start_{it} = 0 \qquad \forall i \in V \forall t \leq a_i \in T \tag{40b}$$

$$b_{i(t-1)} + start_{it} \leq 1 \qquad \forall i \in V \forall t \geq a_i + 1 \in T \tag{40c}$$

$$\sum_{n=\max(a_i,\min(T))}^{t-1} b_{in} \leq T_{max}(1 - start_{it}) \qquad \forall i \in V, \forall t \geq a_i + 1 \in T \tag{40d}$$

$$b_{i(t-1)} + start_{it} \geq b_{it} \qquad \forall i \in V, \forall t \geq a_i + 1 \in T \tag{40e}$$

$$end_{it} = 0 \qquad \forall i \in V, \forall t \leq a_i + p_i^{min} - 1 \in T \tag{40h}$$

$$\sum_{n=t}^{\max(T)} +T_{max}(end_{it} - 1) \leq 0 \qquad \forall i \in V, \forall t \geq a_i + p_i^{min} \in T \tag{40i}$$

$$b_{i(t-1)} \leq b_{it} + end_{it} \qquad \forall i \in V, \forall t \in T\setminus \min(T) \tag{40j}$$

4.5 Cutting planes

A third method of speeding up a solver is by adding cutting planes to improve the dual bound. In this section we discuss the method with which we add the adjacent cover inequalities from Section 3.3.2 to the time-invariant quay crane assignment problem. The minimal covers are found using algorithms Covers and FindCover, and the adjacent cover inequalities are lifted according to algorithm Algorithm3. We only consider lifting the vessels 'per item' and in random order.

The original adjacent cover inequalities for 2D-Knapsack (17) are designed for binary variables $y_{iv} := 1$ if and only if the leftmost position of item i was slice w. The most similar variable of the time-invariant QCAP is y_{iqt}, which equals 1 if and only if vessel i start service at time t with q cranes. Since it starts service at time t, this is equivalent to saying that t is the leftmost slice of the time axis occupied by vessel i. The addition of the cranes results in several y_{iqt}-variables that can lead to a vessel only occupying the left slice, right slice, or both slices. Let w and $w+1$ be the left and right slices. Let P_w^{left}, P_w^{right}, and P_w^{both} be sets of (i,q,t) triples such that (i,q,t) is in set $P_w^{left}/P_w^{right}/P_w^{both}$ if and only if $y_{iqt} = 1$ means vessel i occupies only the left/right/both slices. Then such triples can be found according to (47), (49), and (51) respectively.

$$P_w^{left} := \Big\{(i,q,t) : y_{iqt} = 1 \implies \sum_{q \in Q_i} x_{iqw} = 1, \sum_{q \in Q_i} x_{iq(w+1)} = 0\Big\} \tag{46}$$

$$= \Big\{(i,q,t) : t = w - P_{iq} + 1\Big\} \tag{47}$$

$$P_w^{right} := \Big\{(i,q,t) : y_{iqt} = 1 \implies \sum_{q \in Q_i} x_{iqw} = 0, \sum_{q \in Q_i} x_{iq(w+1)} = 1\Big\} \tag{48}$$

$$= \Big\{(i,q,t) : t = w + 1\Big\} \tag{49}$$

$$P_w^{right} := \Big\{(i,q,t) : y_{iqt} = 1 \implies \sum_{q \in Q_i} x_{iqw} = 1, \sum_{q \in Q_i} x_{iq(w+1)} = 1\Big\} \tag{50}$$

$$= \Big\{(i,q,t) : P_{iq} \geq 2, w \geq t \geq w - P_{iq} + 1\Big\} \tag{51}$$

The adjacent cover inequality for some cover C with side item j is then given by

$$\sum_{i \in C, (i,q,t) \in P_w^{both}} y_{iqt} + \sum_{q \in Q_i} y_{jq(w-P_{iq}+1} + y_{jq(w+1)} \leq |C| - 1. \tag{52}$$

The variables can be grouped in the same manner as in Section 3.3.3 to enable to use of algorithm ALGORITHM3 to lift the inequalities. One important aspect to note for implementation is that, contrary to knapsack items, vessels have arrival times. Our time-invariant model does not include y_{iqt}-variables for times $t < a_i$. Similarly, y-variables that would cause the vessel to still be in service at the end of the time horizon also do not exist. Hence, such y-variables must be removed from the lifted inequality before adding it to the model as a cutting plane.

4.6 Computational results

In this section we discuss all results related to the quay crane assignment problem. We first compare the various versions of the time-variant with respect to the time taken to prove optimality and the LP-bounds. We then show the effects of the SCIP parameters, warm start and rolling horizon on the time-variant version. This is followed by a comparison of formulations for the time-invariant version, and the effect of the adjacent cover inequalities on the LP relaxation. Finally, we compare the best time-variant version with the best time-invariant version in terms of the computation time and the obtained objective values.

Six variations of the time-variant quay crane assignment problem were proposed in Section 4.1. The percentage of instances completed within a given time are shown in Figure 14 for instance with 10, 15, or 20 vessels. We do not consider instances with 25 or 30 vessels here, because it can clearly be seen that versions A_2 and B_2 are significantly better than the other models, where 20 to 50 instances were not solved to optimality within the time limit of 10 minutes.

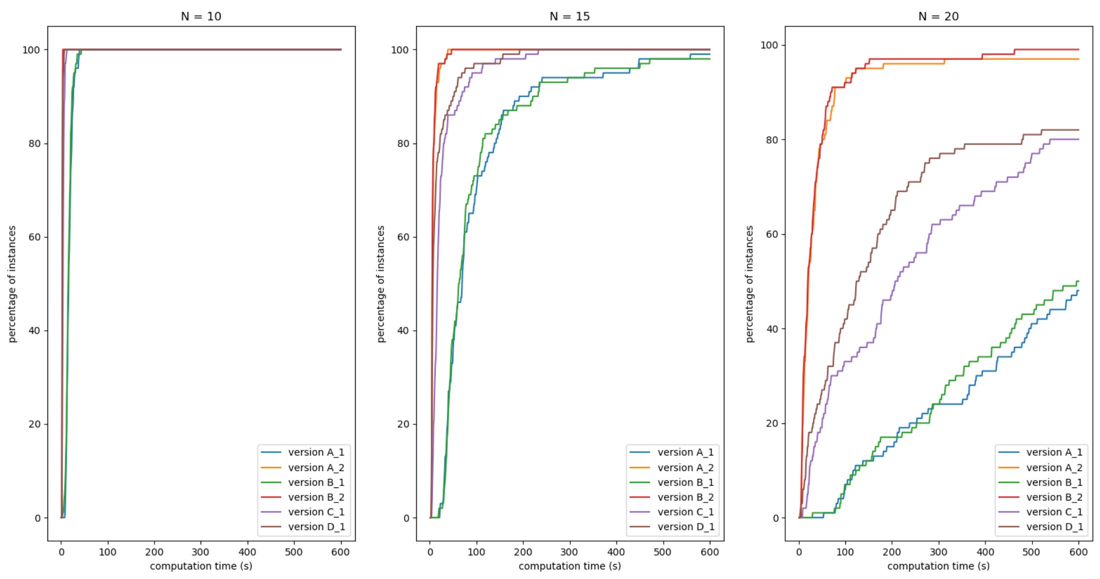


Figure 14: The percentage of instances for which optimality was proven within a given time t for instances with 10, 15, and 20 vessels for all 6 variations of the time-variant version of the quay crane assignment problem. A total of 100 instances were considered for each number of vessels.

As an additional factor of comparison we consider the LP relaxation value. Since the quay crane assignment problem is a minimization problem, a higher LP relaxation value results in a small gap between the primal and dual. Let formulation A_1 given in (34) be taken as the base formulation. We compare the LP relaxations of the other five formulations as a function of A_1 by taking the difference and dividing by the value obtained from A_1. The average, minimum and maximum of this relative difference are shown in Table 22 for instances with 10, 15 or 20 vessels. Since stage 1 solves a minimization problem, the LP relaxation gives a lower bound. A higher value for the LP relaxation is thus better. Formulations A_2, B_1, B_2, and C_1 all show improvement in the LP relaxation compared to formulation A_1, whilst formulation D_1 is always worse. There is improvement for formulations B_1, B_2 and C_1, so we conclude that modelling the upper and lower limits for quay crane usage as bounds instead of constraints has a positive influence on the LP relaxation. This is as expected, since it restricts the number of variables thus decreasing the possibilities for the LP relaxation to 'cheat' compared to the MILP. The most significant improvement is seen in formulation B_2, although it only has a small difference with formulation B_1. This is the effect of the additional constraints that is also seen in A_2. Although the improvement in the LP relaxation value is small compared to the effect of the quay crane bounds, the effect on the computation time is considerably greater. Formulation B_2 obtains both the best computation time and LP-bounds on average and in the worst case, so we will henceforth only consider this formulation in the rest of this report when referring to the time-variant version to the QCAP.

TABLE 22: The average, minimum and maximum relative increase in the LP relaxation value of the various time-variant formulations compared to formulation A_1. A total of 100 instances were considered for each N.

	A_2			B_1			B_2			C_1			D_1		
N	mean	min	max	mean	min	max	mean	min	max	mean	min	max	mean	min	max
10	0.08	0.04	0.12	0.29	0.21	0.49	0.31	0.23	0.5	0.28	0.16	0.49	-0.04	-0.9	0
15	0.08	0.05	0.13	0.29	0.22	0.38	0.31	0.25	0.41	0.28	0.18	0.38	-0.04	-0.87	0
20	0.08	0.05	0.12	0.29	0.22	0.37	0.31	0.25	0.39	0.28	0.19	0.36	-0.04	-0.89	0

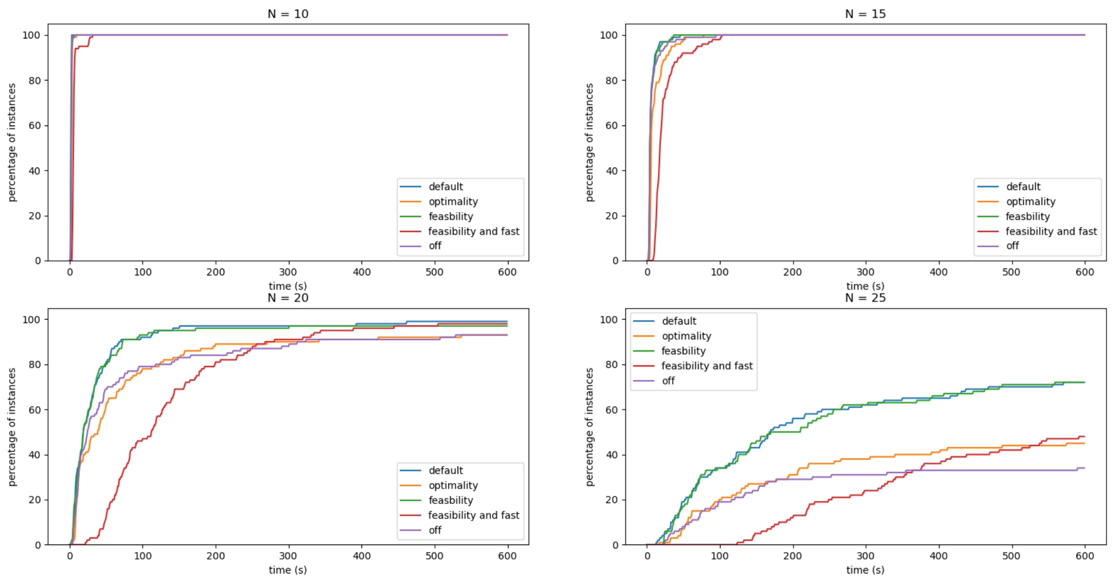


FIGURE 15: The percentage of instances for which optimality was proven within a given time t for instances with 10, 15, and 20 vessels for five parameter settings. A total of 100 instances were considered for each number of vessels.

One possible way to improve the speed with which the solver proves optimality is by changing its parameter settings, some of which are briefly described in Section 2. We first tested all mentioned parameter settings on 10 instances with 20 vessels to get an indication of the effect, after which we tested the more promising settings on 100 instances for 10, 15, 20 and 25 vessels. The results are shown in Figure 15. It can be seen that only the feasibility setting of the emphasis parameter results in similar behavior as the default settings, while the other three settings show a clearly worse performance. In order to make a distinction between the default and feasibility setting, we look at the gap in the instances that did not terminate within 10 minutes. It can be seen in Table 23 that setting the emphasis parameter to feasibility instead of the default results in a slightly worse gap. Hence, we decided to stick with the default parameter settings for all parameters. It is possible that a combination of different parameters could have a positive effect on the computation time. This is a possible direction of future research that will require more extensive knowledge of SCIP as well as more experience in how certain parameters affect the solver.

Table 23: The average and maximum gap (in %) for the default and feasibility settings for instances where optimality was not proven within 10 minutes with 10, 15, 20, and 25 vessels.

	default		feasibility	
N	mean	max	mean	max
10	-	-	-	-
15	-	-	-	-
20	3	3	4	7
25	7	25	8	25

The second method to improve the computation time that we tried was to apply a warm start. Our warm start solution was taken to be the solution with the best objective value from the four greedy algorithms given in Section 4.3. Warm starting our time-variant model has no effect on the number of instances for which the solver has proven optimality within the time limit, but it does improve the computation time on average (see Figure 16). We thus consider the warm start to be a slight improvement.

