Bergische Universität Wuppertal
Fakultät für Wirtschaftswissenschaft
Schumpeter School of Business and Economics

Interference aware scheduling of gantry cranes in container yards

Dissertation zur Erlangung des akademischen Grades
Doktor der Wirtschaftswissenschaft
vorgelegt von
Lennart Zey (M.Sc.)
Wuppertal, Februar 2019

Betreut von: Prof. Dr. Dirk Briskorn / Prof. Dr. Stefan Bock

Die Dissertation kann wie folgt zitiert werden:

urn:nbn:de:hbz:468-20190529-101201-3
[http://nbn-resolving.de/urn/resolver.pl?urn=urn%3Anbn%3Ade%3Ahbz%3A468-20190529-101201-3]

Acknowledgements

After a little more than 4 years, this dissertation comes to an end and would probably haven taken a little longer without the influence of several people. First of all and most importantly, I want to thank Prof. Dr. Dirk Briskorn for his continuous support and advice and the door that stood open for me when a detour or deadlock in research seemed to appear on the horizon (pun intended). On top of that, I appreciate the perfect mixture of guidance, liberties and encouragement that made my time in Wuppertal really enjoyable, letting me not once doubt my decision to step on to this path. Furthermore, I want to thank Prof. Dr. Stefan Bock for being my second adviser and for reviewing this dissertation.

Michael, Lena, Marcel and Bart: thank you for creating a helpful and enjoyable working atmosphere at the chair. I would particularly like to thank Michael, for being a passable roommate at several conferences and workshops and for being an appreciative sparring partner in terms of research, but more importantly in terms of humor.

I want to thank my beloved wife Vera for supporting me and accepting the changes in our life plan that come with writing a dissertation. Thank you for taking all duties off my back whenever one of those last-minute errors in the coding had to be fixed.

Lastly, I deeply want to thank my parents, for encouraging me to try out new things while supporting me if something would go wrong.

Contents

Abbreviations

B&B	branch-and-bound approach
MIP	mixed integer programming
RMG	rail mounted gantry crane
TRCIRP	triple-crossover-crane interference resolving problem
TRCSP	triple-crossover-crane scheduling problem
TCSPH	twin-crane scheduling problem in presence of a dedicated handshake area

Tables

Figures

List of Symbols

(c, c', θ, θ') Branch, implying that crane c' conducts operation θ' before c conducts θ

(c, k) The kth request of crane $c \in C$

1 Notation of the crossover-crane

2 Notation of the seaside-twin-crane

3 Notation of the landside-twin-crane

$\hat{d}^b_{j(i)}$ Bay position of the drop off request of job $j(i)$

$\hat{d}^r_{j(i)}$ Row position of the drop off request of job $j(i)$

$\hat{d}_{j(i)}$ Position as $(\hat{d}^b_{j(i)}, \hat{d}^r_{j(i)})$ of the drop off request of job $j(i)$

$\hat{o}^b_{j(i)}$ Bay position of the pick up request of job $j(i)$

$\hat{o}^r_{j(i)}$ Row position of the pick up request of job $j(i)$

$\hat{o}_{j(i)}$ Position as $(\hat{o}^b_{j(i)}, \hat{o}^r_{j(i)})$ of the pick up request of job $j(i)$

ν Start point of a detour

σ_c Routing of crane c

$\varepsilon_{o,c}$ Plane on obstacle o indicating the prioritization of crane c

A_c Jobs assigned to crane c

b^h	Bay position of the handshake area
b_j^O	The bay position of the drop off request of job j
b_j^U	The bay position of the pick up request of job j
b_c^0	Starting bay of crane $c \in C$ at the beginning of the planning horizon
$b_{c,k}$	Bay where the container of request (c, k) gets picked up or dropped off
C	Set of Cranes
C_r^h	Capacity of stacking position r
C_{max}	Makespan
d_j^O	Duration necessary to conduct the drop off request of job j
d_j^U	Duration necessary to conduct the pick up request of job j
d_i	Destination position of container i
$d_{c,k}$	Duration necessary to conduct request (c, k)
e_j^O	Earliest period when the drop off request of job j may be completed
e_j^U	Earliest period when the pick up request of job j may be completed
$ec_{c,k}$	Earliest possible completion time of request (c, k)
G_c	Number of requests of crane $c \in C$
H_2	Positions in the handshake area exclusively used by crane 2 for dropping off containers
H_3	Positions in the handshake area exclusively used by crane 3 for dropping off containers
I^i	Set of inbound containers
I^o	Set of outbound containers

$I^{i,l}$	Set of inbound containers to be handled by the landside-crane
$I^{i,s}$	Set of inbound containers exclusively handled by the seaside-crane
$I^{o,l}$	Set of outbound containers to be handled by the landside-crane
$I^{o,s}$	Set of outbound containers exclusively handled by the seaside-crane
J	Set of container transport jobs
$j(i)$	Transport job related to container i
$j^1(i)$	Job describing the storage of i in the handshake area
$j^2(i)$	Job describing the retrieval of i from the handshake area
J^R	Set of jobs, having row positions in the handshake area minimizing laden travel necessary for retrieving jobs
J^S	Set of jobs, having row positions in the handshake area minimizing laden travel necessary for storing jobs
J_c	Set of container transport jobs of crane c
$l(\sigma_c)$	Duration necessary to carry out σ_c
$l(n_c)$	Length of the non-delay routing based on n_c
l_j^O	Latest period when the drop off request of job j may be completed
l_j^U	Latest period when the pick up request of job j may be completed
$lc_{c,k}$	Latest possible completion time of request (c, k)
N_c	Jobs related to containers, handled by crane c only
n_c	The job sequence of crane $c \in C$
o_c^0	Initial position of the cranes
o_i	Origin position of container i

p	Duration necessary for lifting or releasing a container
$p_{c,t}$	Bay position of crane $c \in C$ at the end of period t
Q	Set of bays
R_c^k	Jobs in R_c for which the row position in the handshake area is determined
R_c^u	Jobs in R_c for which the row position in the handshake area is not determined
R_c	Jobs implying a retrieval from the handshake area by crane c
S_c^k	Jobs in S_c for which the row position in the handshake area is determined
S_c^u	Jobs in S_c for which the row position in the handshake area is not determined
s^c	Safety distance to be kept between a twin-crane's centre and the crossover-crane's centre
s^t	Safety distance to be kept between the twin-crane's centres
S_c	Jobs of c implying a storage in the handshake area
T	Planning horizon
W_c	Total workload of crane c based on its assigned jobs
w_j	Workload of job j
$x_{t,j,c}^O$	Binary variable: 1 if conducting the drop off request of job j, performed by crane c is completed in period t; 0 otherwise
$x_{t,j,c}^U$	Binary variable: 1 if conducting the pick up request of job j, performed by crane c is completed in period t; 0 otherwise
$x_{c,k,t}$	Binary variable: 1, if crane $c \in C$ finishes conducting the kth request at the end of period t; 0, otherwise

$y_{j,c}$ Binary variable: 1 if a job j is assigned to crane c; 0, otherwise

$z^O_{j,c}$ Binary variable: 1 if crane $c \in \{2,3\}$ is positioned in a larger bay than crane 1 while 1 conducts the drop off request of job j; 0 otherwise

$z^U_{j,c}$ Binary variable: 1 if crane $c \in \{2,3\}$ is positioned in a larger bay than crane 1 while 1 conducts the pick up request of job j; 0 otherwise

$z_{c,k}$ Binary variable: 1, if crane $c \in \{2,3\}$ stays in a larger bay than crane 1 while the kth request of crane 1 is conducted; 0 otherwise

Chapter 1

Introduction

1.1 Motivation

The importance of maritime container transport in global supply chains is undisputed. Over the last decades it constantly increased, from 5984 million tons loaded globally in 2000 to 10702 million tons in 2017 (see UNCTAD [41]). With growing transport volumes and likewise growing capacity of container ships, increases the need for efficient processes in container terminals. Here, the containers arrive on vessels, or by land, on trucks or trains, in order to be transshipped to their final destination. A schematic terminal layout is depicted in Figure 1.1.
Keeping the berthing times of ships short is a key goal of both, terminal operators and shipping companies, resulting from its major influence on profitability. In order to allow a high container throughput, all involved terminal subsystems, such as the quay-cranes needed to unloaded ships, the storage yard, as well as inner terminal transport devices such as trucks or automated guided vehicles, are in high need of elaborate planning procedures.

This dissertation has its focus on the container storage yard. Here, containers are intermediately stored after they arrive at the terminal whenever the designated means of onward transport is not yet available. Hence, the storage

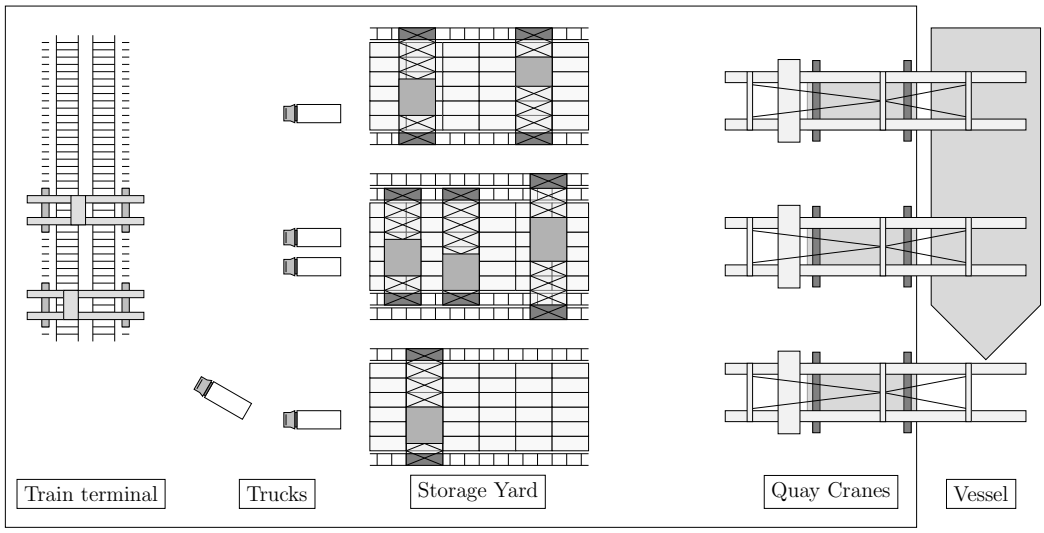


Figure 1.1: Schematic layout of a container terminal

yard acts as a key interface between sea- and land-transport and is involved in most terminal operations. Typically, it consists of multiple container storage blocks, in which containers are grouped and temporarily stored. Usually these blocks are designed according to one of two layouts. Either they are placed parallel to the wharf, following the so called Asian Layout, or they are placed perpendicular (i.e. European Layout), see Carlo et al. [15]. This dissertation focuses on the latter.

After being transported to the designated storage block by an inner terminal transport device, there exist multiple types of vehicles that carry out all operations associated with storing or retrieving containers. Among those are Rubber Tired Gantries, being cranes that serve multiple blocks due to their mobility, as well as Lift Trucks or Straddle Carriers. This dissertation however examines Rail Mounted Gantry Cranes (RMGs). These cranes move on pairs of rails along both sides of a container storage block and are hence assigned permanently to it. The key advantage of RMGs is, that they are able to operate (nearly) fully automated and that they can be employed on large blocks, while being able to handle heavy containers (Libbey [34]). We find them employed in all major container terminals such as the port of Antwerp, Hamburg and Rotterdam (see Kemme [29]).

In- and outbound containers handled by RMGs, enter or leave a block then through one of two transfer access points, referred to as the land- or seaside, where they are (un-)loaded (from) onto terminal transport devices. There exist multiple types of RMG setups, usually being one of three settings. First, only a single RMG is employed on the block, as depicted on the bottom block in Figure 1.1. Second, there is a pair of RMGs working jointly, as shown on the top block in Figure 1.1. Either, it consists of two cranes of equal height and width, referred to as twin-cranes, that cannot cross positions. Or, on the other hand, it consists of a larger crossover-crane, moving on a separate pair of rails along the block, that can cross a smaller RMG. Finally, there can be three cranes employed, consisting of a pair of twin-cranes and a crossover-crane (see the middle block in Figure 1.1), referred to as the triple-crossover-crane setting.

There exist multiple areas of optimization with the storage yard and its blocks being involved, such as (among others) design decisions, container-to-block assignment, storage position allocation, as well as crane scheduling for a given block. The research objective of this dissertation is on crane scheduling, assuming that all previously named decisions are already taken. Specifically, the scheduling of triple-crossover cranes is examined, being a crane setup that promises a high container throughput (see Klaws et al. [30]). Furthermore, the topic of twin-crane scheduling during seaside workload peaks is covered, where the workload between cranes is shared by handing over containers from one crane to the other. For both areas, literature focusing on holistic scheduling approaches is relatively scarce. In addition, interference between cranes, being a key driver of productivity, is oftentimes resolved by simple rules only and key constraints such as precedence relations and stacking capacities are only covered to some extend. Hence, throughout this dissertation several interference aware scheduling approaches are developed that aim at contributing to the respective fields of research.

1.2 Outline

This dissertation focuses on two dominant topics, being the scheduling of triple-crossover-cranes as well as the scheduling of cooperating twin-cranes in presence of a dedicated container handover area. After providing an overview of literature on crane scheduling in Section 1.3, a holistic scheduling approach for triple-crossover-cranes is presented in Chapter 2. Here, a pair of twin-cranes of equal height and width and a crossover-crane that can cross the other cranes, serve a container storage block. The chapter begins by introducing a sophisticated routing approach and continues by detailing a holistic scheduling framework. Within the framework, job sequences for cranes are constructed under the objective of providing minimum makespan, interference free, schedules. For this purpose two branch-and-bound approaches are developed and compared by means of a computational study.

In Chapter 3 a pair of twin-cranes that serve a single storage block, is considered. It is assumed that the crane located at the seaside has a higher workload, e.g. during seaside workload peaks when a vessel is to be(un)loaded. Hence, all containers arrive and leave the block at the seaside. In order to share the workload between the cranes, a handshake area is employed within the yard, where containers are handed over from one crane to the other. The position of the handshake area then affects the workload of the cranes and eventually the minimum makespan necessary to carry out all transport jobs. Three branch-and-bound approaches that are able to obtain minimum makespan schedules for a given handshake area, are developed and are later on benchmarked against heuristic approaches in a computational study. The chapter concludes with providing insight on how the position of the handshake area as well as capacities affect the makespan and how the approaches perform within a rolling horizon setting.