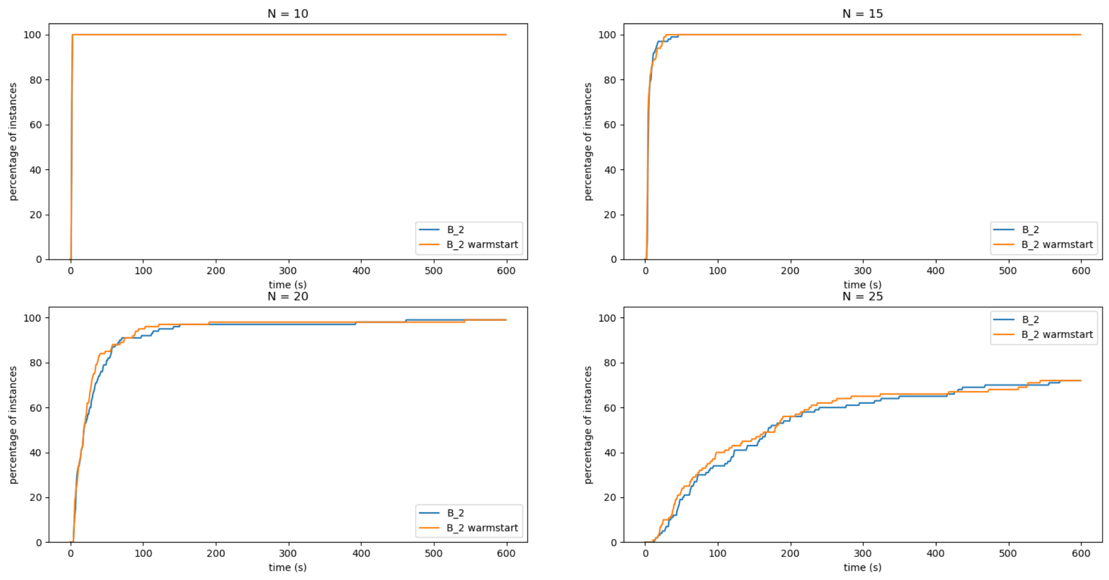


Figure 16: The percentage of instances for which optimality was proven within a given time t for instances with 10, 15, 20, and 25 vessels with and without the warm start. A total of 100 instances were considered for each number of vessels.

The third method to improve the computation time is by using a rolling horizon strategy according to Section 4.4. Contrary to changing the parameters or applying a warm start, using a rolling horizon strategy can result in suboptimal solutions even if an optimal solution is found during every time segment. It is thus considered to be an approximation method. We investigate the impact of two factors on the average and maximum deviation from the optimal solution: rescheduling and the amount of overlap between consecutive time segments.

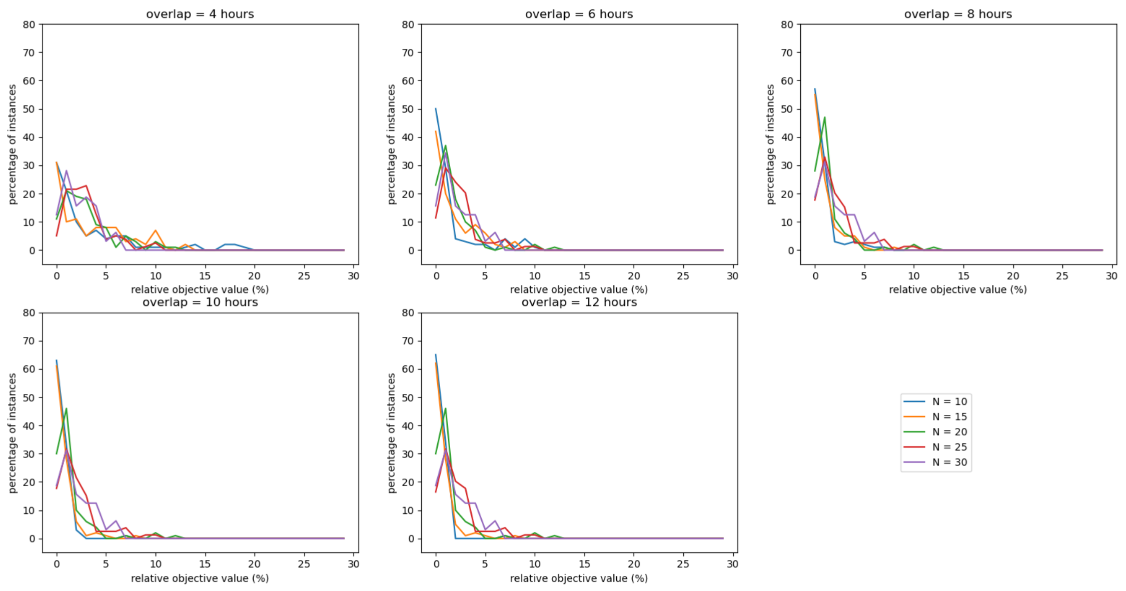


FIGURE 17: The percentage of instances for which the solution obtained using the rolling horizon strategy without rescheduling deviates from the optimal solution by a given percentage.

In order to determine the impact on the deviation from the optimal solution, we must first know the optimal solution. Hence, we can only make this comparison for instances where we found the optimal solution. To this end, we ran the time-variant model normally with a time limit of 30 minutes instead of 10. This resulted in proving optimality in all 100 instances for 10, 15 and 20 vessels, and finding it for 76 and 32 instances for 25 and 30 vessels respectively. The relative deviation Δ from the optimal solution is then determined by

$$\Delta = 100 \cdot \frac{\text{rolling} - OPT}{OPT}. \tag{53}$$

The results are shown in Figure 17 for the rolling horizon with an overlap of 4, 6, 8, 10 or 12 hours, and Figure 18 shows the same for the rolling horizon with rescheduling. It can be seen in the former that an overlap of only 4 or 6 hours has a worse performance than an overlap of 8, 10 or 12. An overlap of 4 results in more instances with a deviation of more than 5%, while an overlap of 6 has more instances with a deviation between 2–5%. The only difference between an overlap of 8, 10 and 12 is that the latter two show a slight increase in the percentage of instances with 10 and 15 vessels where the rolling horizon strategy also finds the optimal solution. However, we do not find this very useful, since those instances can already be solved to optimality by the time-variant model within 10 minutes.

Any terminal works with shifts for their crane operators. Common shift lengths are 8 or 12 hours, so we suggest taking the shift length as the overlap, and setting the first time step of the model at the start of a shift. In the case of shifts of 8 or 12 hours, this nicely results in planning a fixed number of shifts in each segment, whether the rolling horizon applies rescheduling or not.

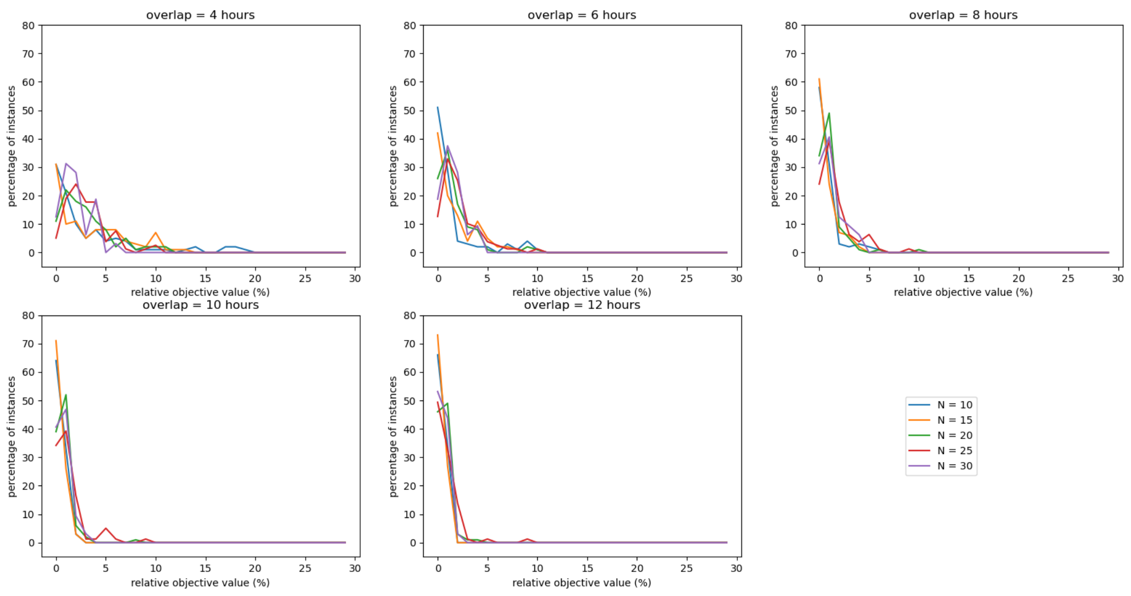


FIGURE 18: The percentage of instances for which the solution obtained using the rolling horizon strategy with rescheduling deviates from the optimal solution by a given percentage.

For the rolling horizon with rescheduling we see a similar pattern for an overlap of 4 or 6 hours. Contrary to the version without rescheduling, the percentage of instances with a deviation of more than 5% continues to decrease as the overlap increases. If the overlap is 12 hours almost all instances deviate less than 3%. It can also be seen that allowing rescheduling lessens the deviations of more than 5%, which is especially visible for an overlap of 10 or 12 hours. However, rescheduling schedules more vessels than without from the second segment onward, thus increasing the time taken by the solver to prove optimality in these segments. This becomes especially notable for an overlap of 10 or 12, where the average computation time increases by a factor of 2 by applying rescheduling. Which is better is therefore a deliberation between which is more important: a better objective value or a faster computation time.

In our case, the purpose of the model is to be able to redo the scheduling in case of unexpected events such as a crane breaking down, unexpected vessels arriving at the terminal or several vessels with large delays. In these situations it is more important to quickly get a new schedule than to get a slightly better schedule in double the time. We therefore continue without rescheduling with an overlap of 8.

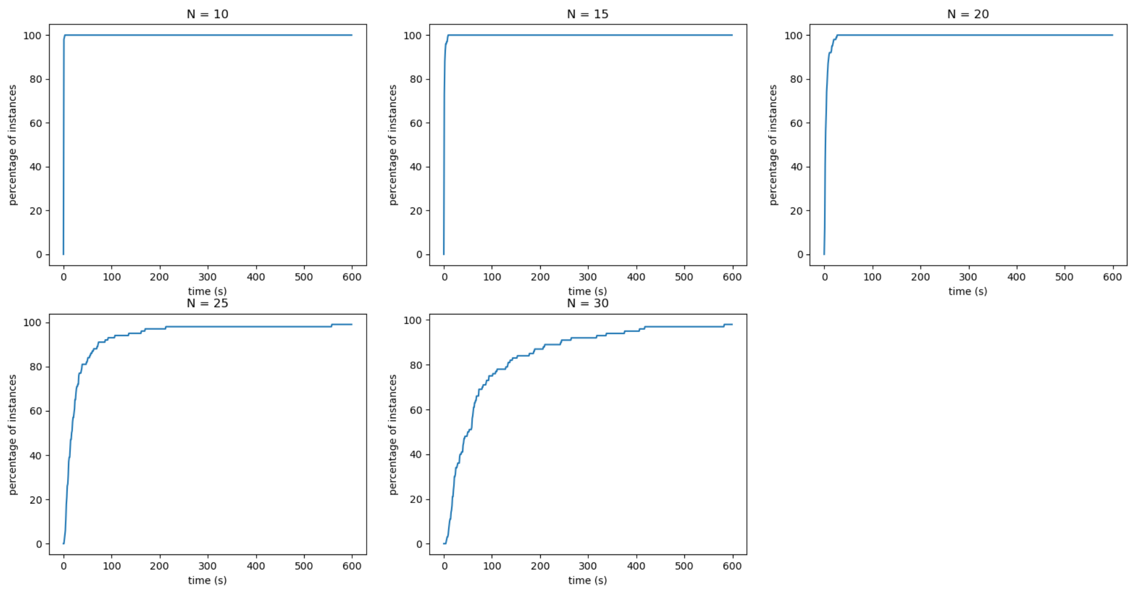


FIGURE 19: The percentage of instances with a computation time within a given time t for instances with 10, 15, 20, 25, and 30 vessels for the rolling horizon strategy without rescheduling with a warm start. A total of 100 instances were considered for each number of vessels.

If we warm start this version of the rolling horizon, we get the computation times shown in Figure 19. The warm start is more significant for the rolling horizon, because the heuristic solution from SCIP is to simply not plan any vessels. The rolling horizon strategy greatly improves the time taken to find an optimal solution and prove its optimality compared to without it. All instances except for three are solved within 10 minutes. The remaining three were solved in 601, 603, and 606 seconds respectively; they reached the cutoff limit in the one of the time segments. The remaining segments were solved within a few seconds. The specific instances, which segment reached the time limit, and the gap within this segment are shown in Table 24. In all three instances, the gap was continuously decreasing the entire 10 minutes through improvements to the dual bound. When the instances were run on a computer with slightly better hardware, the instances were completed within 10 minutes.

TABLE 24: The segment and corresponding gap for the three instances were terminated by the time limit. The instances are identified by the number of vessels N and the instance number.

N	instance number	segment	gap (%)
25	91	1	6.73
30	5	2	2.85
	30	1	3.10

The time-invariant quay crane assignment problem has a much smaller set of feasible solutions than the time-variant version. It is solved to optimality in a significantly shorter amount of time, which is shown by Figure 20. It shows the computation times of all instances for the five variations of our time-invariant model. It is difficult to see, but the lines for INV_2 and INV_5 are almost perfectly on top of each other; the same holds for the other three, with INV_1 a slight bit above the other two. We expected a version with (42e) to be the fastest due to the equality. Contrary to expectations, all LP relaxation values are the same for all five versions, and the number of cuts used by the solver had no notable differences.

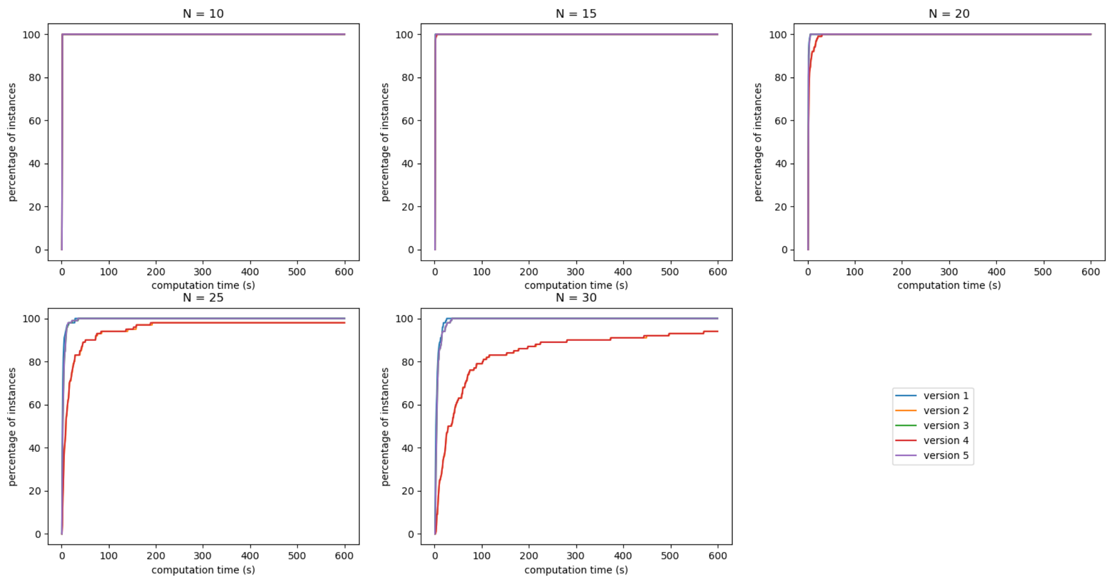


Figure 20: The percentage of instances for which optimality was proven within a given time t for instances with 10, 15, 20, 25, and 30 vessels for all 5 variations of the time-invariant version of the quay crane assignment problem. A total of 100 instances were considered for each number of vessels.

We also investigated the effect of adjacent cover inequalities on the LP relaxation of the time-invariant quay crane assignment problem. The inequalities are added as cutting planes using a separator; each time the separator was called by the solver all violated lifted adjacent cover inequalities are added. The resulting improvements for instances with 15 vessels are shown in Table 25. The time-invariant formulation INV_1 has many instances where the LP relaxation is equal to the optimal solution. In these cases, it is naturally impossible to obtain any improvement by adding cutting planes. Removing these instances from consideration leaves 45 out of 100 instances to compare. Out of these 45, only 8 instances show a higher (and thus better) LP bound by adding the adjacent cover inequalities. The relative improvement to the LP-bound is determined according to (33). It can be seen that if the adjacent cover inequalities have a nonzero influence on the LP relaxation, then the improvement is quite large, with all except one instance showing an improvement of 50% or more. However, the instances where the inequalities are useful are quite limited, leading to an average improvement of less than 11%. In comparison, using the same method for adding the inequalities for 2D-Knapsack resulted in an average improvement of 53% (see Table 15).

We also compare the relative improvement using adjacent cover inequalities to that of SCIP's default separator setting, which uses multiple types of cutting planes. The rest of the parameters were kept as described in Section 2. Using the default setting results in an average improvement of 100% by including between 1 and 45 cuts. In other words, the solver found an optimal solution without branching in all 45 instances. This is a far better result with far fewer cutting planes. Hence, we recommend to just use the solver separators. Note that this does not mean the adjacent cover inequalities are inherently worse than the types of cutting planes added by the solver. The solver uses multiple types of inequalities with a (almost certainly) better implementation than in this work.

TABLE 25: The 8 instances with 15 vessels for which applying the adjacent cover inequalities showed a positive effect on the dual bound. For each instance, the relative improvement and the number of adjacent cover inequalities used to obtain that improvement are given.

instance	relative improvement (%)	cuts
5	55	14725
8	60	1512
17	28	18816
56	79	293
58	100	5758
76	50	2
84	50	373
96	53	15948

Although the time-invariant version is solved much quicker even after the rolling horizon strategy is implemented for the time-variant version, we also need to consider the obtained objective values. Since the time-invariant version has fewer feasible solutions, it can be used as an approximation method for the time-variant version, similar to the rolling horizon strategy. We compare the objective value obtained by the time-invariant version to the objective value of the time-variant version for the instances with a known optimal solution. We also compare the time-invariant version and the rolling horizon to see which is the better approximation method for the time-variant QCAP. The relative objective values are obtained via

$$100\frac{\mathrm{INV}_1 - OPT}{OPT} \qquad \text{and} \qquad 100 \cdot \frac{\mathrm{INV}_1 - \text{rolling}}{\text{rolling}} \tag{54}$$

and are shown on the left and right plots of Figure 21 respectively. The time-invariant version deviated less than 10% from the time-variant version in the instances for which the optimal solution to the time-variant version is known. If this can be extrapolated to the instances where this is unknown, the time-invariant version is a good approximation of the time-variant version. However, it is still a worse approximation than the rolling horizon. There are a few instances where the time-invariant version obtains a better objective value than the rolling horizon, but the rolling horizon strategy is better by 1-7% in the majority of the instances.

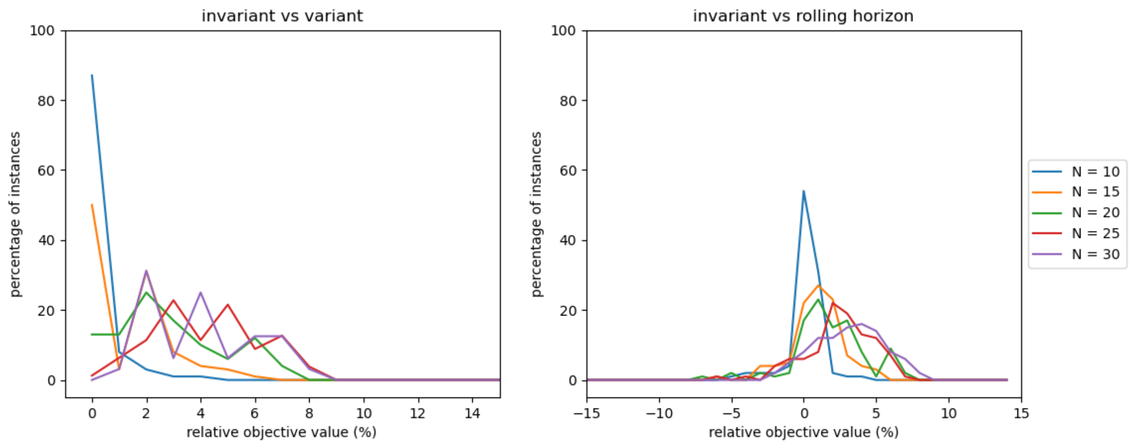


FIGURE 21: The percentage of instances for which the solution obtained by the time-invariant version deviates from the optimal solution (left) and rolling horizon solution (right) by a given percentage.

We suggest using a rolling horizon strategy for the time-variant QCAP as it obtains better results than the time-invariant QCAP in most instances in a time limit of 10 minutes. Since this stage 1 still needs to be integrated with stage 2 and 3 to obtain a schedule for the BACASP, the total computation time will increase a bit more. In case a terminal receives 30 to 40 vessels in a day, using the time-invariant QCAP would still guarantee a decent solution within 10 minutes, while this is not true for the rolling horizon. However, the terminal on which our data is based on cannot handle more than 30 vessels within 24 hours, meaning that the arrival of more than 30 vessels will certainly cause a queue buildup with the vessels arriving the next day. In order to use the rolling horizon we suggest starting the first segment with only the 30 earliest vessels and adding the remaining vessels from the second segment onward.

5 Berth allocation (stage 2)

After assigning the quay cranes and berthing time to each vessel, we need to check if a feasible allocation of berthing locations exists for this solution to stage 1. The berth allocation problem (BAP) is usually categorized based on the characteristics of the berths and arrival times. This leads to the continuous and discrete BAP depending on the berths, and the static and dynamic BAP depending on the arrivals. Sometimes, the draft of vessels in relation to the water depth is also taken into account [2].

In the discrete BAP the quay is split into several berths, where each berth is occupied in its entirety by a vessel. This allows for a very simply discretization of the quay, but it limits the possible schedules, since both a large vessel and a small vessel fully occupy a berth even though two small vessels could fit in it. The continuous BAP removes this limitation and allows vessels to be moored anywhere along the quay. The continuous BAP is modelled either by discretizing it into many small segments (e.g. 10 meters) or by using the relative positions of vessels. The MILP formulations associated with these are the position assignment formulation (see [7, 13, 15, 19, 21, 22, 23]) and the relative position formulation (see [9, 10, 14, 16, 17, 20, 24, 48, 49, 50, 51, 52]) respectively. In recent research, the relative position formulation is more popular, because it has fewer variables and fewer constraints unless the number of vessels is larger than the number of berth segments.

The static BAP assumes that all vessels to be scheduled have arrived at the start of the time horizon. This assumption was mainly made when research into the BAP just started, but it is not used much anymore [2]. In the dynamic BAP each vessel has an arrival time. These arrival times are actually estimated arrival times, since a ship can easily arrive earlier or later than expected depending on the weather, other ships, bridges and many other factors. Hence, research into the dynamic BAP is split into research with the assumption that the arrival times are deterministic and research where they are probabilistic [2, 49].

Our model for the second stage in our 3-stage model is based on the relative position formulations for the dynamic berth allocation problem. This allows for a truly continuous berth as well as inclusion of the arrival times, which is considered a necessity for a realistic model. We do not take the draft into account, since inland terminals predominantly deal with barges, which have a very small draft and thus fit anywhere along the quay. These decisions result in a formulation for stage 2 that is exactly the same as the model for the second stage of our 2-stage model for 2D-Knapsack given in (4).

5.1 Relative position formulation (RPF)

The relative position formulation prevents vessels from being in the same location along the berth at the same time through the relative positions of vessels. These are then used in a series of non-overlapping constraints. Two vessels can either occupy the same berthing location at different times, meaning that one vessel has to finish berthing before the other starts, or two vessels can be berthed at the same time so long as they do not occupy the same berthing location. Contrary to the normal version, the relative positions in our formulation only need to be determined in a single dimension (spatial), since the other dimension (temporal) has already been determined in the first step of the 3-stage approach. The decision variables and parameters are shown in Table 26.

Table 26: The parameters (above) and decision variables (below) for the second step in the 3-step approach.

S_i	The set of feasible berthing locations for vessel i: $S_i = [0, L - s_i]$.
z_{ij}	Equals 1 if and only if vessel i completes service before vessel j starts.
M	A sufficiently large number ($M \geq L$).
b_i	The berthing location of the front end of vessel i.
w_{ij}	Equals 1 if and only if vessel i is berthed below vessel j.

The z_{ij} parameters follow from the start and completion times determined in the first step. If the completion time of vessel i is before the start time of vessel j, $z_{ij} = 1$, and vice versa. If there is overlap between the time frame in which vessel i and j are served, $z_{ij} = 0$. The parameter does not exist for $i = j$.

$$\begin{aligned}
\min \quad & 0 && && \text{(55a)}\\
\text{s.t.: } b_j + M(1 - w_{ij}) &\geq b_i + s_i && \forall i, j \in V, i \neq j && \text{(55b)}\\
w_{ij} + w_{ji} + z_{ij} + z_{ji} &\geq 1 && \forall i, j \in V, i \neq j && \text{(55c)}\\
w_{ij} &\in \{0, 1\} && \forall i, j \in V, i \neq j && \text{(55d)}\\
b_i &\in S_i && \forall i \in V && \text{(55e)}
\end{aligned}$$

The objective is to find a feasible solution. Constraint (55b) ensures w_{ij} can only equal 1 if vessel i is berthed below j. Constraint (55c) requires that vessel i must complete service before j starts (or vice versa), i must be berthed below j (or vice versa), or both. Hence, if vessels i and j overlap in handling time, one must be berthed below the other. This ensures that no two vessels can occupy a piece of the berth at the same time. Constraints (55d) and (55e) define the decision variables for the berthing location and spatial overlap respectively.

5.2 Computational results

Our goal is to get a good schedule in a short amount of time. The second step is completed considerably faster than the first step in both the average and worst instances, as can be seen in Figure 22. All 500 instances of up to 30 vessels were completed within 10 seconds. In the case that 40 vessels could arrive on the same day, the berth allocation was completed within 150 seconds for all instances. Note that stage 2 needs to be solved repeatedly to obtain a solution to stage 1 for which a solution to stage 2 also exists. The computation times shown in Figure 22 are only for a single iteration of stage 2. The impact repeated solving of stage 2 has on the total time taken to obtain a schedule to the BACASP is discussed in Section 7.

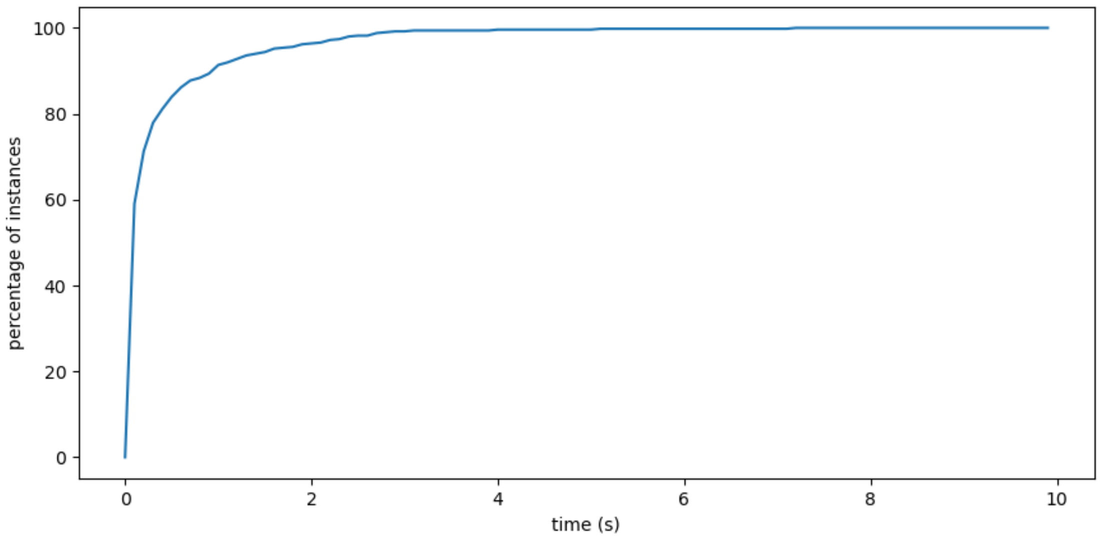


FIGURE 22: The percentage of instances for which step 2 was completed within a given time t. The computation time was tested over a total of 500 instances, 100 each for 10, 15, 20, 25, and 30 vessels. The start time and completion time of vessels that were used as input were provided by the heuristic in Section 4.3.

6 Specific quay crane assignment (stage 3)

The third step in the 3-stage approach assigns specific quay cranes to the vessels based on their quay crane assignment and berth allocation from the previous steps. An important aspect to take into account is that quay cranes cannot pass each other, leading to the requirement of a non-crossing constraint. The specific quay crane assignment is always solved in conjunction with the quay crane assignment, and can be incorporated into it in various ways. Park and Kim [15] and Meisel and Bierwirth [20] solve the QCAP and specific QCAP sequentially. The cranes are assigned with the goal to minimize the total number of times a crane starts handling a new vessel (the number of setups). Cheimanoff et al. [17], Ji et al. [23], Correcher [16] and Abou Kasm et al. [22] restrict possible schedules by requiring that if a crane starts handling a vessel, then it cannot handle a different vessel until the former has loaded/unloaded all its containers. They do not incorporate crane setup time or crane movement time into their models, although Abou Kasm et al. [22] discuss a possible method to add it to their model.