Finally, Chapter 4 concludes the dissertation and gives an outlook on future research.

1.3 Literature

The literature overview presented in this section is based on the findings in Briskorn and Zey [6, 7] and Zey et al. [43]. All quotes are taken from [7].

The topic of terminal optimization is intensively studied and many overviews regarding the topic exist. "Stahlbock and Voß [39] and Steenken et al. [40] provide general overviews about operations in sea port container terminals and give insights into the related optimization problems and solution approaches. Carlo et al. [13] focus on optimization problems located at the seaside of a terminal, whereas Carlo et al. [15] give an overview about the optimization problems that arise from transport operations between sea, yard and landside. Further Carlo et al. [16] provide an overview about layout decisions and optimization problems at storage yards concerning multiple crane settings. A comparison between different types of storage crane settings can be found in Vis [42]. When the cranes are of equal size and move on the same pair of rails the scheduling problem shares similarities to the scheduling of factory or warehousing cranes in automated storage and retrieval systems (AS/RS). Boysen et al. [4] provide a general classification scheme for crane scheduling with interference in multiple logistics areas and give an overview about that topic.

In the literature multiple approaches exist that consider the scheduling of a single crane that serves a yard block, see e.g. Daganzo [19] or Ng and Mak [36], for yard crane scheduling with release dates of jobs or Gharehgozli et al. [22] for an exact approach when aiming at finding a sequence of fulfillment with minimal total travel time. Further approaches can be found in Boysen and Stephan [3]. However, naturally, interference is not an issue when scheduling a single crane, such that single-crane approaches have to be extended when considering multiple cranes.

Among the work that focuses on two cranes working on a block few regard interference in an exact manner: In Li et al. [32] a crane scheduling model for two yard cranes moving on a common track with a discrete time horizon is proposed that solves a problem similar to the one in Ng [35] but is less

resource demanding and therefore obtains solutions faster. The approach assigns container jobs to cranes and determines a sequence in which they are processed that regards interference. In Li et al. [33] the authors enhance the model proposed in [32] such that the time horizon is modeled continuously which reduces the number of variables, and improves the run time. All three studies provide optimal solutions at the cost of large run times even for small instances. Thus, in [32] and [33] a rolling horizon perspective is applied as well as heuristics that provide good solutions. In Choe et al. [17] a local search based approach is developed, while Choe et al. [18] provides a genetic algorithm, both addressing the scheduling of twin-cranes under consideration of non-crossing constraints. Gharehgozli et al. [23] provide a heuristic approach tackling the problem of scheduling twin-cranes when storage and retrieval operations are given under the objective of minimizing the makespan. Briskorn and Angeloudis [5] focus on the routing decision for given assignment of jobs to cranes and sequences. They develop a graphical model representing the problem setting and a strongly polynomial DP algorithm in order to provide conflict-free schedules that minimize the overall makespan. Nossack et al. [37] describe a two-stage decomposition approach for a crossover-crane-setting in which one larger crane can cross the smaller one. The problem involves all three parts of the decision, that is assignment, sequences, and routing have to be decided. Interference is regarded in an exact manner employing the DP approach by [5]. The approach solves small to medium size instances in reasonable time up to optimality and provides good solutions for larger instances."

For the previously presented approaches it is assumed that a container, once lifted, is completely transported by a single crane. However, as a measure to share the workload, container handover can be allowed. Then, cranes store containers in intermediate positions where they get afterwards picked up by the other crane and the remaining transport is conducted. For such a setting, Briskorn et al. [8] develop an heuristic approach, aiming at makespan minimization during seaside workload peaks. Jaehn and Kress [27] extends the approach and regards landside-related workload as well whereas Kress et al. [31] introduces an exact solution procedure for this optimization task.

In order to simplify planning, container handover can be restricted to a dedicated handshake area within the storage yard. For this premise Carlo and Martínez-Acevedo [14] propose interference avoidance rules for cranes operating in the handshake area. The effect of container handover in any bay is compared to the handover in a dedicated area by means of simulation in Gharehgozli et al. [24].

"Klaws et al. [30] analyze the benefits of a triple-crossover-crane setting as compared to a setting with two cranes moving on the same pair of rails by means of a simulation study. Here storage location, re-stacking strategies and conflict avoidance between cranes are decided using heuristic approaches. The authors conclude that that a triple-crossover-crane setting is indeed more productive than a dual crane setting even when providing the routing in a non-optimal manner. Besides a general overview about the design and operations of automated container storage systems, Kemme [29] proposes a model formulation for the scheduling and routing of containers in a triple-crossover-setting under the objective of minimizing the sum of the cranes' waiting times. The increase in computational complexity caused by simultaneously tackling the routing and scheduling results in large computing times even for a small number of jobs to be handled. Heitmann [26] proposes a mixed integer model, again aiming at minimizing the total waiting time of triple-crossover-cranes. Here, release dates, due dates and precedence relations between jobs are considered. A two-stage decomposition approach is presented, consisting of a first step tackling the simultaneous assignment and sequencing of jobs while in the second step a subsequent interference free routing is determined. For the same number and type of cranes, Dorndorf and Schneider [20] develop an algorithm that solves the assignment and sequencing. Interference is then resolved in a branch and bound algorithm under the objective of minimizing the sum of final completion times of crane schedules. The provided routing is afterwards evaluated by a multi-criteria objective function." For given job sequences, Briskorn and Zey [6] develop a routing approach under a makespan minimization objective, that is as well presented in this dissertation.

Chapter 2

Scheduling of Triple-Crossover-Cranes

The content presented in this section is as well depicted in Briskorn and Zey [6] and Briskorn and Zey [7].

When faced with increasing container transport volumes, increasing the container handling capacity of a block, by employing more cranes seems to be a viable measure in order to sustain a high productivity. Hence, in this chapter we pick up on the so called triple-crossover-crane setting, that was e.g implemented at Container Terminal Burchardkai (see [20]), in order to significantly increase the container throughput. In this setting, three RMGs, being a larger crossover-crane and two twin-cranes of identical size, that move on pairs of rails alongside the block, are considered. The smaller twin-cranes move on the same pair of rails and are of equal size, and hence, cannot pass each other. The larger crossover-crane however, moves on a separate pair of rails such that the other cranes can pass below it as long as its spreader is completely lifted. Figure 2.1 illustrates the setup, depicting a single container block embedded in the terminal from above while Figure 2.2 provides a detailed look on the block only.

On the right hand side in Figure 2.1, we have quay cranes (un)loading ships that arrive at the terminal. Container transport in between the storage block

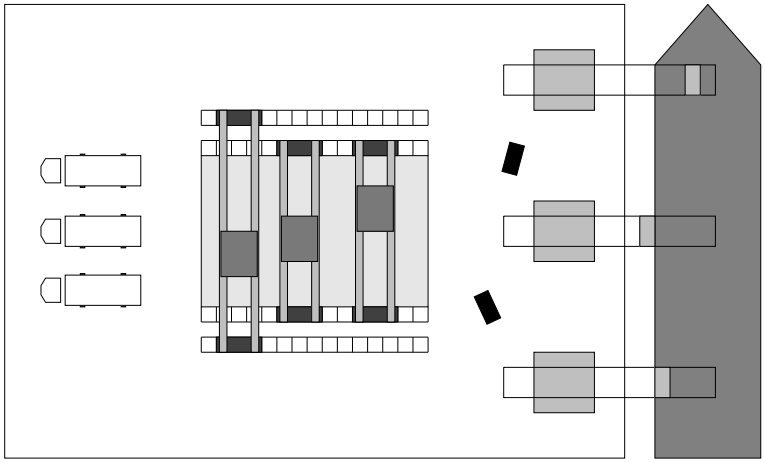


Figure 2.1: A triple-crossover-crane setting deployed on a storage yard

and the quay cranes is conducted by inner terminal transport devices such as Automated Guided Vehicles whereas containers reach or leave the block on trucks on the left hand landside.

Even though increasing the container handling capacity by employing more cranes seems beneficial, it comes at the cost of potentially more conflicts between crane activities. As a result, incorporating a sophisticated interference avoidance strategy is a key measure to achieve the desired productivity. Hence, a fast approach for conflict resolution aiming at minimal makespan, when job sequences of cranes have been decided, is developed in Section 2.1.

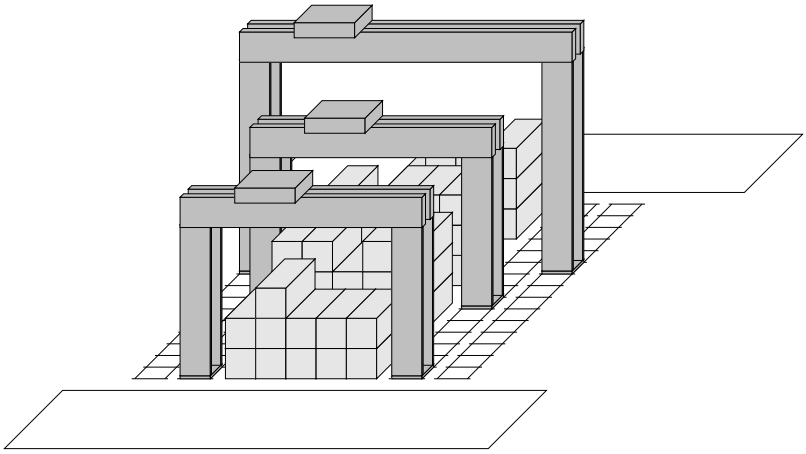


Figure 2.2: Example of a triple-crossover-crane setting

In Section 2.2 an algorithm tackling

- the problem of determining an assignment of transport jobs to cranes,
- of constructing job processing sequences, based on the assignments, and
- of determining interference free routings, by employing the approach from Section 2.1
- under the objective of minimizing the overall makespan,

is developed. Hereby it is assumed that a set of jobs to be planned is given and pick up and drop off positions and durations of these jobs are predetermined by an other component of the terminal control system.

2.1 The Triple-Crossover-Crane interference resolving problem

The content presented in this section is as well published in Briskorn and Zey [6] and all quotes are taken from this very article.

"This section is structured as follows. In Section 2.1.1 we give a formal definition of the problem, with a single container type only and a safety distance of one container length between cranes followed by a mixed integer programming (MIP) model representing it. Section 2.1.2 presents a graphical model capturing the problem. In Section 2.1.3 we give insight on how the model can be extended to account for different container types and arbitrary safety distances between cranes. Although we can guarantee to find a feasible schedule, the corresponding approach is a heuristic since the network does not cover all dominant routing strategies. Implementation details are outlined in Section 2.1.4. We show that we obtain near-optimal results by means of a computational study in Section 2.1.5. In particular, we compare our results

to the solutions achieved by a standard solver using the MIP model and a greedy heuristic.

2.1.1 Problem definition and model formulation

The problem setting in this section considers triple-crossover-cranes, having to avoid interference while working jointly on a container yard block and we refer to it as the *triple-crossover-crane interference resolving problem* (TR-CIRP).

The yard block consists of a set of bays $Q := \{0, 1, \ldots, B + 1\}$ whereas 0 and $B + 1$ are handover areas for containers that approach or leave the block by truck or AGV. We denote the set of cranes as $C := \{1, 2, 3\}$ with 1 representing the crossover-crane. It can cross the twin-cranes since it moves on different rails on the yard block and has a larger height and width and thus cannot collide with them as long as its spreader is not lowered. The twin-cranes are of equal height and width and move on the same pair of rails along the block and, therefore, cannot cross each other or move to the same position. They are denoted by 2 and 3 whereas 2 is always positioned in a bay with a smaller number than 3.

Throughout this section it is assumed that the cranes, the containers and the bays have the same length. Such an assumption is not a crucial restriction. We see in [29], that the current generation of gantry cranes roughly has the length of a 40-foot container and hence we consider 40-foot containers only. However, we will give insight on how to handle containers of different sizes in Section 2.1.3.2.

The time horizon is assumed to be continuous and is segmented into periods where period $t \in \mathbb{N} \setminus \{0\}$ covers the interval between $[t - 1, t]$. We are given a sequence of *transport jobs* for each crane with one job corresponding to the lift of a container in its origin position and the release of that container in its destination position. Each crane processes the jobs in its sequence in the respective order. Since with regard to interference of cranes there is

no difference between picking a container up or dropping one off, we split each job into two separate *requests*, corresponding to lifting or releasing a container. This allows us to obtain a sequence of requests for each crane, based on the sequence of transport jobs. We denote the number of requests of crane c, $c \in C$, by G_c, the kth, $k = 1, \ldots, G_c$, request of crane c, $c \in C$, by (c, k), and the bay where c conducts request (c, k) by $b_{c,k}$. In order to conduct request (c, k), c has to be positioned in $b_{c,k}$ for a duration of $d_{c,k}$ which reflects the time necessary for lowering the spreader, grabbing a container or releasing it, and bringing the spreader up again. Note that this duration may depend on the request since the duration of moving the spreader and adjusting it may depend on the very location of the request and its surrounding containers.

Each crane $c \in C$ is positioned in a starting bay b_c^0 at the beginning of the first period and has to conduct its assigned requests in the predefined *sequence* n_c. In between conducting two requests (c, k) and $(c, k+1)$ crane c has to move from $b_{c,k}$ to $b_{c,k+1}$. Each crane can move with a speed of 1 bay per period.

A routing σ_c of crane c can be described by detailing the activity of c in each period t. We consider three types of activities:

- moving from bay b to bay $b-1$ or $b+1$ with $0 \leq b-1$ and $b+1 \leq B+1$,
- conducting request (c, k) in bay $b_{c,k}$ and
- waiting in bay b.

The first two activities are necessary for a crane to fulfill requests in $1, \ldots, G_c$ and we synonymously refer to them as operation throughout this section.

A crane is positioned in a bay $b \in Q$ for a whole period if a crane waits or conducts a request. However, its position gradually changes with a constant rate (reflecting its speed of 1 bay per period) when it moves from b to $b+1$ $(b-1)$ in period t'. Then, at point of time $t \in [t'-1, t']$, the crane's position is $b + (t - t' + 1)$ or $b - (t - t' + 1)$ respectively.

A routing of crane c is considered feasible, with regard to movements and the sequence of requests if

- c processes $(c, 1)$ to (c, G_c) in the given order,
- c is located in bay $b_{c,k}$ while conducting (c, k) for $d_{c,k}$ consecutive periods, and
- if c is located in bay b at the end of period t, then the next operation starts from or takes place in b.

The *non-delay routing* based on n_c, of a crane c is the routing where crane c conducts all of its requests as early as possible, starting from bay b_c^0 assuming that no interferences with other cranes occur. The length of a non-delay routing is denoted by $l(n_c)$.