A crane setup does not only include the basic setup consisting of communication between the crane handler and workers on the ground and some checks, but it also includes the time taken to move the crane to the correct position on the quay. We consider three different mixed integer programs that minimize the number of setups, the total distance covered during setups, and the total setup time respectively. We do not restrict a crane from moving to handle a different vessel while its current vessel remains berthed. All three models use the same input parameters. These are given in Table 27.

Table 27: The parameters for the specific quay crane assignment.

V_t	The set of vessels berthed at time t (from step 1) sorted by b_i in increasing order.
QC_{it}	The number of cranes used by vessel i at time t (from step 1).
b_i	The berthing location of the center of vessel i (from step 2).
T_{max}	The end of the time horizon.
T	The set of time steps: $T := \{1, ..., T_{max}\}$.
N	The number of vessels.
V	The set of vessels: $V := \{1, ..., N\}$.
Q_{max}	The total number of quay cranes.
$\mathcal{Q}$	The set of quay cranes: $\mathcal{Q} := \{1, ..., Q_{max}\}$.
L	The total berth length.

The parameters V_t and QC_{it} are derived from the results of the first step of the 3-step approach, and b_i is derived from the second step. V_t is determined based on the start time and completion time of each vessel's service; vessel i is added to set V_t if and only if t is between the start and completion times of vessel i. It is sorted by increasing berthing location. Q_{it} follows directly from the y_{iqt} variables in both the time-variant and time-invariant formulations. The berthing location b_i is equal to the variable of the same name in the relative position formulation. The other parameters follow from the terminal specifications or the instance.

6.1 Number of setups

This formulation minimizes the number of crane setups needed for the schedule.

Table 28: The decision variables for the formulation to minimize the number of setups.

x_{iat}	Equals 1 if and only if crane a is handling vessel i at time t.
C_{iat}	Equals 1 if and only if crane a starts handling vessel i at time t.
y_{iat}	Equals 1 if and only if crane a is the leftmost crane handling vessel i at time t.

$$\min \quad \sum_{i\in V}\sum_{a\in \mathcal{Q}}\sum_{t\in T} C_{iat} \tag{56a}$$

$$\text{s.t.: } x_{ia0} = 0 \qquad \forall i \in V, \forall a \in \mathcal{Q} \tag{56b}$$

$$\sum_{a\in\mathcal{Q}} x_{iat} = QC_{it} \qquad \forall i \in V, \forall t \tag{56c}$$

$$\sum_{a=1}^{Q_{max}-QC_{it}+1} y_{iat} = 1 \qquad \forall i \in V, \forall t \in \{start_i, ..., c_i\} \tag{56d}$$

$$\sum_{a=1}^{Q_{max}-QC_{it}+1} y_{iat} = 0 \qquad \forall i \in V, \forall t \notin \{start_i, ..., c_i\} \tag{56e}$$

$$\sum_{b=\max(1,a-QC_{it}+1)}^{\min(a,Q-QC_{it}+1)+1} y_{ibt} = x_{iat} \qquad \forall i \in V_t, \forall a \in \mathcal{Q}, \forall t \tag{56f}$$

$$\sum_{i\in V} x_{iat} \leq 1 \qquad \forall i \in V, \forall a \in \mathcal{Q}, \forall t \tag{56g}$$

$$\sum_{a=1}^{Q-QC_{i_v t}+1} a y_{i_v at} + QC_{i_v t} \leq \sum_{a=1}^{Q-QC_{i_{v+1} t}+1} a y_{i_{v+1} at} \qquad \forall i_v \in V_t, v = 1, ..., len(V_t)-1, \forall t \tag{56h}$$

$$C_{iat} \leq x_{iat} \qquad \forall i \in V, \forall a \in \mathcal{Q}, \forall t \tag{56i}$$

$$C_{iat} \leq 1 - x_{iat(t-1)} \qquad \forall i \in V, \forall a \in \mathcal{Q}, \forall t \tag{56j}$$

$$C_{iat} \geq x_{iat} - x_{ia(t-1)} \qquad \forall i \in V, \forall a \in \mathcal{Q}, \forall t \tag{56k}$$

$$x_{iat}, y_{iat}, C_{iat} \in \{0, 1\} \qquad \forall i \in V, \forall a \in \mathcal{Q}, \forall t \tag{56l}$$

The objective is to minimize the total number of setups. Constraint (56b) defines that all cranes are unoccupied at the start of the time horizon. Constraint (56c) fixes the number of cranes handling each vessel at each time to the number determined in step 1. Constraints (56d) and (56e) assign a single feasible crane to be the 'lowest numbered' crane to handle vessel i at time t. Constraint (56f) determines if a crane is handling vessel i at time t based on the leftmost crane handling it, and constraint (56g) limits the number of vessels a crane can handle in a single time step to 1. Constraint (56h) ensures that cranes cannot pass each other along the quay. Constraints (56i), (56j), and (56k) ensure that a setup occurs if and only if the crane is handling vessel i at time t and did not do so the time step before. The decision variables are defined by constraint (56l).

6.2 Crane movement distance

This formulation minimizes the total amount of distance moved by the quay cranes across the quay when handling a new vessel. Two assumptions are made in this formulation. Firstly, when a quay crane is handling a vessel, its position is at the center of the vessel. Secondly, the quay cranes start evenly spaced along the quay at the beginning of the time horizon.

Table 29: The decision variables for the formulation to minimize the total distance covered by quay cranes during setup.

x_{iat}	Equals 1 if and only if crane a is handling vessel i at time t.
y_{iat}	Equals 1 if and only if crane a is the leftmost crane handling vessel i at time t.
l_{at}	The location of crane a at time t.
d_{at}	The distance crane a moved between time $t-1$ and t.

This formulation has constraints (56b)–(56h) and (56l) for the x and y variables from Section 6.1 in addition to the constraints below. The objective has also changed.

$$\min \quad \sum_{a\in\mathcal{Q}}\sum_{t\in T} d_{at} \tag{57a}$$

$$\text{s.t.: } l_{a0} = \frac{L}{Q}(a-0.5) \qquad \forall a\in\mathcal{Q} \tag{57b}$$

$$\sum_{i\in V_t} b_i x_{iat} + l_{a(t-1)}(1-\sum_{i\in V_t} x_{iat}) = l_{at} \qquad \forall a\in\mathcal{Q},\forall t \tag{57c}$$

$$d_{at} \geq l_{at} - l_{a(t-1)} \qquad \forall a\in\mathcal{Q},\forall t \tag{57d}$$

$$d_{at} \geq l_{a(t-1)} - l_{at} \qquad \forall a\in\mathcal{Q},\forall t \tag{57e}$$

$$l_{at}, d_{at} \in \{0,...,L\} \qquad \forall a\in\mathcal{Q},\forall t \tag{57f}$$

The objective is to minimize the total distance travelled by the quay cranes during setup. Constraint (57b) places the quay cranes evenly spaced along the quay at the start of the time horizon. Constraint (57c) sets the location of crane a at time t to its previous location if it does not handle any vessel at time t, or to the location of the vessel it is handling. Constraints (57d) and (57e) define the distance covered during setup between time $t-1$ and t.

Constraint (57c) is quadratic, which greatly slows down the computation time. Hence, it is linearized in by constraints (58) according to the method described in [53]. Constraints (58a) and (58b) set $l_{at} = l_{a(t-1)}$ if crane a is unoccupied at time t. This is also a valid solution to constraints (58c) and (58d). If crane a is handling some vessel i at time t, constraints (58c) and (58d) set $l_{at} = b_i$, which similarly does not violate constraints (58a) and (58b).

$$\sum_{i\in V_t} b_i x_{iat} + l_{a(t-1)} - l_{at} \leq L\sum_{i\in V_t} x_{iat} \qquad \forall a\in\mathcal{Q},\forall t \tag{58a}$$

$$\sum_{i\in V_t} b_i x_{iat} + l_{a(t-1)} - l_{at} \geq 0 \qquad \forall a\in\mathcal{Q},\forall t \tag{58b}$$

$$\sum_{i\in V_t} b_i x_{iat} - l_{at} \leq 0 \qquad \forall a\in\mathcal{Q},\forall t \tag{58c}$$

$$\sum_{i\in V_t} b_i x_{iat} - l_{at} \geq -L\left(1-\sum_{i\in V_t} x_{iat}\right) \qquad \forall a\in\mathcal{Q},\forall t \tag{58d}$$

6.3 Total setup time

The setup time of a crane consists of a basic setup that is necessary every time a crane starts handling a new vessel, and travel time. Travel time occurs only if the crane needs to move to a different position on the quay. Hence, the total setup time is a linear combination of the number of setups and the distance covered. The basic setup is assumed to take 5 minutes, and the travel speed of a quay crane is set to 5 kilometers per hour.

The formulation for the total setup time has a set of decision variables and constraints in addition to all constraints from the total distance formulation, as well as a different objective. The additional decision variable is S_{at}, which equals 1 if and only if crane a starts handling some vessel at time t. This is decision variable can be seen as $\sum_{i \in V} C_{iat}$ from Section 6.1. The constraints to set this variable to 1 as needed are

$$S_{at} \geq x_{iat} - x_{ia(t-1)} \qquad \forall i \in V_t \forall a \in \mathcal{Q}, \forall t \in T \qquad (59a)$$

$$S_{at} \in \{0,1\} \qquad \forall a \in \mathcal{Q}, \forall t \in T. \qquad (59b)$$

No constraint is needed to set it to zero if $x_{iat} - x_{ia(t-1)} = 0$ or -1, because the minimization will automatically set it to zero whenever possible. The new objective is

$$\min \frac{1}{5000} \sum_{a \in \mathcal{Q}} \sum_{t \in T} d_{at} + \frac{1}{12} \sum_{a \in \mathcal{Q}} \sum_{t \in T} S_{at}. \qquad (60)$$

6.4 Computational results

In this section we give a comparison of the time the solver takes to prove optimality. We also show the disadvantages of disregarding either the setup time or the movement time in the created schedule. We tested all three models for 100 instances for 10, 15, 20, 25 and 30 vessels. The inputs were generated by taking the solution of the WARMSTART algorithm as the solution to stage 1 and using this to obtain a feasible solution to stage 2. The results are shown in Figure 23. It can clearly be seen that an optimal solution is found within 30 seconds for all three models, with the model minimizing the number of setups having a maximum computation time of less than a second.

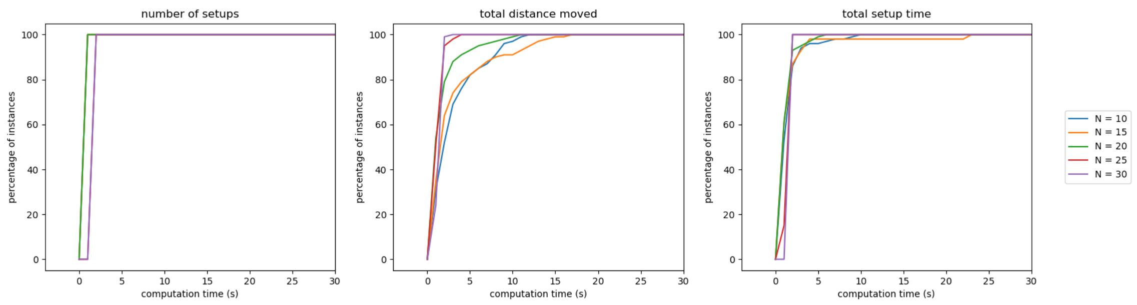
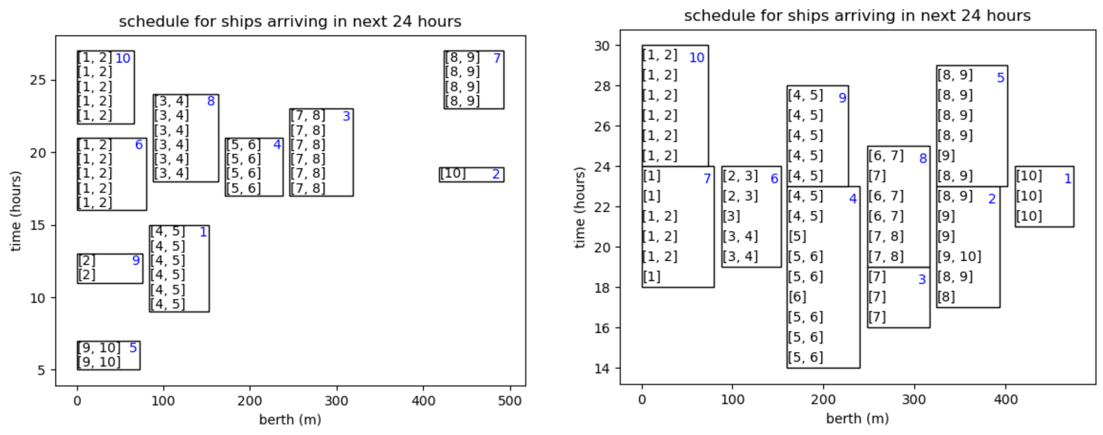


FIGURE 23: The percentage of instances that were solved to optimality within a given time t for the three different models from Section 6.1 (left) Section 6.2 (middle), and Section 6.3 (right).

If we look at the middle and right figures, we see that the solving time actually shows the opposite relation to the number of vessels compared to stage 1 and 2: all formulations for the quay crane assignment problem show an increase in computation time as a function of the number of vessels, while the models for the specific quay crane assignment show a decrease. This intuitively makes sense. As the number of vessels increases, the number of quay cranes occupied at each time step also increases; this limits the possible quay crane schedules that do not violate the non-crossing constraint. Hence, an increase in the number of vessels limits the possible schedules, thus improving the time it takes to prove optimality of a schedule.



(A) number of setups (B) amount of distance

FIGURE 24: Schedules made when minimizing only the number of setups (A) or minimizing only the amount of distance the cranes moved (B). Each vessel is represented by a rectangle, with the vessel number in the upper right corner. The specific cranes handling the vessel at each time step are given in brackets.