A schedule $(\sigma_1, \sigma_2, \sigma_3)$ is an assignment of a routing that is feasible with regard to movements and the sequence of requests to each crane. Let $T_{(\sigma_1,\sigma_2,\sigma_3)}$ be the makespan of such a schedule which is the maximum number of periods among these routings with regard to $(\sigma_1, \sigma_2, \sigma_3)$. If the routing for a crane c has less than $T_{(\sigma_1,\sigma_2,\sigma_3)}$ periods, then we append waiting activities in the crane's last position such that each routing has $T_{(\sigma_1,\sigma_2,\sigma_3)}$ periods.

Conflicts between the cranes must be resolved while conducting requests on the container block. A schedule, therefore, is regarded as feasible if cranes do not interfere, that is if

- while conducting request $(1, k)$ in a time interval $[t, t + d_{1,k}]$ no twin-crane is located in a position b such that $b_{1,k} - 1 < b < b_{1,k} + 1$ at any point of time $t' \in [t, t + d_{1,k}]$ and
- at each point of time $t' \in [0, T_\sigma]$ cranes 2 and 3 are located in bays b and b', respectively, with $b \leq b' - 1$. Hence, it is assumed that crane 2 (3) does not operate in bay $B + 1(0)$.

Our problem then is to find a feasible schedule with minimum makespan. Briskorn and Angeloudis [5] show that the corresponding problem for two

cranes (either cranes 1 and 2 or cranes 2 and 3) can be solved in (strongly) polynomial time. These results build on a rough analogy to the problem to schedule two jobs (corresponding to cranes) in a job shop (with machines corresponding to bays) aiming at minimum makespan. This problem has been shown to be solvable in polynomial time in, e. g. Brucker [12]. We are not able to settle the computational complexity of TRCIRP but believe it to be NP-hard since the problem to schedule three jobs (corresponding to three cranes in TRCIRP) in a job shop aiming at minimum makespan has been proven to be NP-hard in Sotskov and Shakhlevich [38].

In the following we formulate a MIP model that represents the TRCIRP. An upper bound on the minimum makespan can be derived easily by considering a schedule where the three cranes operate strictly one after another. That is, crane 1 conducts its requests first and cranes 2 and 3 do not have anything to do but to move out of its way if necessary. Afterwards, crane 2 conducts its requests and, finally, crane 3 does. We set the time horizon T in our MIP model to this upper bound. Furthermore, we can derive a lower bound as $\max\{l(n_c) \mid c = 1, 2, 3\}$.

Note that for a given lower bound and upper bound on the makespan we can determine an earliest completion time $ec_{c,k}$ as well as a latest completion time $lc_{c,k}$ for each request (c, k). We employ binary variable $x_{c,k,t}$ signaling whether ($x_{c,k,t} = 1$) or not ($x_{c,k,t} = 0$) c completes (c, k) at the end of period t. We describe the bay in which c is positioned at the end of period t with $p_{c,t}$. Since these variables are time based it is expected that the pre-determined upper- and lower bounds have an effect on the performance of the MIP. We will give further insight into this circumstance in Section 2.1.5. Finally, we use binary variable $z_{c,k}$, $c \in \{2, 3\}, k = 1, 2, \ldots, G_1$, which has value 0 if crane c is positioned in a bay $b \leq b_{1,k} - 1$ while crane 1 conducts $(1, k)$ and has value 1 if crane c is positioned in a bay $b \geq b_{1,k} + 1$ while crane 1 conducts $(1, k)$.

$$\text{Min } C_{max} \tag{2.1}$$

$$C_{max} \geq \sum_{t=ec_{c,G_c}}^{lc_{c,G_c}} x_{c,G_c,t} \cdot t \qquad \forall c \in C \tag{2.2}$$

$$\sum_{t=ec_{c,k}}^{lc_{c,k}} x_{c,k,t} = 1 \qquad \forall c \in C, k = 1, \ldots, G_c \tag{2.3}$$

$$\sum_{t=ec_{c,k}}^{lc_{c,k}} x_{c,k,t} \cdot t \leq \sum_{t=ec_{c,k+1}}^{lc_{c,k+1}} (t - d_{c,k+1}) \cdot x_{c,k+1,t} \quad \forall c \in C, k = 1, \ldots, G_c - 1 \tag{2.4}$$

$$(B+1) \cdot \left(1 - \sum_{t'=\max\{t,ec_{c,k}\}}^{min\{t+d_{c,k},lc_{c,k}\}} x_{c,k,t'}\right) + b_{c,k} \cdot \left(\sum_{t'=\max\{t,ec_{c,k}\}}^{min\{t+d_{c,k},lc_{c,k}\}} x_{c,k,t'}\right) \geq p_{c,t} \tag{2.5}$$

$$\forall c \in C, k = 1, \ldots, G_c, t = ec_{c,k} - d_{c,k}, \ldots, lc_{c,k}$$

$$b_{c,k} \cdot \left(\sum_{t'=\max\{t,ec_{c,k}\}}^{min\{t+d_{c,k},lc_{c,k}\}} x_{c,k,t'}\right) \leq p_{c,t} \tag{2.6}$$

$$\forall c \in C, k = 1, \ldots, G_c, t = ec_{c,k} - d_{c,k}, \ldots, lc_{c,k}$$

$$(B+2) \cdot \left(2 - \left(z_{c,k} + \sum_{t'=\max\{t,ec_{1,k}\}}^{min\{t+d_{1,k},lc_{1,k}\}} x_{1,k,t'}\right)\right) \geq b_{1,k} + 1 - p_{c,t} \tag{2.7}$$

$$\forall c \in \{2,3\}, k = 1, \ldots, G_1, t = ec_{1,k} - d_{1,k}, \ldots, lc_{1,k}$$

$$(B+2)\cdot\left(2-\left((1-z_{c,k})+\sum_{t'=\max\{t,ec_{1,k}\}}^{min\{t+d_{1,k},lc_{1,k}\}} x_{1,k,t'}\right)\right)\geq p_{c,t}-(b_{1,k}-1) \tag{2.8}$$

$$\forall c\in\{2,3\}, k=1,\ldots,G_1, t=ec_{1,k}-d_{1,k},\ldots,lc_{1,k}$$

$$p_{c,0}=b_c^0 \qquad \forall c\in C \tag{2.9}$$

$$p_{c,t}-p_{c,t-1}\leq 1 \qquad \forall c\in C, t=1,\ldots,T \tag{2.10}$$

$$p_{c,t-1}-p_{c,t}\leq 1 \qquad \forall c\in C, t=1,\ldots,T \tag{2.11}$$

$$p_{2,t}\leq p_{3,t}-1 \qquad \forall t=1,\ldots,T \tag{2.12}$$

$$x_{c,k,t}\in\{0,1\} \qquad \forall c\in C, k=1,\ldots,G_c, t=ec_{c,k},\ldots,lc_{c,k} \tag{2.13}$$

$$z_{c,k}\in\{0,1\} \qquad \forall c\in\{2,3\}, k=1,\ldots,G_1 \tag{2.14}$$

Objective function (2.1) reflects the goal to minimize the makespan. Constraint (2.2) bounds the makespan from below whereas (2.3) enforces that each request is conducted exactly once. Constraint (2.4) ensures that the jobs are executed in the given sequence. The cranes being located in the bay they are conducting a request in is enforced by (2.5) and (2.6). Note that constraints (2.5) and (2.6) restrict position $p_{c,t}$ of c in period t to values in $[0, B+1]$ if no request is conducted in t. If, however, crane c conducts its kth request in t, then position $p_{c,t}$ is fixed to $b_{c,k}$. Constraints (2.7) and (2.8) ensure that the twin-cranes are not positioned in a bay in which the crossover-crane is currently conducting a request in. No significant restrictions are imposed on positions of cranes 2 and 3 by constraints (2.8) and

(2.7) if no request is conducted by crane 1 in t. If, however, crane 1 conducts its kth request in t, then position $p_{2,t}$ is bounded from below by $b_{1,k} + 1$ or bounded from above by $b_{1,k} - 1$ depending on whether $z_{2,k} = 1$ or $z_{2,k} = 0$. The positions of crane 3 are restricted analogously.

The starting position of the cranes is given in (2.9). Restriction (2.12) prevents the twin-cranes from passing each other or moving to the same bay. Finally, (2.10) and (2.11) restrict the speed of each crane to one bay per period. The domains of the decision variables are defined in (2.13) and (2.14).

2.1.2 Graphical model

We develop a graphical model picking up the foundation laid by Briskorn and Angeloudis [5]. We detail the representation of our problem by this model in Section 2.1.2.1. The model allows us to find a schedule by finding a path through the three-dimensional model. Since there typically is an infinite number of paths we narrow the paths under consideration down in the following subsections, and in doing so, potentially miss an optimal path.

The intuition behind this reduction is to let cranes conduct their respective request sequences simultaneously as often and as long as possible. This naturally supports short makespans. Only in order to avoid interferences one or two cranes may wait or make a detour which delays completion of request sequences. The graphical model representation allows us to identify promising situations where waiting or detours may actually be beneficial.

In Section 2.1.2.2 we determine a network of paths representing schedules where the only allowed option to avoid interference is to let a crane wait in order to prioritize another crane in case of a conflict. That is, we do not take detours into account here. In Section 2.1.2.3 we extend this network to account for detours. In both cases, we can then determine a schedule by finding a path through the network.

2.1.2.1 Graphical representation

Before developing the model, let us quickly revisit the foundations laid by Briskorn and Angeloudis [5]. In their work, the authors develop an exact routing approach for two cranes. The approach yields feasible (conflict-free) schedules with a minimum makespan. The cranes under consideration are either two twin-cranes or a crossover-crane and a smaller crane that work in the same storage block. For two non-delay routings of cranes c and c', a graphical model is presented, encompassing a rectangular area with a width equal to $l(n_c)$ and a height equal to $l(n_{c'})$. Each point $(t_c, t_{c'})$ describes the progress of both cranes with regard to their non-delay routings, that is crane c and c' completed the first t_c and $t_{c'}$ periods of their non-delay routings. Naturally, such points may violate interference constraints, e. g. if it means that both cranes conduct requests at the same time in the same bay. Infeasible points are clustered to *obstacles*. A path from the lower left to the upper right corner of the rectangle corresponds to a schedule and if such a path does not cut through any obstacle it is feasible. The authors construct a network of paths such that at least one path in the network represents a feasible schedule with minimum makespan. Finding this path within the network can be accomplished by finding a shortest path in a directed acyclic graph.

Throughout this section, the graphical representation of our problem is embedded in a cube with width, height and depth each equal to the length of one of the non-delay routings of the cranes such that the axis corresponding to crane c starts at 0 and ends at $l(n_c)$. Each point in the cube, referred to as a tuple $t = (t_1, t_2, t_3)$, again represents the progress of each crane with regard to its non-delay routing. That is, crane c has processed the first t_c periods of its non-delay routing. Note that the position of c can be derived from t_c and the non-delay routing of n_c. Furthermore, we can see whether crane c is currently conducting a request or not.

A schedule corresponds to a path from $(0, 0, 0)$ to $(l(n_1), l(n_2), l(n_3))$. Such a path can be partitioned into segments. A *segment* is a part of the path

where the direction of the path does not change. Each segment represents a subset of cranes processing with regard to their non-delay routings for a certain duration. Depending on the very subset of cranes processing we have an orientation of the segment within the cube. The orientation can be seen as a tuple with binary entries (p_1, p_2, p_3) where $p_c = 1$ if and only if crane c is processing. Hence, we have potentially seven different types of segments, where at least one crane is processing. Note that orientation $(0, 0, 0)$ does not lead anywhere. If a segment connects points (t_1, t_2, t_3) and (t'_1, t'_2, t'_3) we must have either $t_c = t'_c$ or $t_c + L = t'_c$ for each crane c and an arbitrary non-negative value L. This segment, then represents a partial schedule starting at states corresponding to (t_1, t_2, t_3) and reaching at states corresponding to (t'_1, t'_2, t'_3). During this partial schedule covering a timespan of L periods crane c processes if and only if $t'_c = t_c + L$ and it does so with full speed. We then say the segment has *length* L.

We can easily transform, possibly by adding waiting times, a schedule in which cranes run in any speed into another schedule of the same makespan in which all cranes run in full speed during moving. Assume that a crane conducts a request in a bay with less than full speed, in order to avoid a conflict at a later part of its schedule. Then, it can as well conduct requests with full speed and wait in the same bay afterwards, in order to compensate for the earlier completion of the request. Now assume that a crane moves towards a bay with less than full speed, in order to avoid a conflict in that bay. It can as well move with full speed and wait next to the bay until the conflict is resolved. Finally, assume that a crane c moves with half speed because another crane c' moves with less than full speed and in doing so blocks c. Crane c' may be blocked in the same way by the third crane but there is at least one crane which is not blocked by another crane (unless we have a deadlock). The crane not being blocked can process at full speed as argued above enabling the other cranes to process with full speed as well.

A path represents a full schedule and its length is given as the total length of its segments.

A point may correspond to states violating the interference constraints pre-

sented in Section 2.1.1. In this case we say that the point is infeasible. Note that whether a point is infeasible or not depends only on the states of pairs of cranes since interference constraints involve pairs of cranes only. Thus we can directly project the *clusters* of infeasible points developed by Briskorn and Angeloudis [5] for the two-crane settings to our cube.

1. For each request of crane 1 in bay b and each time crane 2 (3) enters b we have a *cluster*. The cluster encompasses all points where crane 1 conducts the request and 2 (3) enters b, potentially conducts a request in b, and leaves b. Whether or not point $t = (t_1, t_2, t_3)$ is contained in such a cluster, hence, depends on t_1 and t_2 (t_3) but not on t_3 (t_2).

2. For each request of crane 2 in bay b and each request of crane 3 in bay b' with $b' \leq b$ we have a cluster. The cluster encompasses all points where crane 2 (3) approaches b (b') and has passed bay b' (b) already, conducts the request in b (b'), and leaves bay b (b') and has not passed bay b' (b) again yet. Whether or not point $t = (t_1, t_2, t_3)$ is contained in such a cluster, hence, depends on t_2 and t_3 but not on t_1.

Note that these clusters may be overlapping. Briskorn and Angeloudis [5] show that for two-crane settings these clusters accurately identify infeasible points. From this result and the independence from the third coordinate as stated above we conclude the following corollary.