Although the solver is done quickly for all three models, the resulting schedules show flaws characterized by their formulations. Two such schedules are given in Figure 24. In the schedule in Figure 24a, vessel 5 is handled by cranes 9 and 10 even though these cranes are normally situated on the right side of the quay. They later have to cross the entire quay to handle vessels 2 and 7. Since both cranes 1 and 2 were also available during the time vessel 5 was berthed, this would have been a far more sensible choice. Hence, not including the distance the cranes need to move to get to a vessel can waste crane resources by wasting time moving the cranes around. Similarly, not including the number of setups will also waste crane resources. Vessel 2 in Figure 24b is first handled by crane 8, then 8 and 9, and then 9 and 10. However, it could have been handled by 9, 9 and 10, and 9 and 10. This way the number of setups is reduced by 1. Hence, we consider a combination of the two aspects to be necessary to prevent schedules that seem somewhat dumb or wasteful to a person looking at them.

7 Integration of the 3 stages

We have discussed various models for stage 1, 2, and 3 in Sections 4 to 6. We have chosen a single 'best' model for each stage based on the time taken by the solver to prove optimality and the objective values obtained by the model. Recall that for stage 1 we chose formulation B_2 for the time-variant quay crane assignment model with a rolling horizon with 8 hours of overlap and without rescheduling, and if an even shorter computation time was needed we chose formulation INV_1 for the time-invariant QCAP. We only considered a single model for stage 2, which showed to be sufficiently fast. We chose to minimize the total setup time in stage 3, because both of the other objective functions have idiosyncrasies in the created schedules.

This section discusses how we will link stage 1 and 2 to obtain a solution that is feasible in both stages. We also discuss the possibilities of integrating stage 2 and 3 in order to obtain a berth allocation that improves the objective in stage 3 compared to solving the two sequentially.

7.1 Integration of stage 1 and 2

Stage 1 and 2 of our model have to be linked in some manner to guarantee obtaining a feasible schedule. If the two are performed sequentially instead, it is possible that stage 2 is infeasible. Although the total berth length occupied by the berthed vessels can never exceed L by (34g), it is possible that there is no possible placement of the vessels without allowing a vessel to move to a different position along the quay. Although this is done in some terminals [30], it is not common practice, so we do not include it in our model (see assumption 8).

We consider two methods of linking stage 1 and 2. Firstly, we consider a feedback loop where stage 1 and 2 are carried out sequentially. This means we first obtain an optimal solution to stage 1, and then use it as input to stage 2. If stage 2 is infeasible, we rerun the stage 1 model with one additional constraint that rejects the previously found solution. Secondly, we consider the use of a constraint handler in the solver. Every time the solver finds a potential new incumbent solution to stage 1, it calls the constraint handler; it then checks if stage 2 is feasible. If it is feasible, the solver updates the primal incumbent solution with the new solution and continues. If it is infeasible, a constraint is added that cuts off the suggested solution to stage 1, and the solver continues.

Both methods have advantages and disadvantages. The feedback loop only checks the optimal solution to stage 1, limiting the number of times it gets called. If the first optimal solution found is also feasible in stage 2, this is as fast as implementing the stages sequentially. However, the solver reruns the entire the solution algorithm for stage 1 every time a solution to stage 1 is rejected, which can lead to a large delay if several solutions are infeasible in stage 2. If we implement a time limit on the combination of the stages, it is even possible that no feasible solution has been found at all. The constraint handler does not have this problem, as any incumbent primal solution found is feasible for the second stage. However, it has to check the feasibility of stage 2 for every new potential primal incumbent solution to stage 1. Depending on how often this happens, it can significantly increase the running time of the solver. Ideally, we would have liked to implement a combination of the two methods by using a constraint handler that is only activated when the potential primal incumbent solution has a small gap. As far as we could find, the constraint handler only checks potential solutions when they are first found, which might have a large gap even if the solution is actually optimal, because the dual bound is not tight. Although there may be a workaround possible, this is left as a possible future direction of research.

In both methods we reject a solution to stage 1 by adding a constraint that makes the solution infeasible. Since the second stage assigns berthing locations to vessels, it has nothing to do with how the quay cranes were assigned in the solution. Thus, any infeasibility in stage 2 must be caused by which vessels are berthed at which time steps. In the time-variant version of stage 1, this directly follows from b_{it}. Suppose we have some solution b' to stage 1, where $B := \{(i,t) : b'_{it} = 1\}$ is the set of (i,t) pairs where vessel i is berthed at time t, then the inequality

$$\sum_{(i,t)\in B} b_{it} \leq |B| - 1 \tag{61}$$

makes this solution infeasible. To be precise, it makes all quay crane assignments for which $b_{it} = 1$ for all $(i,t) \in B$ infeasible, since all of these solutions would be infeasible in stage 2. It does not affect the feasibility of any other primal solution, because any other solution must have at least one $(i,t) \in B$ for which $b_{it} = 0$ thus satisfying (61). It can affect the LP relaxation, but that only makes the dual bound tighter, which is a positive effect.

In the time-invariant version of stage 1, which vessel is berthed at which time is given by $\sum_{q \in Q_i} x_{iqt}$, but x follows directly from y. Let y' be a solution to stage 1 with $Y := \{(i,q,t) : y'_{iqt} = 1\}$, then adding

$$\sum_{(i,q,t) \in Y} y_{iqt} \leq |Y| - 1 \tag{62}$$

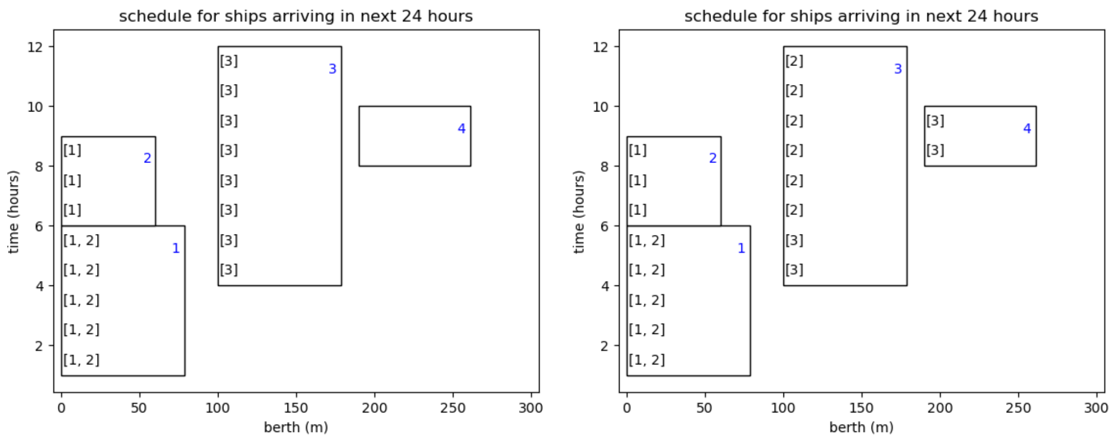
will make this (and only this) solution infeasible.

7.2 Integration of stage 2 and 3

One reason given in literature (see [54, 16]) for requiring integration of stage 2 and 3 is because stage 3 could be infeasible if solved sequentially. This happens because they have an additional requirement on the specific quay crane assignment; namely that it is not allowed for a crane handling a vessel to be switched by another. This limits the feasible solutions as is shown in the example given below.

Example 7.1. Consider a solution to an instance to the time-invariant BA-CAP with 4 vessels, a total berth length $L = 300$ and 3 available quay cranes where the start time, completion times, berthing locations and quay cranes are assigned according to Example 7.1. Then the resulting specific crane assignment problem is infeasible if the restriction is applied. This is shown in Figure 25.

vessels	1	2	3	4
size (m)	79	60	79	71
berthing location (m)	0	0	100	190
start time (hours	1	6	4	8
completion time (hours)	6	9	12	10
quay cranes	2	1	1	1

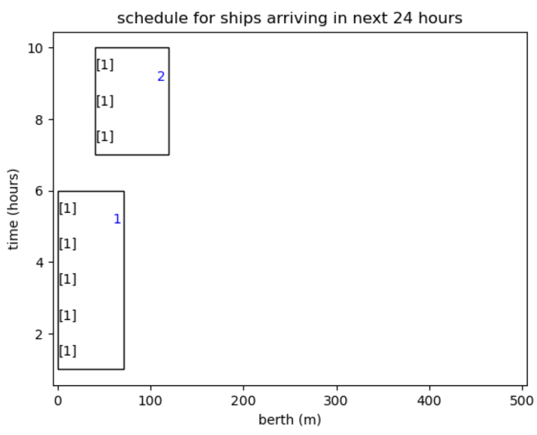


(A) with restriction (B) without restriction

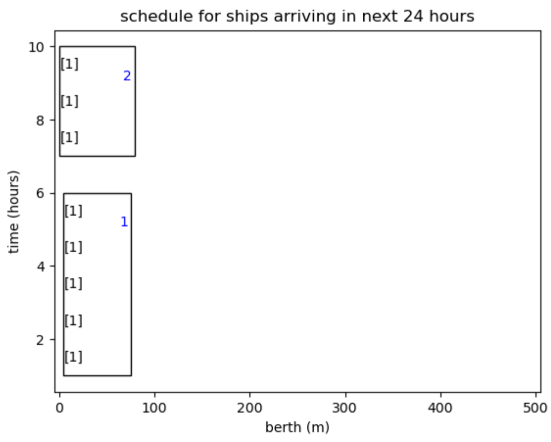
FIGURE 25: The schedule visualisation of a solution to the BACAP that does not have a feasible specific crane assignment if the restriction is applied (A), while it does if the restriction is not applied (B). Each vessel is represented by a rectangle, with the vessel number in the upper right corner. The specific cranes handling the vessel at each time step are given in brackets.

The same articles that place this additional requirement on the specific quay crane assignment also claim that the crane setup time is negligible compared to the time it takes to handle a vessel, and they therefore ignore it. We find the combination of restricting crane movement whilst simultaneously claiming it to be negligible to be a bit contradictory, but we suppose it was done to limit the possible influence of something they decided to ignore. All articles mentioning this restriction are solving the time-invariant BACASP; this restriction then ensures that all crane setups occur during the berthing/unberthing of a vessel. Hence, all crane setups could be ignored with impunity because berthing takes longer than setup. We also ignore the influence of the crane movement time on the feasibility of our obtained solution to stage 1 of the model. Any crane setup while a vessel either starts or finishes service can be ignored as vessel berthing takes longer than crane setup, provided that the schedule does not require a crane to move across the whole quay. The influence of a crane setup midway during service can be mitigated by adding a few containers to the actual container numbers of each vessel. This also creates a buffer if the crane productivity falls below the average we are working with.

We consider integrating stage 2 and 3 not due to this restriction, but because integrating the stages can lead to an improvement in the specific quay crane schedule. Stage 2 finds any feasible schedule, meaning that this schedule can lead to unnecessary crane movement or setups. An example of this is shown in Figure 26, where it can be seen that crane 1 does not need to move at in between handling vessels 1 and 2 in Figure 26b, whereas it does in Figure 26a.



(A) sequential



(B) integrated

FIGURE 26: Schedules made when executing stage 2 and 3 sequentially (A) or integrated (B).

Integrating stage 2 and 3 is not as simple as making one formulation with the constraints from both models, because the models for stage 3 use the locations assigned in stage 2 as parameters. Since these locations become variables in the integrated version, constraints (58) become quadratic constraints instead of linear constraints. Additionally, constraint (56h) must be reformulated. Let V_t be the *unsorted* set of vessels berthed at time t. Let w_{ij}, b_i and z_{ij} be as in Table 26. We then add all constraints from (55) to the model from Section 6.3, replacing the inequality in (55c) by equality to redefine $w_{ij} := 1$ if and only if vessels i, j are berthed at the same time and i is berthed below j.

To remove the requirement that V_t is sorted by berthing location, we replace constraint (56h) by (63). We use an additional variable $A_{ijt} := 1$ if and only if vessels i, j are both berthed at time t and $b_i > b_j$. The first part is only true if $t \in T_{ij}$ with $T_{ij} := \{\max(start_i, start_j), \min(c_i, c_j)\}$, and the second part is only true if $w_{ji} = 1$. We then add (63b) to ensure that the cranes cannot cross each other. This constraint was derived from the work of Correcher [16], where it was implemented for a time-invariant model.

$$A_{ijt} = w_{ji} \qquad \forall i \neq j \in V, t \in T_{ij} \tag{63a}$$

$$\sum_{a=1}^{Q_{max}-QC_{it}+1} y_{iat} \geq \sum_{a=1}^{Q_{max}-QC_{jt}+1} y_{jat} + QC_{jt} - Q_{max}(1 - A_{ijt}) \quad \forall i \neq j \in V_t, \forall t \in T \tag{63b}$$

$$A_{ijt} \in \{0, 1\} \qquad \forall i \neq j \in V_t, \forall t \in T \tag{63c}$$

In order to linearize the term $\sum_{i \in V_t} b_i x_{iat}$ in (58), we introduce F_{at}, which equals 0 if crane a is not handling any vessel at time t or b_i if it is handling vessel i. This is done through (64). For the MILP, F_{at} is equal to $\sum_{i \in V_t} b_i x_{iat}$, and it is thus valid to replace the summation by F_{at} in (58). Note that this replacement is only needed when the objective includes minimizing the distance the crane move during service.

$$F_{at} \leq L \sum_{i \in V_t} x_{iat} \qquad \forall a \in Q, \forall t \in T \tag{64a}$$

$$F_{at} \leq b_i + 0.5 s_i + L(1 - x_{iat}) \qquad \forall a \in Q, \forall t \in T \tag{64b}$$

$$F_{at} \geq b_i + 0.5 s_i - L(1 - x_{iat}) \qquad \forall a \in Q, \forall t \in T \tag{64c}$$

$$F_{at} \in \{0, ..., L\} \qquad \forall a \in Q, \forall t \in T \tag{64d}$$

7.3 Computational results

In this section we discuss all computational results pertaining to the integration of the three stages into the 3-stage approach. We first compare the computation time of the two methods for integrating stage 1 and 2. This is followed by a comparison in terms of computation time and objective value for the integration of stage 2 and 3. Finally, we show the time taken by the 3-stage approach, split over each stage. This is joined by a short discussion on the turnaround time and waiting time of the produced schedules.

The possible advantages and disadvantages of both the feedback loop and the constraint handler were discussed in Section 7.1. The results shown in Figure 27 follow our expectations. Both methods are comparable for all instances, except in cases where the solver takes a long time to prove optimality in a segment, only to have that solution rejected. In these cases the feedback loop does not find a feasible solution within the time limit of 10 minutes, because stage 1 has to be resolved in its entirety each time a solution is rejected. If this happens several times, the computation time quickly increases. There is no notable difference in the computation time of the feedback loop and constraint handler in instances where no solution to stage 1 is rejected. Additionally, the computation time using the constraint handler is similar to that of just stage 1 shown in Figure 19. This means that there is little room for improvement in the linking between stage 1 and 2.

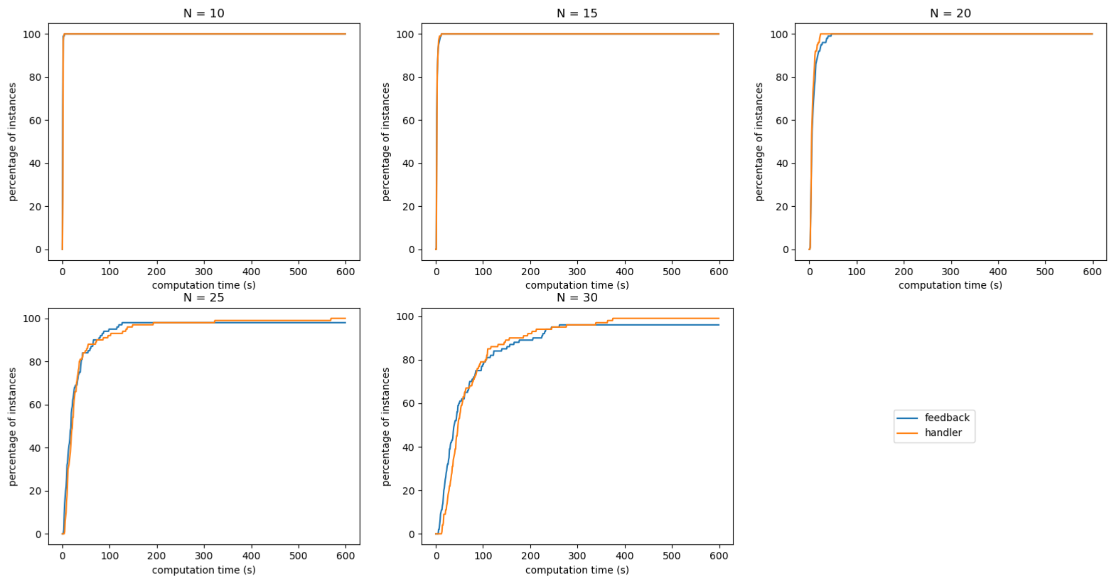
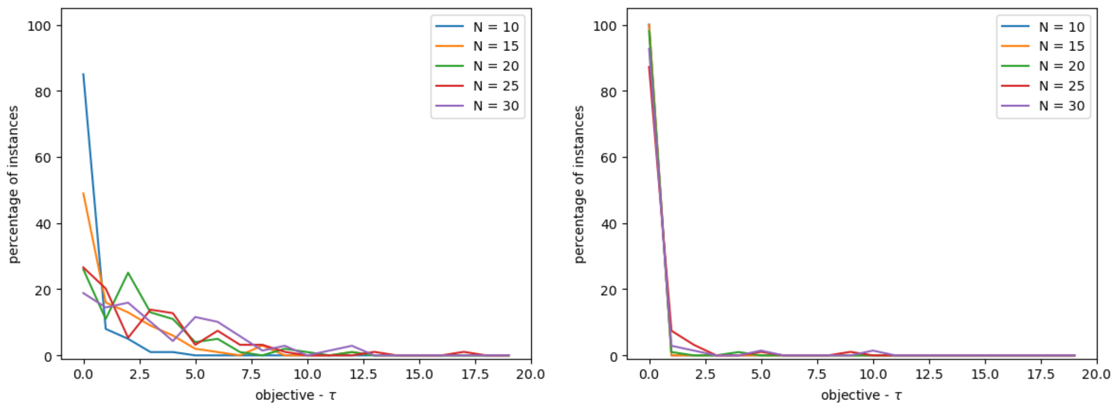


FIGURE 27: The percentage of instances with computation time within a given time t for instances with 10, 15, 20, and 25 vessels when the time-variant model for stage 1 was linked with stage 2 through a feedback loop or a constraint handler.

We integrated stage 2 and 3 to improve the specific quay crane schedule with regard to the number of setups and the total setup time. The number of crane setups needed for a given solution to stage 1 has a theoretical minimum. Each time the number of cranes handling a vessel increases by some number, an equal number of crane setups are needed. Any other crane setups are cases where one crane replaces another while the number of cranes handling the vessel at that time does not change. These setups are unwanted. We investigate the difference in the number of unwanted setups when solving stage 2 and 3 sequentially or in an integrated manner. The results are shown in Figure 28. It can be seen that the number of unwanted setups greatly decreases by integrating stage 2 and 3. However, the integration also has several disadvantages. Firstly, the solver was unable to find a feasible solution in 6 instances with 25 vessels and 31 instances with 30 vessels within 10 minutes. Secondly, the average time taken by the solver to prove optimality for the instances where it found the optimum also shows a significant increase from less than 1 second to 30 seconds. One notable discovery is that it is possible that there does not exist a berth allocation such that the theoretical minimum is achievable. This occurred in only a single instance with 25 vessels where the optimum was found. Naturally, it is possible it is also true for instances where optimality was not proven (or no solution was found). Simply integrating the stage 2 and 3 is thus not sufficient if the restriction mentioned in Section 7.2 is desired.



(A) sequential (B) integrated

FIGURE 28: The (absolute) difference between the objective value obtained by solving stage 2 and 3 sequentially (A) or integrated (B) within 10 minutes and the theoretical minimum number of setups τ.

These disadvantages are even more prevalent for the integration of stage 2 with the total setup time formulation of stage 3. In this case, there is no clear theoretical minimum for the objective function, since it depends on both the number of setups and the distance the quay cranes need to move across the quay, so we compare the relative increase in the objective value obtained by solving the stages sequentially or through integration instead. Through preliminary investigation we found that no instances were solved to optimality within 10 minutes, but many instances with a small number of vessels obtained a solution with a gap smaller than 5% within that time limit. Hence, we decided to add another cut-off to the solver; it should stop if the gap was less than 5% or if the time limit of 10 minutes was reached. The integrated formulation was tested for 100 instances with 10 or 15 vessels and for 20 instances with 20, 25, or 30 vessels (a total of 260 instances). The results are shown in Table 30. It is clear that the integrated formulation is not usable with a time limit of 10 minutes, because no solution would be found for many instances with 20 or more vessels.

Table 30: The number of instances for which a solution was found within 10 minutes and the number of instances for which a solution with a gap less than 5% was found for the integrated formulation of stage 2 and 3 minimizing the total setup time.

N	# of instances	solved	gap $< 5\%$
10	100	100	97
15	100	99	86
20	20	17	11
25	20	12	3
30	20	1	0

We also compare the relative increase in the objective value with respect to the solution obtained when stage 2 and 3 are solved sequentially. The relative increase is defined as

$$100 \cdot \frac{z_{int} - z_{seq}}{z_{seq}}, \tag{65}$$

where z_{int} is the objective value obtained through the integrated formulation, and z_{seq} is the objective value for the sequential formulation. The results are shown in Figure 29 for the instances where a solution was found for the integrated formulation. It can be seen that the relative increase is negative in the majority of the instances. This means that the objective obtained by solving the integrated version is better than that obtained by solving the stages sequentially even with the additional cut-off. Although the differences are concentrated between -10 and 0, there are instances that show significant improvements due to the integration of the stages. The few instances with a positive relative increase were caused by instances where the sequential model got 'lucky' and obtained a solution within 5% of the optimal solution, and instances where the integrated model was terminated due to the time limit with a large gap. Hence, we conclude that integration of the two stages results in a considerably better solution in most instances, but it is currently not usable in practice due to the large solving time.

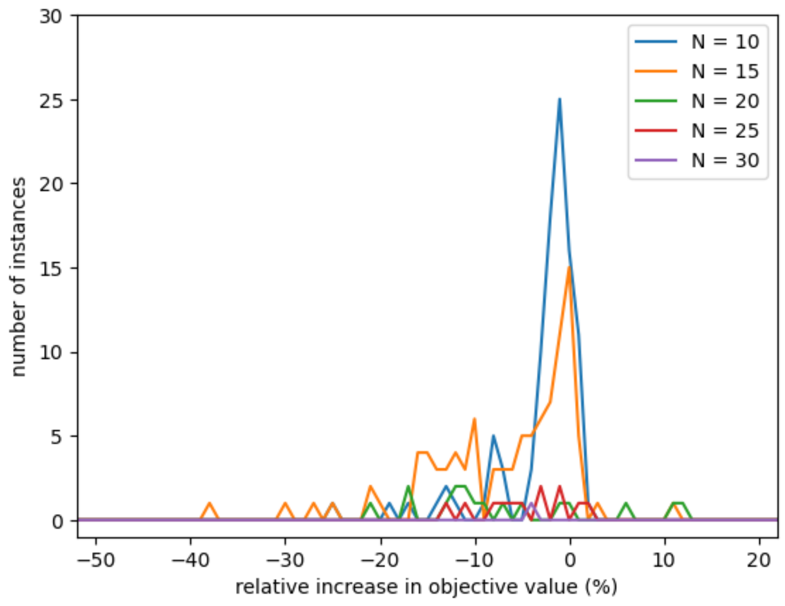


Figure 29: The relative increase in the objective value obtained integrating stage 2 with the total setup time model for stage 3 compared to solving stage 2 and 3 sequentially for instances where a feasible solution was found within 10 minutes for both models.

Finally, we evaluate the time taken during each stage of our 3-stage approach as well as the quality of the obtained solutions. The average and maximum time spent in each stage are given in Table 31. All times are rounded to whole seconds; everything smaller than 0.5s is displayed as < 1 instead of 0. Stage 1 in this table refers to the integration of stage 1 and 2, it thus includes the time spent in stage 2 checking feasibility. This time in stage 2 is less than a second, so we have included it in stage 1. It can clearly be seen that the stage that is the most time-consuming is dependent on the number of vessels. If only 10 or 15 vessels arrive, stage 3 takes up the most time both on average and in the worst case. If 20 or more vessels arrive, the most time-consuming stage becomes stage 1, which takes up significantly more time than stage 3 did for instances with 10 and 15 vessels. Stage 1 is the only stage that comes close to our time limit of 10 minutes. Hence, any future research regarding improvements to the computation time of the model should be solely focused on stage 1.

Table 31: The average and maximum time taken per stage for the time-variant model for each number of vessels N in an instance. The average and maximum were taken over 100 instances for each number of vessels N.

	stage 1 (s)		stage 2 (s)		stage 3 (s)	
N	mean	max	mean	max	mean	max
10	1	2	<1	<1	4	13
15	2	7	<1	<1	3	15
20	6	68	<1	<1	2	16
25	46	601	<1	3	1	6
30	136	623	1	6	1	3

In this thesis we minimized the total turnaround time, so we have displayed average and maximum turnaround and waiting times in Table 32. Since we are minimizing the total turnaround time, we expect that the model behaves similarly to scheduling problems that minimize the total completion time. In these models, it can be better to delay a job with a large processing time in favor of many jobs with a shorter processing time. We indeed see the same behavior here. The average turnaround and waiting times are very nice. On average, vessels only need to wait less than 2.5 hours to begin service even if the instance contains 30 vessels. On the contrary, the maximum turnaround and waiting times are quite large; vessels may need to more than 20 hours. Since all vessels are arriving in a 24-hour time frame and the vessels' handling time is below 10 hours, this is quite a long waiting time.

Table 32: The average and maximum turnaround time and waiting time of a vessel in hours per number of vessels N in an instance. The average and maximum were taken over 100 instances for each number of vessels N.

	turnaround time (h)		waiting time (h)	
N	mean	max	mean	max
10	5.48	14.7	0.54	4.77
15	5.49	19.95	0.64	10.95
20	5.86	23.49	0.94	13.49
25	6.66	31.51	1.64	22.51
30	7.23	36.63	2.29	25.63

8 Discussion

In this work, we have proposed a 3-stage model for the time-variant and time-invariant BACASP with partial integration between the stages in inland terminals. Additionally, we have presented a rolling horizon strategy to solve the model more efficiently. We have made two assumptions different from literature. Firstly, we include the berthing and unberthing time of a vessel in our time-variant model. To our knowledge, this has not been done before in the time-variant BACASP. Secondly, we do not model the quay crane interference using the function from Meisel and Bierwirth, but instead assume that the quay crane speeds are known to the terminal. We have also introduced a new type of valid inequalities for the two-dimensional knapsack problem, and evaluated the influence of these inequalities on our BACASP model.

BACASP

Many formulations for the time-variant and time-invariant BACAP were considered in this work, as well as three different models to assign specific quay cranes. Numerical experiments have shown that we can obtain an optimal solution to the time-invariant BACAP within 30 seconds and a solution within 10% of the optimum for the time-variant BACAP within 10 minutes. The solution to the time-variant BACAP is better than that of the time-invariant BACAP in most instances. An optimal assignment of specific quay cranes can then be found within a few seconds. The clear bottleneck to our 3-stage approach remains stage 1 regardless of whether the time-variant or time-invariant model is considered.

We considered only the total turnaround time as the objective to minimize. This results in schedules with a small average waiting time, but some vessels may have to wait for more than 20 hours. This can be undesirable for both the terminal and the vessel. There are several methods to mitigate this effect. The first solution is to include a desired finishing time and add a penalty for every hour that a vessel finishes service after this time. A second solution is to add a constraint that forces all vessels to have a waiting time below a certain value. This carries the risk that the BACAP becomes infeasible. A third solution is to switch the objective function to minimizing the maximum waiting time, or to a combination of the two objectives. Which objective is more suitable is largely dependent on the wishes of the terminal. All three of these solutions do require a new investigation into the time taken by the solver to prove optimality.

We have also investigated the possibility of fully integrating stage 2 and 3. The integration shows a clear improvement on the number of crane setups and total setup time needed in the resulting schedule. However, the time taken to obtain the optimal solution has greatly increased, making the integration ill-suited for practical use in its current form. We noticed that good solutions to the specific crane assignment problem tend to consist of several sets of vessels, where vessels in the same set berth at the same locations and use a similar number of quay crane used during service. Hence, we surmise that a set partitioning model could show good results. Alternatively, a heuristic using the aforementioned properties could also work well.