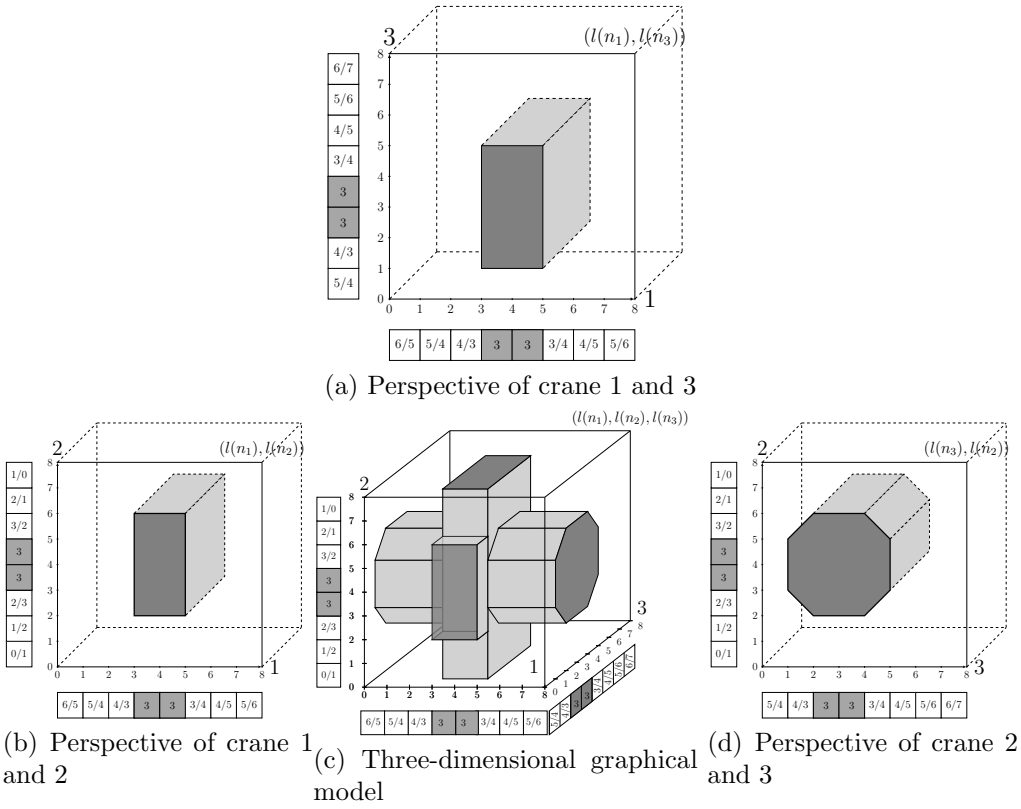
Corollary 1. A point is infeasible if and only if it is contained in one or more such clusters.

We say a path is feasible if it does not cut through any cluster. Therefore, we refer to these clusters as *obstacles* in the following. We have two types of obstacles, then, corresponding to the two types of clusters decribed above. A feasible path represents a schedule where at no time the cranes' states are in conflict. The problem to find a schedule with minimum makespan, then, is equivalent to finding a feasible path with *minimum length*.

Figure 2.3 gives an example of both types of obstacles in a three dimensional model. The cranes each have to conduct one request in bay 3. Crane 1 is positioned in $b_1^0 = 6$ at the beginning of the planning horizon and has to return to that bay after conducting the request. Crane 2 starts and ends its schedule in bay 0, while crane 3 is positioned in 5 and has its parking position in bay 7. The cranes cannot conduct their requests simultaneously because in doing so they would operate in the same bay at the same time. The non-delay routings are annotated at the respective axes. Gray squares represent conducting requests in the bay noted in the square while white squares represent moves, e.g. the first square on the axis of crane 1 describes a move from bay 6 to bay 5. Considering cranes 1 and 2 (and 1 and 3), we can restrict ourselves to the two-dimensional perspective in Figure 2.3b and 2.3a where for each case the obstacles run across the complete third axis which is only implied. The dark-gray area on the axis of 1 and 2 (1 and 3) covers the infeasible states, namely when both cranes conduct a request in bay 3 at the same time and when 2 (3) enters or leaves bay 3 while 1 conducts the request. We find these dark-gray areas as well in Figure 2.3c. Considering cranes 2 and 3, we can restrict ourselves to the two-dimensional perspective in Figure 2.3d. Analogous to the obstacle between 1 and 2 (1 and 3), states where both cranes conduct a request in bay 3 simultaneously are covered and highlighted in dark-gray. In addition the obstacle covers states where 2 and 3 have crossed positions. In Figure 2.3c the corresponding obstacles in the cube are depicted. We observe that there are points where only two cranes are in conflict, e. g. cranes 1 and 3 in $(4, 1, 4)$ and cranes 2 and 3 in $(2, 4, 4)$, but there are also points where all three cranes are in conflict with each other, e. g. $(4, 4, 4)$.

In Figure 2.4 we see a feasible path through the cube developed in Figure 2.3. The respective position of the cranes in each period implied by the path is depicted in Figure 2.5. We denote the bay position on the y-axis while the periods take the x-axis. Additionally we marked the states referring to the graphical model.

In the depicted path the cranes execute their non-delay routings simultane-



(a) Perspective of crane 1 and 3

(b) Perspective of crane 1 and 2

(c) Three-dimensional graphical model

(d) Perspective of crane 2 and 3

Figure 2.3: Obstacles from a two- and three-dimensional perspective

ously for one period corresponding to a segment from $(0, 0, 0)$ to $(1, 1, 1)$ of length 1. Crane 3 waits in bay 4 such that it does not interfere with the next request of crane 2. Cranes 1 and 2 process with regard to their non-delay routings for 2 periods corresponding to a segment from $(1, 1, 1)$ to $(3, 3, 1)$ of length 2. At the end of period 3, crane 1 waits in bay 3 with its spreader lifted to let 2 conduct the request first. Accordingly, crane 2 conducts its request and leaves the bay while both other cranes wait corresponding to a segment from $(3, 3, 1)$ to $(3, 6, 1)$, having length 3. At the beginning of period 7, while crane 3 still waits crane 1 starts to conduct its request. Crane 1 operates for an additional period and crane 2 completes its non-delay routing at the end of period 8, corresponding to a segment from $(3, 6, 1)$ and $(5, 8, 1)$ with a length of 2. From here on, cranes 1 and 3 can proceed without interruption. This corresponds to, first, a segment from $(5, 8, 1)$ to $(8, 8, 4)$ where crane

1 completes its non-delay routing and, second, a segment from $(8, 8, 4)$ to $(8, 8, 8)$ where crane 3 completes its own with length 3 and 4 respectively. The length of the path then equals 15.

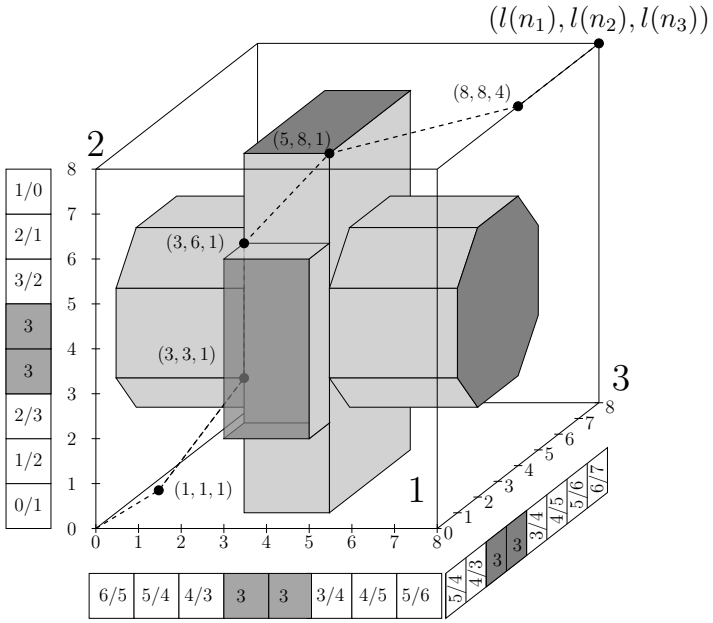


Figure 2.4: A feasible path through the model

2.1.2.2 Schedules without detours

Given the graphical representation of a problem instance there will usually be an infinite number of paths from $(0, 0, 0)$ to $(l(n_1), l(n_2), l(n_3))$. As mentioned in Section 2.1.2.1 we restrict ourselves to paths consisting of segments where cranes conduct requests either with full speed or not at all.

Still, restricting to such paths may leave us with an infinite number of paths. In the section at hand, therefore, we further reduce the set of paths to be considered. We do so by constructing a network with a finite set of segments and, therefore, a finite set of paths from $(0, 0, 0)$ to $(l(n_1), l(n_2), l(n_3))$.

In order to provide an intuition we first outline the basic idea of our procedure. Of course, segments with orientation $(1, 1, 1)$ are most promising when aiming at a small makespan since all three cranes process their non-delay

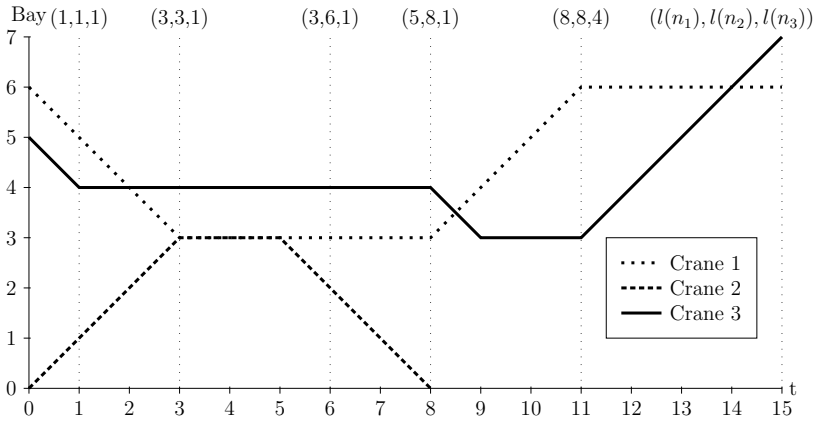


Figure 2.5: Crane positions over time for the depicted path in Figure 2.4

routings in parallel. Hence, we deviate from this orientation only if (potentially) necessary. The same holds for orientations with two operating cranes from which we deviate by letting a second crane wait only if (potentially) necessary. Hence, we aim at using segments with orientation $(1, 1, 1)$ as much as possible and make use of segments with less than three cranes processing their non-delay routings only in order to circumnavigate obstacles. This rule translates into the crane environment as the following property of an optimum solution.

Property 1. We let cranes wait only if (potentially) necessary in order to prevent interferences of cranes.

Furthermore, if crane c waits for another crane c' it does so with respect to one of two types of conflicting operations: First, requests r of c and r' of c' are in conflict with each other and we decide that c' can conduct r' before c conducts r. Second, a moving operation of twin-crane c' is in conflict with a request $r = (1, k)$ of $c = 1$ and we decide that c' can move through the bay $b_{1,k}$ before crane 1 conducts $(1, k)$ (or the other way round). In both cases, crane c waits for crane c' with respect to an obstacle as defined in Section 2.1.2.1.

Property 2. If crane c waits for crane c' with respect to an obstacle, then it proceeds along its non-delay routing before waiting (up to the point where

processing its operation related to the obstacle is in conflict with c' processing its operation first).

This property restricts the set of points in the cube where we have to allow segments with orientations with less than 3 cranes processing. It is easy to see that an optimum schedule with this property exists. Consider an optimum schedule without this property and the first point of time t a crane starts waiting earlier than it would be necessary in order to avoid interference. We choose an arbitrary crane c among those starting to wait in t earlier than necessary and modify the schedule by letting this crane process its non-delay routing for one more period. This, obviously, yields a feasible schedule and cannot increase the makespan. Furthermore, either the first point of time where a crane waits earlier than necessary has been increased or the number of cranes doing so in t has been decreased. Hence, by repeatedly applying this step we can achieve an optimum schedule having the desired property.

Note that the two properties above are in line with properties developed in Briskorn and Angeloudis [5] and transferred to the case with three cranes here, only.

In order to construct the network we apply two basic procedures, namely *branches* and *restarts*, whereas a restart refers to restarting the crane from previously waiting. A branch (c, c', θ, θ') determines a certain crane c' to process operation θ' before another crane c processes operation θ. Obviously, branches need to be applied only for pairs of operations θ and θ' giving rise to an obstacle since other pairs can be processed in parallel. Branches are applied to a certain point on segment s reducing the set of cranes currently processing their non-delay routings by a single crane. It is implied that crane c stops processing its non-delay routing and stays in the bay corresponding to the point on s. Obviously such a crane c waiting in this bay must not be in conflict with θ'. Restarts are applied at the end points of some segments and start a new segment with more cranes processing their non-delay routings from there.

We find restarts in state $(3, 6, 1)$ or $(5, 8, 1)$ in the path depicted in Figure

2.4 where crane 1 (or 3) can continue operating after previously waiting. A branch is applied to the segment starting at $(0, 0, 0)$ in $(1, 1, 1)$, where crane 3 begins to wait in order to not interfere with crane 2. The two conflicting operations inducing the branch are the moves to bay 3 of cranes 2 and 3. Consequently we start a *partial network*. A second branch in the same partial network is applied at $(3, 3, 1)$ where crane 1 begins to wait as well for 2. Here the conflicting operations of 1 and 2 are the ones implying conducting the requests in bay 3.

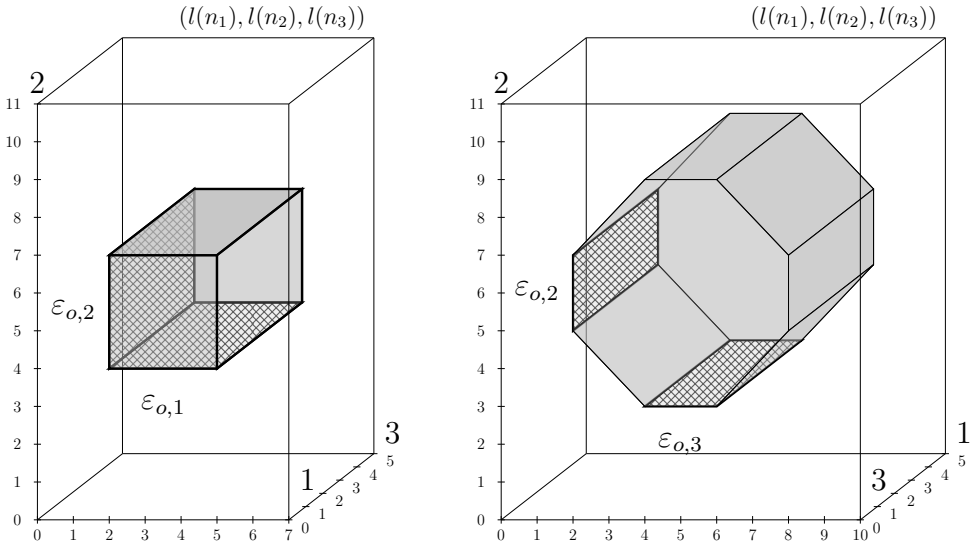
Intuitively speaking, a partial network begins when a crane c starts to wait and it ends, when the conflict causing the waiting has been resolved. The partial network then consists of all the segments that describe different prioritizations of the other cranes c' and c'' while c is waiting.

In the proposed example the partial network first induced by letting crane 3 wait does contain two segments. However this is not necessarily the case for every partial network. A partial network with a waiting crane c can consist of multiple segments implying different prioritization for pairs of conflicting operations of the other cranes c' and c''.

We claim that given an arbitrary schedule branches and restarts (plus the processing along a segment) are sufficient to describe the schedule. Note that we need to allow segments to have length zero between two such procedures since we have to account for schedules where two cranes start waiting at the same point of time.

When generating the network we apply branches and restarts with regard to the following scheme. A branch (c, c', θ, θ') is applied to a segment s in order to induce a *partial network* where crane c is waiting for crane c' with respect to operation θ and θ'. When, eventually, a path through this partial network is chosen, then θ' is conducted before θ. This partial network starts with a segment s' with an orientation differing from the one of s only in c' processing as on s and c waiting with regard to s'. This partial network ends at one edge of the obstacle (corresponding to θ and θ') which indicates that c' has finished operation θ' and the conflict is resolved. Note that while crane c is

not making any progress with respect to its non-delay routing the other two cranes may encounter conflicts which have to be resolved. There might be several options how to resolve these conflicts resulting in a (partial) network of segments rather than a single segment.



(a) Obstacle between crane 1 and 2 (b) Obstacle between crane 2 and 3

Figure 2.6: Planes potentially touched by segments resulting from branching

Figure 2.6 depicts both types of obstacles (corresponding to cranes 1 and 2 on the left side and to cranes 2 and 3 on the right side). Each obstacle has two crosshatched *planes*. If a partial network exists where the other crane (involved in the perspective) waits for crane c' to finish its operation corresponding to obstacle o first, this partial network runs along the plane labeled $\varepsilon_{o,c'}$. Note that the position of the partial network with respect to crane c coincides with the one of $\epsilon_{o,c'}$ due to Property 2.

A segment of the partial network ends

- at the last point where it intersects with $\varepsilon_{o,c'}$,
- at the point where it encounters another obstacle or
- the outer boundaries of the cube.

Note that the point where a segment reaches the upper edge of $\varepsilon_{o,2}$ or the right edge of $\varepsilon_{o,1}$ and $\varepsilon_{o,3}$ of an obstacle corresponding to θ and θ' is the state from which on c can continue processing its non-delay routing (after prioritizing c'). In such a state potential restarts are applied and partial networks can end. Note, furthermore, that since further branches may be applied to s' (and the resulting partial network) there may be more than one such segment in the partial network induced by branch (c, c', θ, θ').

If c can wait in the same position for more than one operation of another crane c', then we apply a branch only for the earliest such operation of c' since we can easily apply yet another branch for letting c wait for a later operation of c' after applying a restart to end points in the first partial network.

It is obvious for segments with orientation $(1, 1, 1)$ which type of branches can be applied. For such a segment each crane may wait for each other crane such that six different branches can be applied. Note that there are only three different orientations of segment s', namely $(0, 1, 1)$, $(1, 0, 1)$, and $(1, 1, 0)$. However, depending on whether orientation $(0, 1, 1)$ results from crane 1 waiting for crane 2 or crane 3 different types of branches can be applied in this partial network. Accordingly, we refine our notation to $(0, \underline{1}, 1)$ and $(0, 1, \underline{1})$ meaning that crane 1 is waiting for crane 2 and 3, respectively. Second, for a segment with only one crane operating we cannot apply branches since in the resulting partial network no crane would be processing.

What then remains to show is which branches can be applied to segments where only two cranes are processing. For such segments we, again, potentially have six different types of branches to apply. However, we will argue that we need to consider only three of them. Consider such a segment s where c waits for c' and c'' is processing its non-delay routing in parallel. We do not need to consider a branch letting c wait for c' or c'' since this does not alter the orientation of the current segment s. So, we can just as well follow s to its end point and – if appropriate – apply the branch here after applying a restart. Furthermore, we do not need to consider a branch letting c' wait for c. Recall that on s crane c is waiting for c' to complete an operation θ'. Applying such a branch to s, then, means that c' starts waiting for c to

complete an operation θ before c' actually completes θ'. Then, there is no reason for c to wait in the first place and c completing θ could have been prioritized over c completing θ' immediately.

To conclude, we apply three types of branches to s where c waits for c' and c'' is processing its non-delay routing:

- letting c' wait for c'',
- letting c'' wait for c', or
- letting c'' wait for c.

Following the proposed example and e.g. a segment with orientation $(0, \underline{1}, 1)$, we could create segments $(0, 0, \underline{1})$, $(0, \underline{1}, 0)$ or $(\underline{0}, 1, 0)$ but omit from creating $(\underline{0}, 0, 1)$. Note that partial networks as collections of segments are potentially *overlapping*. More specifically, we have two types of overlaps. First, two partial networks may be *nested*. That is, within a partial network a second one is initiated by a branch and it has a unique end point within the first partial network. This is necessarily the case if the first partial network reflects c waiting for c' and the second partial network reflects c' to wait for c''. Second, two partial networks may be partially overlapping. That is, within a partial network a second one is initiated by a branch and all end points of the second partial network end outside the first partial network. This is necessarily the case if the first partial network reflects c to wait for c' and the second partial network reflects c'' to wait for c. If the second partial network results from a branch letting c'' wait for c' then both partial networks may be partially overlapping or nested. Since the part of the second partial network that overlaps with the first one consists of a single path only there is either a unique end point of the second partial network within the first one or all end points of the second partial network lie outside the first one.

We find an example of such a partial network in the path depicted in Figure 2.4. Here, we start the first partial network by letting crane 3 wait for crane 2. Hence the orientation of the segment starting at state $(1, 1, 1)$ in the cube

is $(1, \underline{1}, 0)$. While the conflict between cranes 2 and 3 is not yet resolved (crane 3 is still waiting) we decide that crane 1 waits, as well, in order to let crane 2 conduct the request first. Hence, we start the second partial network within the first and the segment has orientation $(0, \underline{1}, 0)$. At this point the first partial network consists of two segments, while the second network consists of a single segment only. Since both networks address the same operation of crane 2 (conducting a request in bay 3) the endpoint of both networks is connected to 2 having left bay 3 at state $(3, 6, 1)$ and, thus, unique. We apply a restart afterwards such that we have a segment with orientation $(1, 1, 1)$. Immediately afterwards, we decide that crane 3 waits for 1. Hence the segment describing the restarts has length 0. We create a new segment and consequently a partial network, with orientation $(\underline{1}, 1, 0)$. Note that this is in line with the described branching strategy.

In the setting in Figure 2.4 there is also a nested partial network to be constructed (not depicted in Figure 2.4) letting crane 3 wait for 2 and letting crane 2 wait for crane 1. The partial network reflecting crane 2 waiting for crane 1 starts within the other partial network (at state $(2, 2, 1)$ in the cube) and, necessarily, has to be completed first before crane 2 can process its non-delay routing further." The respective crane positions over time are depicted in Figure 2.7.

"A further partially overlapping network is to be constructed for crane 3 waiting for crane 2, and crane 1 waiting for crane 3 after entering bay 3. Here, the start point of the second partial network lies inside the first one at $(3, 3, 1)$ but its endpoints lie outside $((3, 8, 5))$, since crane 1 keeps on waiting for crane 3 after crane 2 has finished its request." For this network the crane positions over time are depicted in Figure 2.8.

"It remains to detail restarts. Restarts are applied to end points of a partial network that lie on the outer edge of planes of the obstacles giving rise to the partial network. At each such end point a new segment is started. When there is no second partial network starting in the first one and having its end points outside the first one we start segments with orientation $(1, 1, 1)$. If there is such a second partial network, then the new segments are obviously

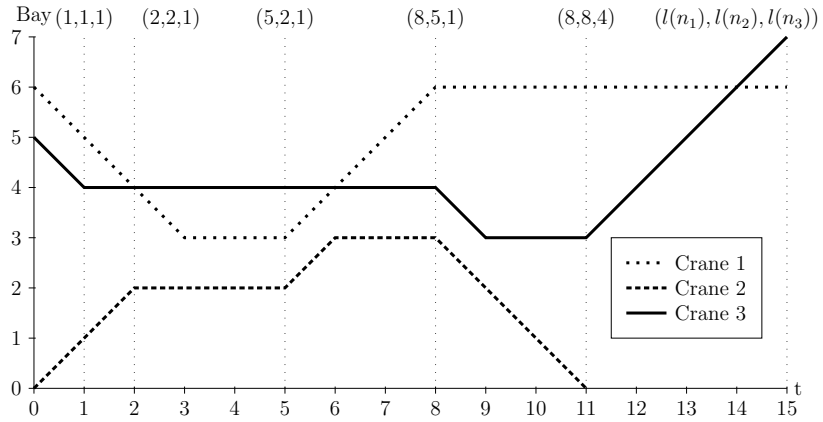


Figure 2.7: Crane positions over time in the nested partial network

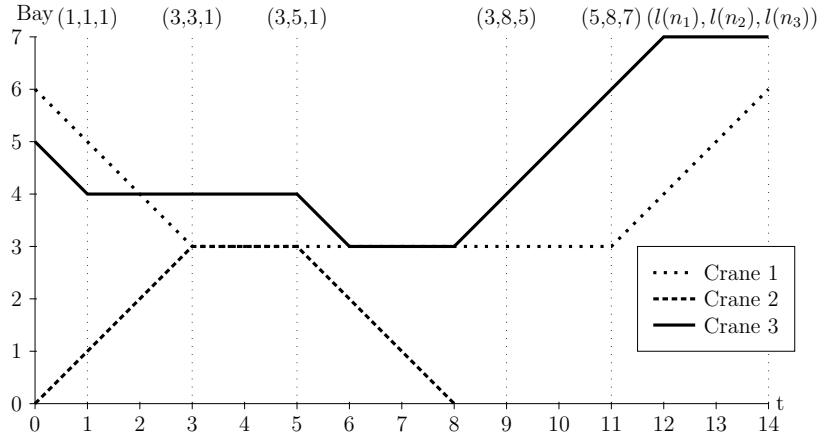


Figure 2.8: Crane positions over time in the partially overlapping network

part of it and, therefore, reflect only two cranes processing their non-delay routings with regard to the branch implying the second partial network.

A procedure constructing the full network now can be specified easily. It maintains a list of points where restarts have to be conducted together with the orientation of the segment to be started. Whenever a segment is started its end point is determined and all branches necessary (as described above) are applied yielding new points to be inserted in the list. This list is sorted in lexicographically non-decreasing order of the points' coordinates. This sorting ensures that each point is considered only once for restarts.

It is obvious that the proposed branching strategy does not necessarily lead to an overall optimal solution and, further, does not even guarantee feasible routings. The latter is the case when a deadlock situation occurs. Deadlocks may occur in even very small instances. Consider an instance where crane 1 does not have any request to conduct and cranes 2 and 3 have a single transport job to carry a container from position 3 to 4 and from position 2 to 1, respectively. If cranes 2 and 3 have initial positions 2 and 3, then there does not exist a feasible schedule without detours. We explain how detours are incorporated in the overall approach in the next section.

2.1.2.3 Detours

Up to now we assumed that cranes only deviate from their non-delay routings by waiting for another crane. Hence, we ignored the opportunity to let a crane deviate from the bay sequence resulting from the non-delay routing in order to give way to another crane. It has been shown in Briskorn and Angeloudis [5], however, that allowing detours may lead to better schedules or are even necessary for obtaining feasible solutions at all. Consequently, we consider this opportunity in the section at hand.

Note that there is no need for crane 1 to deviate from the bay sequence in its non-delay routing since it suffices to have its spreader up in order not to interfere with the other cranes. Therefore, we treat two cases only, namely the case where a twin-crane gives way to crane 1 in Section 2.1.2.3.1 and the case where one twin-crane gives way to the other in Section 2.1.2.3.2.

2.1.2.3.1 Detours of twin-cranes prioritizing crane 1 In this section we consider detours starting from a point in the network constructed as described in Section 2.1.2.2 only. In order to simplify notation we restrict ourselves to crane 2 giving way for crane 1. Crane 3 giving way to crane 1 can be handled analogously. A detour of crane 2, then, can be beneficial in a state where crane 1 cannot process its non-delay routing with crane 2 doing the same if we stick to the bay sequences. The detour of crane 2, then, allows

crane 1 to pick up or drop off a container in a bay b with crane 2 processing its non-delay routing before.

Let us give two intuitive reasons on when detours of crane 2 prioritizing crane 1 can be beneficial, before we detail the specific cases. First, crane 2 is located in the bay of the next request of crane 1 in order to avoid interference with crane 3. Either it cannot leave this bay following its non-delay routing since it is blocked by crane 3, or it entered the bay (following its non-delay routing) because waiting in a different bay would have blocked crane 3. This reason applies to the first part of case 1 and the second part of case 3 below. Second, crane 2 may have to conduct a certain series of requests in conflict with operations in the non-delay routing of crane 1. Without allowing detours this series can be started only after crane 1 has processed these operations, or on the other hand, must completely be conducted before crane 1 proceeds. A detour of crane 2 allows then both, to start the series and intermediately give priority to crane 1. This reason applies to the second part of case 1, case 2 and the first part of case 3 below.

Case 1. The next operation of crane 2 is to wait in b for crane 3 or to wait after finishing its non-delay routing.

Note that a conflict with crane 1 can occur only if crane 2 is not waiting for crane 1. Thus, (i) crane 2 waits in b for crane 3 to operate in $b+1$ and prevents crane 1 from conducting a request in b or (ii) it finished its non-delay routing.

Case 2. Both the previous operation and the next operation of crane 2 involve conducting requests in b but both operations belong to different requests.

Whenever the previous operation and the next operation belong to the same request, and hence a request takes multiple periods to be conducted a detour is not possible since we do not allow preemption of requests. Case 2, then, corresponds to a situation where crane 2 conducts two requests consecutively in b, i.e. dropping off a container and picking up another in the same bay afterwards, and gives way to crane 1 in between.