In this work we have only considered a terminal where the berth consists of 1 continuous piece. This is not necessarily true, as there are several clients of Cofano where the berth consists of 2 pieces, where each berth has its own quay cranes. We have adapted the model for one such terminal by adding a subscript designating which quay it belongs to for the y, QC^{start}, QC^{end}, and z-variables ($z-$variables were defined in Section 4.4). All constraints containing one of these variables must hold for all quays. The computational results are comparable to that of a terminal with a single berth.

Before our model is truly implementable in a terminal, there are still several aspects that need to be taken into consideration. In consultation with a client of Cofano we have pinpointed three issues. Firstly, our assumption that shifts connect seamlessly, and that personnel does not have breaks is unrealistic. The terminal in question has 15-minute breaks every 3 to 4 hours for all personnel, and a shift change takes up to 30 minutes. We propose including this in our model by changing the quay crane speed to be time-dependent and adapting constraints (34h)–(34n) as shown in (66). The idea of the adaptation is to include the 15-minute breaks in every fourth hour, and to include the shift changes every 12 hours. This means the effective crane speeds decrease by 25% or 50% for these hours. Let v_{qt} be the effective crane speeds for every $q \in Q, t \in T$ and let $v_{q0} = v_q$. We define T_b as the set of time steps that contain a break or shift change. Since both personnel breaks and berthing/unberthing take 15 minutes, breaks can nicely fit in the berthing/unberthing times. Hence, if the start or finishing time step of a vessel includes a break, the crane productivity of that hour is still only reduced by 25% and not by 50%. This also holds in case of a shift change. To include this in the model, we only set $QC_{iq}^{start}, QC_{iq}^{end}$ equal to 1 if q cranes are used during the start/end of service and this time step is not in T_b.

$$\sum_{t\in T}\sum_{q\in Q} v_{qt}y_{iqt} \geq Con_i + \frac{1}{4}\sum_{q\in Q} v_{q0}(QC_{iq}^{start} + QC_{iq}^{end}) \quad \forall i \in V \tag{66a}$$

$$\sum_{q\in Q} QC_{iq}^{start} \leq 1 \quad \forall i \in V \tag{66b}$$

$$\sum_{q\in Q} QC_{iq}^{end} \leq 1 \quad \forall i \in V \tag{66c}$$

$$QC_{iq}^{start} \geq y_{iqt} - b_{i(t-1)} \quad \forall q \in Q, \forall i \in V, \forall t \geq 2 \in T\backslash T_b \tag{66d}$$

$$Q_{iq}^{start} \geq y_{iq1} \quad \forall q \in Q, \forall i \in V \tag{66e}$$

$$QC_{iq}^{end} \geq y_{iqt} - b_{i(t+1)} \quad \forall q \in Q, \forall i \in V, \forall t \leq T_{max} - 1 \in T\backslash T_b \tag{66f}$$

$$Q_{iq}^{end} \geq y_{iqT_{max}} \quad \forall q \in Q, \forall i \in V \tag{66g}$$

The second aspect that needs to be taken into consideration is the berthing time. In practice, the exact berthing/unberthing time is dependent on the distance between the waiting area and the berth as well as the maneuverability of the vessel. Our model can easily be changed to make the berthing time vessel dependent by changing the factor of 0.25 in (34h) to a parameter dependent on the vessel.

The third aspect that needs to be taken into account is that we assume that we start with an empty berth. In practice, some vessels scheduled late today will only unberth some time tomorrow. Hence, some parts of the berth and some of the cranes will already be occupied at the start of the planning horizon. This can be included by changing Q_{max} and L to become time-dependent.

2D-Knapsack

Numerical experiments show that our 2-stage model for the 2D-Knapsack problem is significantly more stable than the two single stage models we considered, as well as being faster on average and in difficult instances. Including the lifted adjacent cover inequalities reduces the gap between the LP relaxation and the optimal solution by 50% on average for the 2-stage model. This reduction is larger than that caused by cover inequalities. We were unable to compare the computational effort required to include the adjacent cover inequalities as cutting planes, because we used SCIP through its Python wrapper. This has led to a poor integration of our separator with SCIP, thus greatly increasing the time taken by the solver even if a separator with no functionality was included. This is a possible direction for further research. A second direction is related to the linking process between stage 1 and 2. In this work, only a single solution to stage 1 is accepted or rejected. However, the actual set of item-positions causing the infeasibility can be narrowed down, thus rejecting several solutions to stage 1 at once, as was shown by Côté et al. [36] for the strip packing problem. They also use *normal patterns* to reduce the number of variables of the model. This has great potential in improving our model for larger instances in regard to both the width of the knapsack and the number of items. We note that in all observed instances where a solution to stage 1 was rejected, the LP relaxation of stage 2 remained feasible. A third direction is to consider up- and down-lifting instead of the simple uplifting considered in this work.

The results when applying the adjacent cover inequalities to stage 1 of the BACASP are less promising, as only a small percentage of instances showed any reduction in the gap. These results are similar to those of Correcher et al. [16], who included several types of inequalities as cutting planes. They only obtained a single additional instance that was solved within their time limit. We have several ideas as to why this might have happened. Firstly, the lack of diversity in the vessel sizes in the BACASP instances compared to the item sizes for the 2D-Knapsack instances causes the lifting coefficients to be 0 more often. Hence, this might be the wrong type of inequality to use on the LP-bound. Secondly, the LP relaxation of the time-invariant model was quite tight, leaving little room for improving the dual bound.

References

[1] U. N. C. on Trade and Development, *UNCTAD Handbook of Statistics 2021*. United Nations, 2021 ed., 2022.

[2] H. Carlo, I. Vis, and K. J. Roodebergen, "Seaside operations in container terminals: literature overview, trends and research directions," *Flexible services and Manufacturing Journal*, vol. 27, pp. 224–262, 2015.

[3] S. Jia, S. Li, X. Lin, and X. Chen, "Scheduling tugboats in a seaport," *Transportation Science*, vol. 55, 2021.

[4] C. Iris and D. Pacino, "A survey on the ship loading problem," pp. 238–251, 2015.

[5] G. Giallombardo, L. Moccia, M. Salani, and I. Vacca, "Modeling and solving the Tactical Berth Allocation Problem," *Transportation Research Part B: Methodological*, vol. 44, pp. 232–245, Feb. 2010.

[6] I. Vacca, M. Salani, and M. Bierlaire, "An Exact Algorithm for the Integrated Planning of Berth Allocation and Quay Crane Assignment," *Transportation Science*, vol. 47, pp. 148–161, May 2013.

[7] F. Meisel and C. Bierwirth, "Heuristics for the integration of crane productivity in the berth allocation problem," *Transportation Research Part E: Logistics and Transportation Review*, vol. 45, pp. 196–209, Jan. 2009.

[8] Iris, D. Pacino, S. Ropke, and A. Larsen, "Integrated Berth Allocation and Quay Crane Assignment Problem: Set partitioning models and computational results," *Transportation Research Part E: Logistics and Transportation Review*, vol. 81, pp. 75–97, Sept. 2015.

[9] W. Liu, X. Zhu, L. Wang, and S. Li, "Rolling horizon based robust optimization method of quayside operations in maritime container ports," *Ocean Engineering*, vol. 256, p. 111505, July 2022.

[10] Iris, D. Pacino, and S. Ropke, "Improved formulations and an Adaptive Large Neighborhood Search heuristic for the integrated berth allocation and quay crane assignment problem," *Transportation Research Part E: Logistics and Transportation Review*, vol. 105, pp. 123–147, Sept. 2017.

[11] J. Blazewicz, T. C. E. Cheng, M. Machowiak, and C. Oguz, "Berth and quay crane allocation: a moldable task scheduling model," *Journal of the Operational Research Society*, vol. 62, pp. 1189–1197, July 2011.

[12] G. Ilati, A. Sheikholeslami, and E. Hassannayebi, "A Simulation-Based Optimization Approach for Integrated Port Resource Allocation Problem," *PROMET - Traffic&Transportation*, vol. 26, pp. 243–255, June 2014.

[13] L. Xiao and Z.-H. Hu, "Berth Allocation Problem with Quay Crane Assignment for Container Terminals Based on Rolling-Horizon Strategy," *Mathematical Problems in Engineering*, vol. 2014, pp. 1–11, 2014.

[14] H. Zheng, Z. Wang, and H. Liu, "The Integrated Rescheduling Problem of Berth Allocation and Quay Crane Assignment with Uncertainty," *Processes*, vol. 11, p. 522, Feb. 2023. Number: 2 Publisher: Multidisciplinary Digital Publishing Institute.

[15] Y.-M. Park and K. H. Kim, "A scheduling method for Berth and Quay cranes," *OR Spectrum*, no. 25, pp. 1–23, 2003.

[16] J. F. Correcher, R. Alvarez-Valdes, and J. M. Tamarit, "New exact methods for the time-invariant berth allocation and quay crane assignment problem," *European Journal of Operational Research*, vol. 275, pp. 80–92, May 2019.

[17] N. Cheimanoff, F. Fontane, M. N. Kitri, and N. Tchernev, "Exact and heuristic methods for the integrated berth allocation and specific time-invariant quay crane assignment problems," *Computers & Operations Research*, vol. 141, p. 105695, May 2022.

[18] T. R. Lalita and G. S. R. Murthy, "Compact ILP formulations for a class of solutions to berth allocation and quay crane scheduling problems," *OPSEARCH*, vol. 59, pp. 413–439, Mar. 2022.

[19] E. Thanos, T. Toffolo, H. G. Santos, W. Vancroonenburg, and G. Vanden Berghe, "The tactical berth allocation problem with time-variant specific quay crane assignments," *Computers & Industrial Engineering*, vol. 155, p. 107168, May 2021.

[20] F. Meisel and C. Bierwirth, "A Framework for Integrated Berth Allocation and Crane Operations Planning in Seaport Container Terminals," *Transportation Science*, vol. 47, pp. 131–147, May 2013.

[21] A. Malekahmadi, M. Alinaghian, S. R. Hejazi, and M. A. Assl Saidipour, "Integrated continuous berth allocation and quay crane assignment and scheduling problem with time-dependent physical constraints in container terminals," *Computers & Industrial Engineering*, vol. 147, p. 106672, Sept. 2020.

[22] O. Abou Kasm, A. Diabat, and T. C. E. Cheng, "The integrated berth allocation, quay crane assignment and scheduling problem: mathematical formulations and a case study," *Annals of Operations Research*, vol. 291, pp. 435–461, Aug. 2020.

[23] B. Ji, M. Tang, Z. Wu, S. S. Yu, S. Zhou, and X. Fang, "Hybrid rolling-horizon optimization for berth allocation and quay crane assignment with unscheduled vessels," *Advanced Engineering Informatics*, vol. 54, p. 101733, Oct. 2022.

[24] T. Wang, X. Wang, and Q. Meng, "Joint berth allocation and quay crane assignment under different carbon taxation policies," *Transportation Research Part B: Methodological*, vol. 117, pp. 18–36, Nov. 2018.

[25] G. Gamrath, D. Anderson, K. Bestuzheva, W.-K. Chen, L. Eifler, M. Gasse, P. Gemander, A. Gleixner, L. Gottwald, K. Halbig, G. Hendel, C. Hojny, T. Koch, P. Le Bodic, S. J. Maher, F. Matter, M. Miltenberger, E. Mühmer, B. Müller, M. E. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, C. Tawfik, S. Vigerske, F. Wegscheider, D. Weninger, and J. Witzig, "The SCIP Optimization Suite 7.0," technical report, Optimization Online, 2020.

[26] S. Maher, M. Miltenberger, J. P. Pedroso, D. Rehfeldt, R. Schwarz, and F. Serrano, "PySCIPOpt: Mathematical programming in python with the SCIP optimization suite," in *Mathematical Software – ICMS 2016*, pp. 301–307, Springer International Publishing, 2016.

[27] K. Wolter, "Implementation of cutting plane separators for mixed integer programs," Master's thesis, Technischen Universität Berlin, Berlin, Germany, 2006.

[28] T. Achterberg, T. Koch, and A. Martin, “Branching rules revisited,” *Operations Research Letters*, vol. 33, pp. 42–54, Jan. 2005.

[29] S. P. Fekete, J. Schepers, and J. C. Van Der Veen, “An Exact Algorithm for Higher-Dimensional Orthogonal Packing,” *Operations Research*, vol. 55, pp. 569–587, June 2007.

[30] F. Landheer, “Product requirement document,” *Cofano Asia*, pp. 5–17, 2020.

[31] M. Iori, V. L. De Lima, S. Martello, F. K. Miyazawa, and M. Monaci, “Exact solution techniques for two-dimensional cutting and packing,” *European Journal of Operational Research*, vol. 289, pp. 399–415, Mar. 2021.

[32] V. Cacchiani, M. Iori, A. Locatelli, and S. Martello, “Knapsack problems — An overview of recent advances. Part I: Single knapsack problems,” *Computers & Operations Research*, vol. 143, p. 105692, July 2022.

[33] V. Cacchiani, M. Iori, A. Locatelli, and S. Martello, “Knapsack problems—an overview of recent advances. Part II: Multiple, multidimensional, and quadratic knapsack problems,” *Computers & Operations Research*, vol. 143, pp. Paper No. 105693, 14, 2022.

[34] C. Chen, S. Lee, and Q. Shen, “An analytical model for the container loading problem,” *European Journal of Operational Research*, vol. 80, pp. 68–76, Jan. 1995.

[35] R. Baldacci and M. A. Boschetti, “A cutting-plane approach for the two-dimensional orthogonal non-guillotine cutting problem,” *European Journal of Operational Research*, vol. 183, pp. 1136–1149, Dec. 2007.

[36] J.-F. Côté, M. Dell’Amico, and M. Iori, “Combinatorial Benders’ Cuts for the Strip Packing Problem,” *Operations Research*, vol. 62, pp. 643–661, June 2014.

[37] M. Delorme, M. Iori, and S. Martello, “Logic based Benders’ decomposition for orthogonal stock cutting problems,” *Computers & Operations Research*, vol. 78, pp. 290–298, Feb. 2017.

[38] G. Belov and G. Scheithauer, “A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting,” *European Journal of Operational Research*, vol. 171, pp. 85–106, May 2006.

[39] S. Martello, D. Pisinger, and P. Toth, “Dynamic Programming and Strong Bounds for the 0-1 Knapsack Problem,” *Management Science*, vol. 45, pp. 414–424, Mar. 1999. Publisher: INFORMS.

[40] E. Balas, “Facets of the knapsack polytope,” *Mathematical Programming*, vol. 8, pp. 146–164, Dec. 1975.