The next case considers settings where the previous operation is moving into b and the next operation involves conducting a request. Note that we do not need a detour if crane 2 can simply wait before moving into b. Hence, we consider detours only in situations where either crane 1 (case 3a) or crane 3 (case 3b) forces crane 2 to move into bay b.

Case 3. a. Both cranes 1 and 2 conduct requests in $b-1$ $(b+1)$ and b consecutively and crane 1 has conducted its request in $b-1$ $(b+1)$ later than crane 2 has.

b. Cranes 2 and 3 have to conduct requests in $b+1$ and b, respectively, next, both cranes approach their bays from a larger bay, and crane 1 has its next request in b.

Case 3a corresponds to the (only) detours between crossover-cranes considered in Briskorn and Angeloudis [5]. Case 3b formalizes the only case where crane 3 can force crane 2 to enter bay b where both, crane 1 and 2, have their next requests.

What follows is an explanation of how detours of crane 2 in order to prioritize crane 1 are conducted. Crane 2 leaves bay b and returns to b immediately after crane 1 has conducted a request in b. Note that this is not necessarily the next request of crane 1 in b. In any case, this determines the progress along the non-delay routings of cranes 1 and 2 at the end of a detour. It remains to detail the routing during such a detour which describes the progress along the non-delay routings of crane 3 at the end of a detour.

The strategy is to let crane 1 process its non-delay routing up to completely conducting the request in b while ensuring that crane 3 does not prevent crane 2 from returning to b immediately after the request of crane 1 in b is conducted. Following this strategy, there will be as little interference as possible between cranes 2 and 3 during the detour. Note that, unless $b = 0$ cranes 2 and 3 will be on different sides of crane 1 and if $b = 0$ crane 2 stays in bay $b + 1 = 1$. Implicitly, this strategy determines how much progress crane 3 can gain during the detour such that it can at most move to $b - 1$ $(b + 2)$.

Such a detour is represented in our network by a segment that reflects no progress of crane 2 with regard to its non-delay routing and cranes 1 and 3 processing their non-delay routings following the strategy described above. Note that such a segment cuts through the obstacle corresponding to the conflict of cranes 1 and 2. The segment ends in a state where crane 2 has completed the evasive move.

2.1.2.3.2 Detours of one twin-crane prioritizing the other In this section, again, we consider detours starting from a point in the network constructed as in Section 2.1.2.2 only. In order to simplify notation we restrict ourselves to crane 2 giving way for crane 3. Similar to Section 2.1.2.3.1, we consider detours of crane 2 only if crane 3 cannot conduct its next request with crane 2 doing the same if we stick to the bay sequences. The detour of crane 2, then, allows crane 3 to pickup or drop off a container without crane 2 processing its non-delay routing. In contrast to Section 2.1.2.3.1, there are instances where detours are necessary to find a feasible schedule at all. To see this, consider cranes 2 and 3 to start from initial bays $b_2^0 = 2$ and $b_3^0 = 3$ and to have a single request in bays 4 and 1, respectively, only.

As in Section 2.1.2.3.1 we limit the states from which a detour starts to a number of cases, analogously following the two underlying reasons when to start a detour from the previous section. First, crane 2 is positioned in a bay in order to avoid interference with crane 1 and crane 3 has to pass that bay. Specifically, crane 2 moved to a bay by following its non-delay routing, because waiting earlier would have blocked crane 1. In the second reason, again, crane 2 has to conduct a series of requests, being in conflict with one (or more) request of crane 3. By allowing a detour we do not restrict ourselves to schedules where either crane 2 finishes the complete series, while 3 is held back, or 2 does not start the series before 3 has conducted the interfering requests. This reason is applied in cases 4 and 5 below.

Case 4. Crane 2 has conducted a request in bay b, has to conduct a request in bay b' next, and crane 3 has to conduct a request in bay b'', $b'' \leq \min\{b, b'\}$, next.

Note that this case is in line with the detours in a twin-crane setting considered in Briskorn and Angeloudis [5] and has been shown to be potentially beneficial. By analogy, the next case covers conflicts of the initial position and the final position of crane 2 with regard to its non-delay routing.

Case 5. a. Crane 2 has to conduct a request in bay b' as first request, and crane 3 has to conduct a request in bay b'', $b'' \leq \min\{b_2^0, b'\}$, next.

b. Crane 2 has to conduct a request in bay b' as its last request, and crane 3 has to conduct a request in bay b'', $b'' \leq b'$, next.

In case 5a crane 2 starts the detour right from b_2^0 before approaching its first request in b'. A detour as described in case 5b is of any meaning only if crane 2 conducts its requests before crane 3 does and its parking position b' would otherwise prevent crane 3 from conducting a request in bay $b'' \leq b'$. While the cases above can be derived easily from the respective non-delay routings in the following case we consider a case where a detour of crane 2 prioritizing crane 3 can be beneficial only in combination with a detour of crane 2 prioritizing crane 1.

Case 6. Both cranes 1 and 2 conduct requests in $b-1$ and b consecutively and crane 1 has conducted its request in $b-1$ later than crane 2 has, and crane 3 has its next request in b.

Note that in Case 6 a detour is relevant not because of the non-delay routings of cranes 2 and 3. Without crane 1 forcing crane 2 to bay b in the first place, crane 2 could simply wait in $b-1$ for crane 3 to conduct its request in b. However, after evading from $b-1$ to b (in favor of crane 1) crane 2 can only conduct its request in b before crane 3 does or start a detour for crane 3 (in order to let crane 3 conduct its request in b first).

In what follows we describe how detours are conducted. A detour in Cases 4 and 5 starts from bay $b^d = \min\{b, b'\}$ and $b^d = \min\{b_2^0, b'\}$, respectively. This decision can be motivated quite intuitively since it simply requires crane 2 not to move towards crane 3 immediately before the detour. In Case 6 the

starting bay of the detour is naturally given as $b^d = b$. The detour ends when crane 2 returns to bay b^d. Note that this may happen whenever crane 3 is located in a larger bay.

While in Section 2.1.2.3.1 we can apply a simple strategy in order to separate cranes 2 and 3 as much as possible (and, therefore, prevent interference) it is different with detours prioritizing twin-cranes since we cannot predict and/or prevent interferences with the third crane (Crane 1) easily. Consequently, we derive the cranes' routings as a path through a cubic sub-model following essentially the same ideas as for the network construction in Section 2.1.2.2. In the following we detail the construction of the cubic model and the obstacles and emphasize some particularities. The path, then, is found just as described in Section 2.1.2.2 and 2.1.2.3.1. Note that this means that there may be a detour of crane 3 prioritizing crane 1 included in a detour of crane 2 prioritizing crane 3.

For a given start point $\nu = (t_1^\nu / t_2^\nu / t_3^\nu)$ of the detour and each maximal interval $[t_3^s, t_3^e]$ in the non-delay routing of crane 3 where it is located in a bay larger b^d we construct a three-dimensional model with the same semantics as in 2.1.2.1. Interval $[t_3^s, t_3^e]$ narrows the progress of crane 3 with regard to its non-delay routing during the detour of crane 2 down. That is, all operations on interval $[t_3^\nu, t_3^s]$ are conducted by crane 3 during the detour of crane 2 but no operation in $[t_3^e, l(n_3)]$. The operations on $[t_3^s, t_3^e]$ are not in conflict with t_2^ν."

A simple example is given as follows. Assume that in t_2^ν, crane 2 is positioned in bay $b^d = 5$, whereas in t_3^ν, crane 3 is positioned in bay 8, and has three requests $(3,k), (3,k+1)$ and $(3,k+2)$ with $d_{3,k} = d_{3,k+1} = d_{3,k+2} = 1$ and $b_{3,k} = 4, b_{3,k+1} = 6$ and $b_{3,k+2} = 5$ remaining to conduct, before it finally moves to bay 7. We then have two intervals where crane 3 is located in a larger bay than b^d. The first interval begins at $t_3^s = t_3^\nu + 7$, being the point when 3 has moved from bay 5 to bay 6 after conducting $(3,k)$. The position of 3 when conducting in $(3,k+1)$ is not in conflict with the position of 2 in t_2^ν, however after finishing the request 3 has to move to $b_{3,k+2} = 5$, hence t_3^e equals $t_3^s + 1 = t_3^\nu + 8$. For the second interval we have $t_3^s = t_3^\nu = 11$ and

$t_3^e = 12$.

"Naturally, the non-delay routings of cranes relevant for this sub-model, namely the non-delay sub-routings, differ from the original ones.

- The non-delay sub-routing of crane 1 coincides with the interval $[t_1^\nu, l(n_1)]$ of its original non-delay routing, that is the sequence of operations not conducted yet.
- The non-delay sub-routing of crane 3 coincides with the interval $[t_3^\nu, t_3^e]$ of its original non-delay routing, that is the sequence of operations in $[t_3^\nu, t_3^s]$ being conducted during the detour for sure and the sequence of operations in $[t_3^s, t_3^e]$ potentially conducted during the detour.
- The non-delay sub-routing of crane 2 is not related to its original non-delay routing. It solely represents the evasive move to bay b^e where b^e is the largest bay where crane 3 does not move during its non-delay sub-routing and the move back from b^e to b^d. Note that when processing the non-delay sub-routing waiting suffices in order to prevent interference between cranes 2 and 3 in the submodel.

While in the model in Section 2.1.2.1 start and end points of paths are given, in the submodel only the start point is specified. A potential end point of a path is reached whenever crane 3 has processed at least the first $t_3^s - t_3^\nu$ periods of its non-delay sub-routing and crane 2 has completed its non-delay sub-schedule. These progresses of cranes 3 and 2 correspond to crane 3 having reached time window $[t_3^s, t_3^e]$ in its non-delay routing and crane 2 returning to b^d. At this point, we reached a feasible state, that exists in the model from Section 2.1.2.1. Hence, it can be transferred to the original model and a restart can be applied. Such a path can be evaluated regarding three potentially conflicting objectives.

- The length of a path and, hence the duration of a detour, should be minimized.

- The progress of crane 1 along its non-delay sub-routing and, hence, along its non-delay routing during the detour should be maximized. Note, however, that only paths are comparable where crane 1 is conducting the same request or no request at all.
- The progress of crane 3 along its non-delay sub-routing and, consequently, along its non-delay routing during the detour should be maximized within time interval $[t_3^s, t_3^e]$. Note, however, that only paths are comparable where crane 3 is conducting the same request or no request at all.

When evaluating paths through the sub-model we identify the set of non-dominated paths. Note that each such path identifies the progress along the non-delay sub-routing and, therefore, the progress along the non-delay routing during the detour. Note, furthermore, that crane 2 does not gain any progress along the non-delay routing during the detour. We introduce a segment in the original model for each non-dominated path representing the corresponding detour, accordingly.

In order to provide an example we detail the following setting. Crane 1 has two requests, namely a pickup operation $(1, 1)$ in bay $b_{1,1} = 8$ with duration $d_{1,1} = 1$ and a release operation $(1, 2)$ in bay $b_{1,2} = 2$ with duration $d_{1,2} = 2$. The crane is initially positioned in bay $b_1^0 = 7$ and its assigned parking position after conducting its workload is bay 4. Crane 2 has $b_2^0 = 3$ and conducts two requests also, namely $(2, 1)$ with $b_{2,1} = 5$ and $d_{2,1} = 2$ and $(2, 2)$ with $b_{2,2} = 6$ and $d_{2,2} = 1$. Finally, it has to approach its parking position in bay 2. Crane 3 starts in bay $b_3^0 = 7$ and has to transport two containers from bay 7 to bay 3. The first container takes 3 periods to lift while the second container takes 1 period to lift. Both containers take three periods to drop off. So, there are requests $(3, 1)$, $(3, 2)$, $(3, 3)$, and $(3, 4)$ with $b_{3,1} = b_{3,3} = 7$, $b_{3,2} = b_{3,4} = 3$, $d_{3,1} = 3$, $d_{3,3} = 1$ and $d_{3,2} = d_{3,4} = 3$. After conducting its requests, crane 3 has its parking position in bay 3.

In the depicted instance, all cranes can follow their non-delay routings for four periods without interfering. At this point crane 1 is about to leave bay

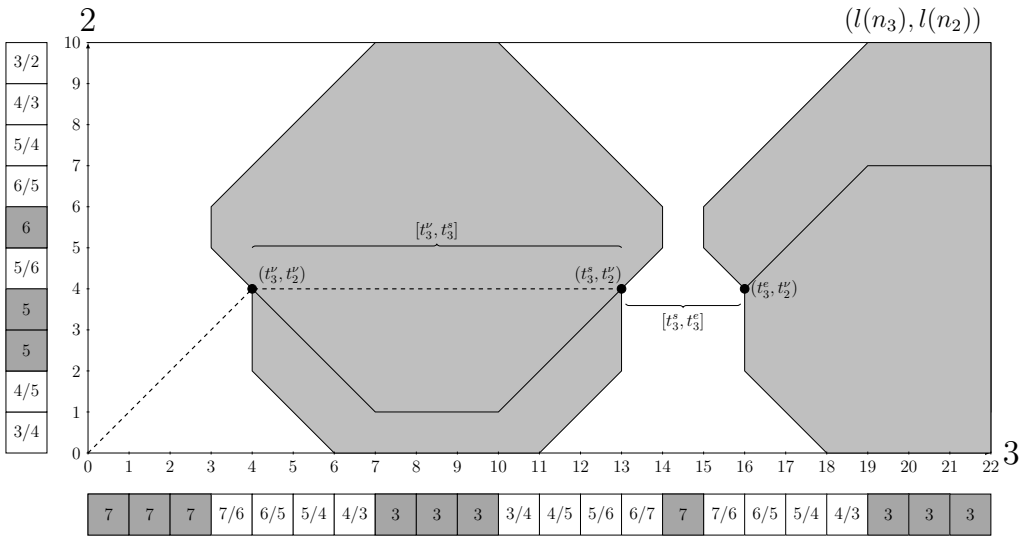


Figure 2.9: A detour of crane 2 for crane 3

6 on its way to approach bay $b_{1,2} = 2$. Crane 2 has just conducted its request $(2, 1)$ in bay 5 and has to enter bay $b_{2,2} = 6$ next. Crane 3 has moved from bay $b_{3,1} = 7$ to 6 and is about to cross the position of crane 2 in order to release the container in bay $b_{3,2} = 3$. The twin-cranes cannot simultaneously process their non-delay routings without interfering, resulting in a case 4 detour of crane 2 prioritizing crane 3 with $\nu = (4/4/4)$. The detour enables crane 3 to conduct request $(3, 2)$ in bay 3 before crane 2 conducts request $(2, 2)$ in bay 6. During this detour the non-delay sub-routing of 2 consists of an evasive move from bay 5 to bay $b_{3,2} - 1 = 2$ and a move back to 5. While the original non-delay routings of 1 and 2 are not in conflict with each other, the evasive move to bay 2 might cause interference with request $(1, 2)$ of crane 1, hence, the corresponding states are covered by an obstacle in the sub-model. The first operation of 3 not in conflict with the position of crane 2 when the detour begins is the move from bay 6 to 7 between conducting requests $(3, 2)$ and $(3, 3)$. The last feasible operation before crossing bay 5 again is the move from 7 to 6 after conducting $(3, 3)$ in bay $b_{3,3} = 7$. As a result, time window $[t_3^s, t_3^e]$ with $t_3^s = 13$ and $t_3^e = 16$ encompasses the three periods of the non-delay routing of crane 3 where it moves from bay 6 to bay

7, conducts request $(3, 3)$, and moves from bay 7 to bay 6.

We depicted the resulting axes of crane 2 and 3 in the graphical model from a two-dimensional perspective in Figure 2.9. The non-delay routings are displayed next to the axes. The dashed line that runs through the obstacle covers the interval $[t_3^\nu, t_3^s]$ of 3's non-delay routing being in conflict with t_2^ν. Hence, this sequence of operations must be conducted by 3 during the detour. Interval $[t_3^s, t_3^e]$, being the progress that crane 3 is allowed to make at most during the detour, is displayed in between the obstacles. Note that in this example the time window is indeed unique since it is the only one where crane 3 is located in bays larger 5 in its remaining non-delay routing. Note, furthermore, that the non-delay sub-routing of 2 is symmetric in its movements and, thus, the obstacles between 3 and 2 (and 1 and 2) in the sub-model are symmetric as well.

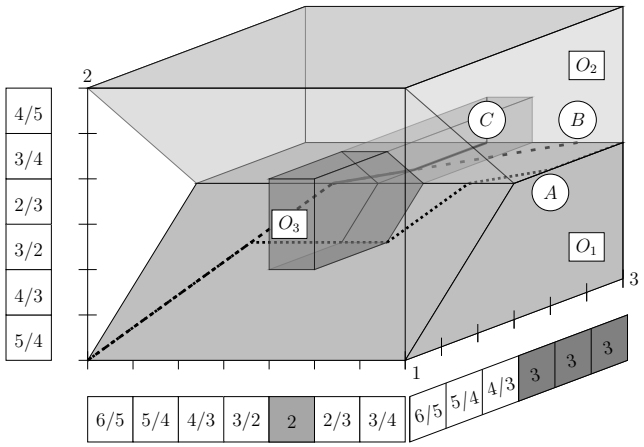


Figure 2.10: Perspective of crane 1 and 2 in the example of the sub-model

Figures 2.10 and 2.11 depict the sub-model for the setting described above. Figure 2.10 emphasizes a view on the axes of cranes 1 and 2. For the sake of clarity we display only the first six periods of the axis of 3. The other perspective, shown in Figures 2.11, emphasizes a view on the complete axes of 2 and 3. Here all states where the twin-cranes would cross positions are covered by either of the two obstacles O_1 and O_2. The obstacle that covers infeasible states of 1 and 2 is denoted by O_3. Feasible paths in this sub-

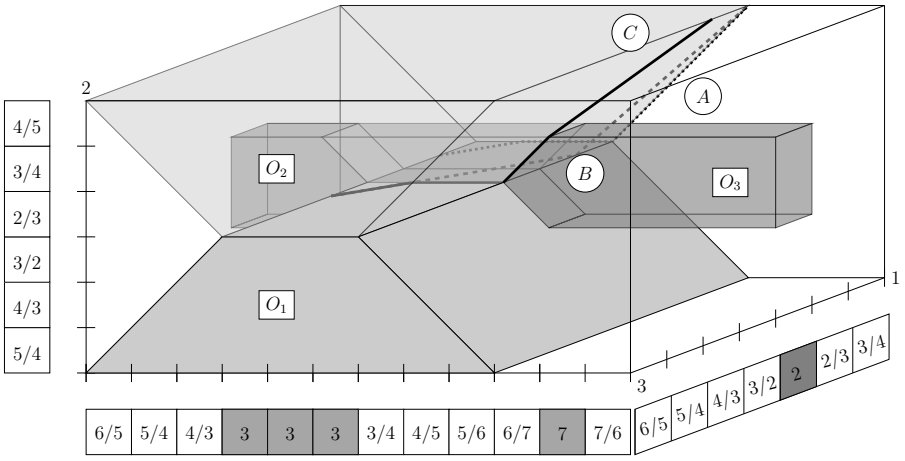


Figure 2.11: Perspective of crane 2 and 3 in the example of the sub-model

model run along the touching surfaces of both of these obstacles, implying that crane 2 waits in bay 2 as long as crane 3 conducts the request $(3, 2)$. Afterwards, the moves of crane 2 are not restricted by crane 3 in the context of this detour since crane 3 moves towards bay $b_{3,3} = 7$ with full speed, following its non-delay sub-routing.

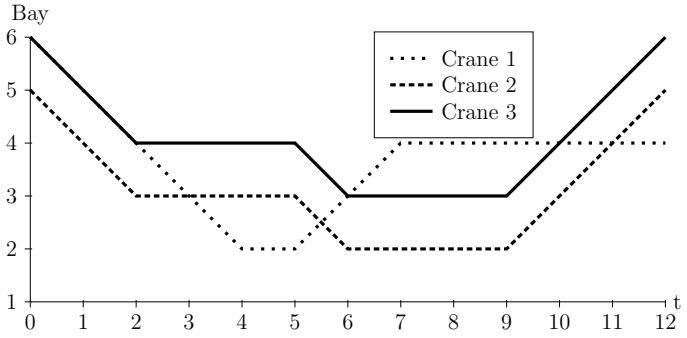


Figure 2.12: Detour A

Following the branching logic proposed in Sections 2.1.2.2 and 2.1.2.3.1 we obtain three paths through the sub-model representing three different instanciations of the detour, namely detours A (dotted lines), B (dashed lines) and C (solid lines). The movement of the cranes during each detour is depicted in Figures 2.12 to 2.14, as well. Analogously to Figure 2.5 we denoted the bay position of the cranes on the y-axis while the periods are depicted on

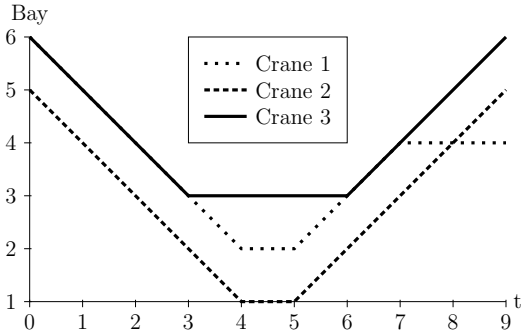


Figure 2.13: Detour B

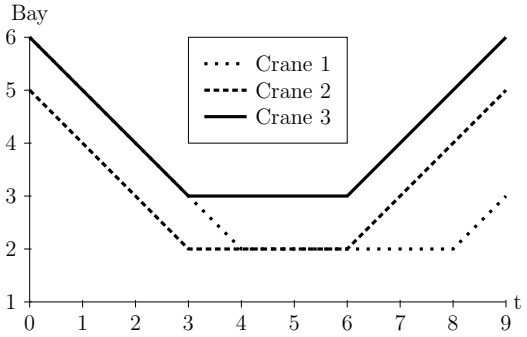


Figure 2.14: Detour C

the x-axis. We give insight into the routings described by the paths. After hitting the surface of O_2, we create the first two branches. The first branch, denoted by A, describes that crane 2 waits for crane 1 during its evasive move for crane 3 in bay 3, and, hence, forces crane 3 to wait in bay 4 until crane 2 can continue the evasive movement. This is represented by the segment of A after the first branch where only crane 1 processes with regard to its non-delay sub-routing. Afterwards, all three cranes process simultaneously until crane 2 has reached bay $b_{3,2} - 1 = 2$. Here, it waits while the other two are processing simultaneously until crane 1 has completed its non-delay routing (and, therefore, its non-delay sub-routing). Now, crane 3 is the only processing crane until it completes $(3, 2)$ and, then, cranes 2 and 3 process their non-delay sub-routings without further interference. The second path B, again, gives priority to crane 1 with respect to cranes 1 and 2 interfering during crane 2's evasive move. Here, crane 2 starts a detour to

bay $b_{1,2} - 1 = 1$ after reaching bay 2, enabling crane 1 to conduct request $(1, 2)$ in bay $b_{1,2} = 2$. Since 2 arrives in $b_{1,2}$ one period before crane 1 and immediately starts the evasive move to $b_{1,2} - 1 = 1$, crane 1 can start $(1, 2)$ without interruption. At the same time when 2 starts the evasive move for 1, crane 3 begins conducting the request in $b_{3,2}$. Crane 2 can begin returning to $b_{3,2} - 1$ when 1 has finished $(1, 2)$, which is two periods after the start of $(3, 2)$. Thus, it finishes the detour for 1 and arrives in $b_{3,2} - 1$ at the same time when crane 3 has conducted the request, hence, both cranes can continue their non-delay sub-routings without further interruption. The third path C gives priority to crane 2 over crane 1, such that crane 1 has to wait until 2 can leave bay $b_{3,2} - 1 = 2$. This is represented by the segment running on the surface of O_2 and O_3. Here, only crane 3 can process since crane 2 is waiting for crane 3 to complete $(3, 2)$ and crane 1 is waiting for crane 2 to leave bay $b_{3,2} - 1 = 2$. After crane 3 has completed $(3, 2)$ both cranes 2 and 3 process for one period while crane 1 is still waiting. After this period, crane 2 has cleared bay $b_{3,2} - 1 = 2$ and all three cranes can process simultaneously. All of the displayed detours end in coordinate (t_3^s, t_2^v) in Figure 2.9. The differences between them are then their lengths and the different progress on the axis of crane 1.

In the example, detour A is dominated by detour B. In both detours crane 1 finishes its non-delay sub-routing and the progress of the twin-cranes is identical. Nonetheless, detour B is shorter since crane 2 does not hinder crane 3 from progressing when prioritizing crane 1. Furthermore, the duration of $(1, 2)$ is short enough such that crane 2 can return to bay 2 at the time when 3 is about to leave bay 3. Hence, crane 2 does not spend more time for detour B than for A. Comparing B and C, we can conclude that B dominates C, as well. The detours' lengths are identical. While crane 1 gains more progress in B than in C, the progress of crane 3 is identical in both detours. We cannot make any statement regarding dominance when comparing detours A and C since both detours lay focus on conflicting objectives. In A the detour is lengthened due to the larger progress of crane 1 whereas C results in less progress of that crane in favor of a shorter length of the detour.

2.1.3 Extensions

In this Section we present two extensions for the former graphical model, namely safety distances as decribed in Section 2.1.3.1 and different container sizes as described in Section 2.1.3.2. Throughout this Section we refer to the model developed in Section 2.1.2 as the basic model.

2.1.3.1 Safety distances between cranes

The basic model assumes that cranes occupy a single bay only and no extra safety distances have to be respected. As a result, it was sufficient to ensure that at each point of time cranes 2 and 3 are located in position b and b' with $b \leq b' - 1$ and neither crane 2 nor crane 3 is located in a position in $]b-1, b+1[$ whenever crane 1 is conducting a request in position b.

For a more general setting let s^t be the safety distance (measured in bays) that has to be kept between the twin-cranes' centres at any given point of time. Analogously, let s^c be the safety distance that has to be kept between a twin-crane's centre and the crossover-crane's centre while the latter is conducting a request. The modified interference constraints are violated in a point $t = (t_1, t_2, t_3)$ of our model if

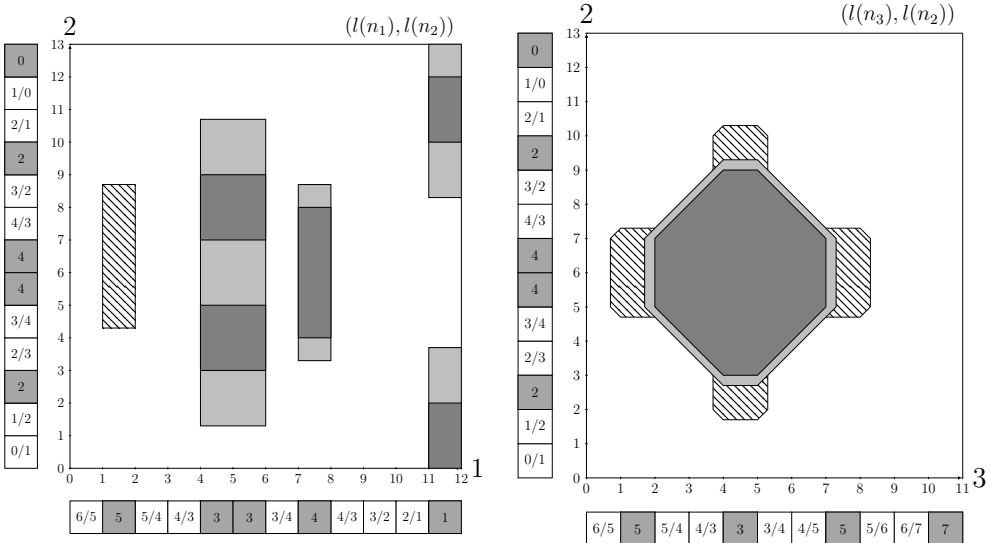
1. crane 1 is conducting a request in bay b and crane 2 or crane 3 is located in a position in $]b - s^c, b + s^c[$ or
2. cranes 2 and 3 are located in positions b and b' with $b > b' - s^t$.

Note safety distances cover the cranes' sizes and that $s^t = s^c = 1$ implements the case we consider in Section 2.1.2. Note furthermore that $0 \leq s^t < 1$ or $0 \leq s^c < 1$ allows to model cranes that do not physically cover a whole bay but only a part of it.

We can easily derive a graphical model that is analogous to the one in Section 2.1.2 but accounts for the modified interference constraints. If $s^t > 1$ ($s^c > 1$) we may have more obstacles and an obstacle for $s^t > 1$ ($s^c > 1$) corresponding

to one with $s^t = 1$ ($s^c = 1$) grows in size but the general shape of the two types of obstacles remains the same.