[41] P. L. Hammer, E. L. Johnson, and U. N. Peled, “Facet of regular 0–1 polytopes,” *Mathematical Programming*, vol. 8, pp. 179–206, Dec. 1975.

[42] L. A. Wolsey, “Faces for a linear inequality in 0–1 variables,” *Mathematical Programming*, vol. 8, pp. 165–178, Dec. 1975.

[43] W.-K. Chen and Y.-H. Dai, “On the complexity of sequentially lifting cover inequalities for the knapsack polytope,” *Science China. Mathematics*, vol. 64, no. 1, pp. 211–220, 2021.

[44] D. Pagurek, “Divide and conquer algorithms.”

[45] F. Meisel, *Seaside Operations Planning in Container Terminals.* Contributions to Management Science, Heidelberg: Physica-Verlag HD, 2009.

[46] L. Zhen, “Tactical berth allocation under uncertainty,” *European Journal of Operational Research*, vol. 247, pp. 928–944, Dec. 2015.

[47] J. H. Chen, D.-H. Lee, and J. X. Cao, “A combinatorial benders’ cuts algorithm for the quayside operation problem at container terminals,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 48, pp. 266–275, Jan. 2012.

[48] A. Imai, X. Sun, E. Nishimura, and S. Papadimitriou, “Berth allocation in a container port: using a continuous location space approach,” *Transportation Research Part B: Methodological*, vol. 39, pp. 199–221, Mar. 2005.

[49] L. Zhen, L. H. Lee, and E. P. Chew, “A decision model for berth allocation under uncertainty,” *European Journal of Operational Research*, vol. 212, pp. 54–68, July 2011.

[50] S. Tang, J. G. Jin, and C. Lu, “Investigation of berth allocation problem in container ports considering the variety of disruption,” *Computers & Industrial Engineering*, vol. 172, p. 108564, Oct. 2022.

[51] C. S. Tang, ed., *Supply chain analysis: a handbook on the interaction of information, system and optimization.* No. 119 in International series in operations research & management science, New York, NY: Springer, 1. ed ed., 2008.

[52] Y. Guan and R. K. Cheung, “The berth allocation problem: models and solution methods,” *OR Spectrum*, vol. 26, pp. 75–92, Jan. 2004.

[53] L. Coelho, “Linearization of the product of two variables,” 2017.

[54] Y. B. Türkoğulları, Z. C. Taşkın, N. Aras, and K. Altınel, “Optimal berth allocation and time-invariant quay crane assignment in container terminals,” *European Journal of Operational Research*, vol. 235, pp. 88–101, May 2014.

[55] Wikipedia, “Straddle carrier,” 2022.

[56] Wikipedia, “Reach stacker,” 2022.

[57] Aicrane, “Container gantry crane,” 2021.

[58] C. Junqueira, M. Quiñones, A. Azevedo, C. Rocco, and T. Ohishi, “An integrated optimization model for the multi-port stowage planning and the container relocation problems,” 2020.

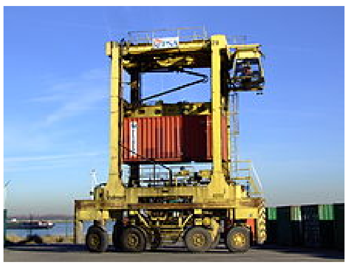
A Definitions

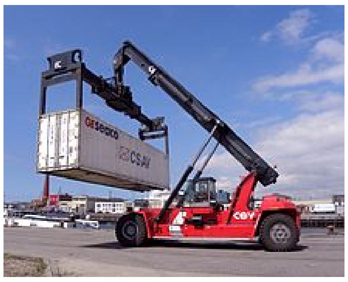
A.1 Locations

terminal	A piece of the port that includes a quay, berths, container storage, equipment and people fixed to this terminal.
sea terminal	A terminal that can serve both barges and big container ships.
inland terminal	A terminal that only serves barges; it is typically connected to a river.
quay	A platform alongside the water for loading and unloading ships.
berth	The place where a ship lies when docked. It is alongside a quay.
storage yard	The place where the containers are stored prior to be loaded on a ship or truck.
seaside	The area of the terminal that includes the berth and quay.
yardside	The area of the terminal that includes the storage yard and gate to leave port by vehicle.
hinterland	A region lying inland from the coast.
dry port	A dry port is an inland intermodal terminal directly connected by road or rail to a seaport, operating as a center for the transshipment of sea cargo to inland destinations. It decreases the need for storage space and the custom clearance directly at the seaport.

A.2 Terminal equipment

quay crane	A crane that unloads (and loads) containers from ships to (and from) transfer vehicles.
yard/gantry crane	A crane that unloads (and loads) container from port yard stacks to (and from) transfer vehicles.
draft	The depth of water drawn by a ship.
straddle carriers	A type of transfer vehicle that transports containers by 'straddling' them, thus not requiring a crane to load/unload a truck.
tugboat	A vessel that maneuvers other vessels by pushing or pulling them, with direct contact or a towline.
reefer	A refrigerated container, a reefer ship is a ship that only carries reefers. The containers have a nonstandard size and require different electrical distribution than normal
reefer connection	A transfer vehicle specifically for the transportation of reefers.
reach stacker	A transfer vehicle that can stack 2 high and 2 deep. It can move quickly over short distances.
flatbed	A transfer vehicle that consists of a flat bed on wheels. It cannot load or unload itself.
empty handler	A type of forklift that can transport empty containers and place them on stacks in the storage yard.


(A) straddle carrier [55]


(B) reach stacker [56]

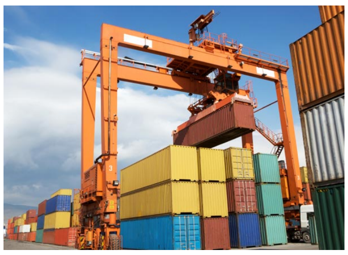

(C) yard/gantry crane [57]

FIGURE 30: Yard equipment that can place containers in the storage yard.

A.3 Ships

barge	A long flat-bottomed boat for carrying freight on canals and rivers, either under its own power or towed by a tugboat. It does not have a hold.
container ship	A cargo ship that carries all of its load in truck-size intermodal containers.
bay	The cross-sections of the ship where containers are stored. They are numbered from bow to stern.
row	The rows run the length of the ship and are numbered from the middle of the ship outwards with even numbers on the port side and odd numbers on the starboard side.
tier	The layers of containers, numbered from the bottom to up.
stack	A column of containers that are stacked on top of each other.
block	A group of containers that are stacked together such that they touch each other.
hold	The part of big container ships that is under the deck. A single container ship can have several holds. Containers in the hold have to be loaded prior to all containers on deck, and unloaded after all the containers on the deck have been removed.
rake	The overhang of a ship's bow or stern.
TEU	TEU stands for Twenty foot Equivalent Unit, and it is the standard size for containers. 40 foot long containers are also common.

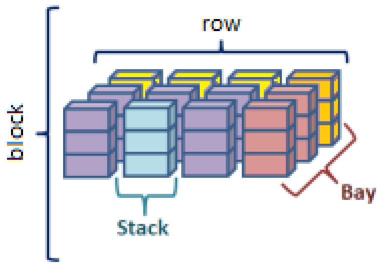


FIGURE 31: A schematic view of a bay, row, stack and block. The image was adapted from [58].

B 2D-Knapsack is NP-hard

Proposition B.1. *The two-dimensional geometric knapsack problem in NP-hard.*

Proof. Let $\mathcal{I}$ be an instance of the bin packing problem with items $i \in V$, weights c_i and bin capacity B. Let $\phi(\mathcal{I})$ be the corresponding 2D-Knapsack problem with items $i \in V$, widths $w_i = 1$, lengths $l_i = c_i$, profits $p_i = 1$, knapsack length $L = B$ and width W. Then the optimal value of the bin packing problem equals k if and only if the smallest W for which the optimal value to $\phi(I) = |V|$ is equal to k.

Suppose we have a solution to the bin packing problem with value k. Then stacking all items in bin b on top of each other at horizontal position $b - 1$ results in a feasible packing of all items for $\phi(\mathcal{I})$ with $W = k$.

Suppose we have a solution to $\phi(\mathcal{I})$ where all items are packed with knapsack width k. Since all items have a width of 1, and the length of the knapsack is precisely the bin capacity, placing all items with $h_i = b - 1$ into the same bin results in a solution to the bin packing problem with value k.

The bin packing problem is strongly NP-hard, and we have found a reduction from it to 2D-Knapsack. Therefore, 2D-Knapsack is also strongly NP-hard.

□

C Proof of Proposition 3.4

Proposition 3.4. The adjacent cover inequalities generated by a minimal cover with side item j are facet-inducing for P if and only if for any item $i \notin C$ with $w_i > 1$ there is a set $C_i \subset C$ with $|C_i| = |C| - 1$ such that

$$l_i + \sum_{k \in C_i} l_k \leq L, \tag{25}$$

and if for any $i \notin C$ with $w_i = 1$ there is a set $C_i \subset C$ with $j \notin C_i, |C_i| = |C| - 2$ such that

$$l_i + \sum_{k \in C_i} l_k \leq L, \tag{26}$$

Proof. This proof works similarly to that of Proposition 3.3, except that there are more item-position combinations to consider. Let F and H be as in Proposition 3.3. Let $\hat{y}^1, ..., \hat{y}^4$ be solutions to the LP relaxation that lie in F constructed in the following way, where e_{iv} is the unit vector with its 1-entry at (i, v):

- $\hat{y}^1$ is a solution where exactly $|C| - 1$ items item-position combinations in $P^{C,w} \cup P_j^{C,w}$ are selected, and no other items are included in the knapsack. Let the set of items in the knapsack be called $C_{\hat{y}}$
- $\hat{y}^2 := \hat{y}^1 + e_{i'v}$ is solution $\hat{y}^1$ where one additional item i' is assigned to any feasible location v in the knapsack.
- $\hat{y}^3 := \hat{y}^1 - e_{iv} + e_{i'x}$ removes some item $i \in C$ placed in the knapsack in $\hat{y}^1$ and includes the remaining $i' \in C$ not included in $\hat{y}^1$ at any position x such that $(i', x) \in P^{C,w} \cup P_j^{C,w}$.
- $\hat{y}^4 := \hat{y}^1 - e_{iv} + e_{ix}$ shifts the position of some item i in the knapsack in $\hat{y}^1$ to a different position x such that $(i, x) \in P^{C,w} \cup P_j^{C,w}$.

Since these solutions are all in F (and thus also in H), we have that $d^T \hat{y}^k = \gamma$ for $k = 1, 2, 3, 4$. Taking the difference of these equations gives us

$$0 = d^T(\hat{y}^1 - \hat{y}^2) = d^T(-e_{i'v}) = -d_{i'v} \qquad \forall i' \notin C_{\hat{y}}, \quad \forall \text{ feasible } v \tag{67}$$

$$0 = d^T(\hat{y}^1 - \hat{y}^3) = d^T(e_{iv} - e_{i'x}) = d_{iv} - d_{i'x}$$
$$\forall i \text{ s.t. } \hat{y}^1_{iv} = 1, (i', v) \in P^{C \setminus C_{\hat{y}}, w} \cup P_j^{C \setminus C_{\hat{y}}, w} \tag{68}$$

$$0 = d^T(\hat{y}^1 - \hat{y}^4) = d^T(e_{iv} - e_{ix}) = d_{iv} - d_{ix}$$
$$\forall i \text{ s.t. } \hat{y}^1_{iv} = 1, \forall x \neq v, (i, x) \in P^{C,w} \cup P_j^{C,w}. \tag{69}$$

This holds for any solution $y \in F$, so this sets $d_{iv} = 0$ for all items i and all positions v that are feasible for some solution $y \in F$. Additionally, all $d_{iv} = D$ where D is some constant for all $(i, v) \in P^{C,w} \cup P_j^{C,w}$.

We first show that this condition on the item lengths results in a facet-inducing inequality by proving that there are no item-position combinations (i, v) that are infeasible for all solutions $y \in F$ in this case. Suppose such a position does exist. Then for every solution y (and thus also $\hat{y}^k$, $k = 1, 2, 3, 4$) there is a slice that exceeds its capacity L if item i is placed at position v. W.l.o.g. this is assumed to be slice w or $w+1$, because all other slices occupy at most the capacity of w and $w+1$ for solutions of the type $\hat{y}^k$ with $k = 1, 2, 3, 4$. The proof for slice $w+1$ is symmetrical to that of w, so only the proof for w will be shown here.

Suppose, for the sake of contradiction, that $i \notin C$ holds and that $w_i > 1$. Then the subset of the $|C| - 1$ smallest items allows for the addition of item i without exceeding the capacity L by (25). However, this means that item i can be added at any position $v \in \{0, ..., W - w_i\}$ to the solution that assigns the $|C| - 1$ smallest items in the cover to positions in $P^{C,w} \cup P_j^{C,w}$ and includes no other items. Similarly, if $w_i = 1$, item i can be added at positions $\{0, ..., w, w+2, .., W-1\}$ to the solution with the smallest $|C|-2$ items excluding j in both slices and side item j in slice $w+1$ due to (26). Adding it at position $w+1$ is possible by switching the position of side item j such that it only occupies slice w. This is a contradiction, so i must be in C. This is impossible by Proposition 3.3, so we get that the facet-inducing inequality for H reduces to

$$D \sum_{(i,v) \in P^{C,w} \cup P_j^{C,w}} y_{iv} + 0 \leq \gamma. \tag{70}$$

This is equal to (17) up to a factor for scaling (set $\frac{\gamma}{D} = |C| - 1$), which means $F = H$, so F is a facet.

We now show that this is the only condition under which adjacent cover inequalities are facet-inducing. Suppose for the sake of contradiction that F is a facet and there is an item a such that either (25) or (26) does not hold for a. Let z be a position such that item a occupies both slices if $w_a > 1$ and slice w if $w_a = 1$. If $y_{az} = 1$, at most $|C| - 2$ item-positions in $P^{C,w} \cup P_j^{C,w}$ can be selected, since any combination including $|C| - 1$ items exceeds L. Then the inequality

$$\sum_{(i,v) \in P^{C,w}} y_{iv} + \sum_{(j,v) \in P_j^{C,w}} y_{jv} + y_{az} \leq |C| - 1 \tag{71}$$

is valid for P. Since this inequality implies (17) (which is the inequality that induces F), F cannot be a facet. This contradicts our assumption, so F is only a facet if (25) and (26) hold. □