Figure 2.15 depicts two two-dimensional planes from a graphical model, for pairs of non-delay routings and a safety distance of $s^c = 1.7$ and $s^t = 1.3$ (instead of $s^c = s^t = 1$). On the left hand side we display the resulting obstacles between crane 1 and 2 while on the right hand side the obstacles between and 2 and 3 are depicted. The non-delay routings are annotated next to the axes of the cranes. Again, white squares and the respective notation denote a move from a certain bay to another, while gray squares denote conducting a request in the noted bay. The dark gray covered areas in the plane mark the original obstacle size with a safety distance of one bay. The light gray areas cover the extended space of the original obstacle, resulting from the larger safety distance. Finally, the hatched areas are entirely new obstacles that result exclusively from the larger safety distances.



(a) Safety distance $s^c = 1.7$

(b) Safety distance $s^t = 1.3$

Figure 2.15: Obstacles covering infeasible states due to safety distances of $s^c = 1.7$ and $s^t = 1.3$

One difference to the basic model is that edges of obstacles may have non-integer coordinates in the extended model. As a result segments of the network may have non-integer lengths. However, the network construction as

presented in Sections 2.1.2.2 and 2.1.2.3 does not rely on integer coordinates. As opposed to this, the mathematical model in Section 2.1.1 in fact relies on integer completion times of requests.

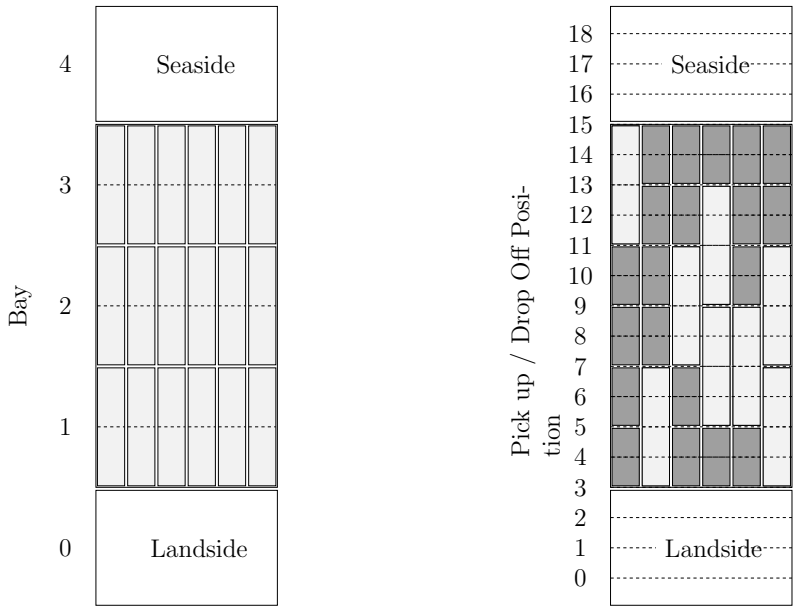
With regard to detours we have to account for the more general view on safety distances, as well. In fact, similar to new obstacles arising from safety distances of $s^t > 1$ or $s^c > 1$ we may have to arrange opportunities for new detours. Nevertheless, from a more generic perspective we apply the same mechanisms considering general safety distances of s^t or s^c instead of $s^t = 1$ or $s^c = 1$.

2.1.3.2 Containers of different sizes

For the basic model, we assumed that containers in the yard are of equal size which then enabled us to discretize the storage yard into bays having a length of exactly one container length. Furthermore, we implicitly assumed that a cranes' size is equal to the length of a container in the basic model. Given that the length of a crane roughly equals the length of a 40 foot container, see Kemme [29], we consequently assumed that only these types of containers are handled. The assumed container layout is shown on the left hand side of Figure 2.16 depicting a container block from above, consisting of 3 equally sized storage bays and two transfer bays at each side of the yard. Each bay can accomodate a 40 foot container. Although the variety of container lengths in seaport terminals is limited and the vast majority of containers belongs to one of two length classes (20 and 40 foot length) the assumption of unique container lengths is restrictive. We will now discuss how the setting treated in Section 2.1.2 can be generalized.

Instead of requiring all containers to be the same length, we can assume that the containers are placed in a grid composed of slots that are 20 foot long. Containers of 20 (40) foot length are placed only such that they fully occupy one slot (two slots).

Whenever a container is picked up or dropped off by a crane, the crane's spreader must be positioned above the center of that container. This gives



(a) Block with equally sized containers

(b) Block with different sized containers

Figure 2.16: Container yard with different sized containers

rise to a discrete set of positions where cranes (their centres) might be located when conducting a request. We number these positions with respect to their alignment. Note that the distance between consecutive positions is unique, and, therefore, we obtain a layout with the same structure as in Section 2.1.2. Such a layout is shown on the right hand side of Figure 2.16 for the same container block as on the left hand side. The darkgray rectangles depict 20 foot containers, while the lightgray rectangles display the 40 foot containers. We see that container positions are aligned with the grid structure. The possible positions from which cranes lift or release a container are marked by dashed lines and the respective position number is shown next to the yard.

Note that a crane usually occupies more than one slot at a time due to its size, see again [29], which is not reflected in our model explicitly. However, by setting safety distances (see Section 2.1.3.1) appropriately we can account for that.

2.1.4 Algorithm implementation

We implemented the algorithm (ALG) described throughout Section 2.1.2 in Java 8. Sections 2.1.2.2 and 2.1.2.3 detail the procedure for generating the whole network. However, we aim at – whenever possible – generating only a part of the network containing the shortest path. We do so by deriving a lower bound whenever a branch is applied to a state as in Sections 2.1.2.2 and 2.1.2.3. Note that while branches have the potential to increase the lower bound associated with a (partial) path, restarts do not. Accordingly, before describing the procedure of generating the network, we detail the lower bounds employed.

When generating the network we maintain lower bounds for the length of each crane's routing in a state. For the initial state these are given as $l(n_1)$, $l(n_2)$, and $l(n_3)$. Whenever a branch or a detour is applied one crane does not process along its non-delay routing for a certain timespan prioritizing another crane. The lower bound of the former crane's routing, then, increases by that timespan accordingly. Partial networks as introduced in Section 2.1.2.2 result from applying a branch and, hence, the increment of the lower bound is related to the whole partial network. For a partial network not overlapping with another it is rather straightforward that the lower bound of the non-processing crane's routing is increasing by the timespan it does not make any progress. However, as discussed in Section 2.1.2.2 partial networks may overlap and we can distinguish three types of overlaps.

1. The processing crane c' in the partial network starting first is the non-processing crane in the second. That is, first crane c waits prioritizing crane c' and while c is waiting, it is decided that c' waits for c'', hence, a second partial networks starts with c' prioritizing crane c''. Then, the lower bound of crane c' increases with regard to the second partial network as described above because it can potentially continue operating after the conflict with c'' is resolved. However, the lower bound of c is increased with regard to both, the first and the second network because it can continue its progress earliest after c' has finished waiting

for c'' and has conducted the operation that initiated the first partial network with c'.

2. The processing crane c' in the partial network starting first is the processing crane in the second network also such that c' is the only crane processing its non-delay routing while the others wait. That is, first crane c prioritizes crane c' and in this partial network crane c'' prioritizes crane c', as well. Then, the lower bounds of cranes c and c'' increase with regard to the first partial network and second partial network, respectively, as described above.

3. The non-processing crane c in the partial network starting first is the processing crane in the second network. That is, first crane c prioritizes crane c' and in this partial network crane c'' prioritizes crane c by anticipating a conflict. Then, the lower bound of crane c increases with regard to the first partial network as described above. However, the lower bound of c'' is increased with regard to the second network and, additionally, to the part of the first partial network overlapping with the second because it can continue earliest after c has finished waiting for c' and after c has conducted the operation initiating the partial network with c''.

The lower bound related to a state and a branch is the maximum among the corresponding lower bounds of the cranes' routing lengths.

On the other hand, we determine upper bounds, as well. We initially set the upper bound $U_{\sum}$ with regard to a feasible schedule where non-delay routings are processed sequentially. Note that a twin-crane is not necessarily positioned in the starting bay b_c^0 when it can start its non-delay routing since it may be on an evasive move for the crane that previously ended its non-delay routing.

The network is then built as follows in order to determine a routing with makespan C_{ALG}. We maintain a list, SL, of branches and restarts to be applied with their respective states. Initially only a restart at $(0, 0, 0)$ is in the

list with lower bound $\max\{l(n_c) \mid c = 1, 2, 3\}$. We maintain the list sorted in increasing order of lower bounds with ties broken by non-increasing euclidean distance of the state to the initial state $(0, 0, 0)$ (reflecting the progress made corresponding to the state). We always apply the first restart/branch in the list and insert resulting restarts and branches accordingly.

Algorithm 1 clarifies the rather simple structure of our algorithm.

Algorithm 1 Network Construction

```
Add (0,0,0) to SL
Set the makespan C_ALG = U_∑
while SL is not empty do
  Select the first branch / restart n in SL
  Remove n from SL
  Apply n resulting in segment s
  if state (l(n_1), l(n_2), l(n_3)) is reached then
    Determine the makespan and update C_ALG if necessary
  else
    Add a restart starting from the end of s to SL (if applicable)
    Add all branches on s to SL
  end if
  Remove all branches / restarts in SL with a lower bound larger or equal
  to C_ALG
end while
```

In Algorithm 1 we omit many details, e.g. how to decide whether a restart is applicable or how to determine all branches on s. These details can be found in Sections 2.1.2.2 and 2.1.2.3.

Further, we develop a greedy heuristic, GRE. Here, we prioritize the crane that arrives in a bay of conflict first, with ties broken arbitrarily. Whenever a deadlock occurs, i.e. situations where the twin-cranes wait for each other to make place for the respective other twin-crane, we initiate a detour of one of the twin-cranes. Again, the crane that would reach the bay of conflict first is prioritized and, therefore, the other crane starts a detour. This heuristic is supposed to mimic a simple but reasonable rule of thumb which can be applied easily in a real world setting. We say the makespan obtained by the

greedy heuristic is C_{GRE}.

As a benchmark, we employ CPLEX 12.6.3 with standard settings in order to solve the MIP model presented in Section 2.1.1.

2.1.5 Experimental Study

In order to evaluate the approaches from Section 2.1.4 we generated a set of test instances with different parameters. All of them are based on a block with 30 bays plus one handover bay on each side where cranes exchange containers with AGVs or trucks. We differentiated two types of workload at the block: each container is to be picked up at one of the handover bays and to be released within the block (st) or we have reshuffle (rs) jobs only, that is each container's pickup bay and drop off bay are within the block. In both workload settings crane 2 starts in bay 0 (in the bay referring to the handover area), crane 3 starts in bay 31 and crane 1 starts in bay 15. Further, we distinguished between two settings regarding the pickup- and drop off locations of the cranes' jobs. In the low-overlap-setting the bays of crane 2's jobs lay on bay-interval $[0, 20]$ while the bays of crane 3's jobs are located between $[11, 31]$. In the high-overlap-setting we increase the intervals to $[0, 25]$ and $[6, 31]$ respectively, supposing that a higher overlap increases potential interference between cranes. In both settings, the requests of crane 1 are located in $[0, 31]$. The number of requests in a test instance is the same for each crane and the time it takes to pick up or drop off a container is randomly drawn from $\{1, \ldots, 5\}$. We generated instances with 4, 8, 16, 24 and 32 requests for each crane by using the test instance generator from [9] as discussed in Briskorn et al. [10].

For combinations covering 4 and 8 requests we created a set of 500 instances for each parameter-combination for which we compared the performance of $CPLEX$ and both solution algorithms. In order to limit the computational burden we created only 20 instances with 16 or more requests per crane, resulting in a total of 4240 instances over all parameters. It does not come as a surprise that the performance of $CPLEX$ depends on the length T of the

time horizon. In order to account for the opportunity to tighten T aiming at a better performance of $CPLEX$ we employed three different values for T: $U_{\sum}$ as described earlier, the makespan C_{ALG} found by ALG (U_A) and 1.5 C_{ALG} ($U_{1.5}$). Regardless of the upper bound we set a solving time limit of one hour. Note that using U_A (and, probably, $U_{1.5}$) gives a substantial advantage to $CPLEX$ since typically U_A (and $U_{1.5}$) will not be known and cannot be determined easily. All tests were conducted on an Intel Core i7-4790 CPU with 3.6 GHz and 32 GB of RAM running Windows 7.

Furthermore we can determine a sophisticated lower bound (LB) using the exact approach for settings with two cranes presented in Briskorn and Angeloudis [5]. For each pair of cranes and respective sequences of requests we determine the optimum schedule ignoring the third crane. The maximum among these three makespan values provides a lower bounds to $TRCIRP$.

In order to evaluate the optimality gap of solutions we use the exact solutions (C^*), for instances with up to 16 requests. For larger instances, we could not provide exact solutions and, therefore, compare our results with the lower bound only.

The results of the computational study are shown in Tables 2.1 and 2.2. Here, the average solution time for all three approaches is depicted in seconds (s). We were able to determine the optimal solution employing $CPLEX$ for every instance with up to 16 requests using U_A.

We observe that the ALG heuristic obtains the optimal solution for every instance with 4, 8 and 16 requests per crane. However, even when we use U_A, $CPLEX$ takes significantly longer to solve an instance than ALG does. When setting the upper bound to $U_{1.5}$ or larger $CPLEX$ is not able to determine optimal solutions for all of the relatively small instances within the time limit. It comes at no surprise that GRE has the shortest run times among all three approaches. We observe that the average gap to the optimal solution of GRE is larger than the average gap of ALG, while run times are nearly the same for the 4 and 8 request test bed. For every test bed with more than 8 requests per crane to handle, GRE clearly outperforms

		CPLEX						*ALG*			*GRE*		
		U_A		$U_{1.5}$		U_Σ							
requests	sett	s	GAP $C^*(\%)$	s	GAP $C^*(\%)$	s	GAP $C^*(\%)$	s	GAP $C^*(\%)$	GAP LB(%)	s	GAP $C^*(\%)$	GAP LB(%)
4	st	0.02	0.00	0.91	0.00	2.09	0.00	0.00	0.00	0.02	0.00	3.54	3.56
	rs	0.15	0.00	4.41	0.00	6.01	0.00	0.00	0.00	0.06	0.00	9.52	9.59
8	st	0.18	0.00	40.58	0.00	827.30	0.00	0.00	0.00	0.07	0.00	4.35	4.42
	rs	15.45	0.00	691.10	0.14	103.05	0.20	0.01	0.00	0.10	0.00	15.03	15.14
16	st	1.51	0.00	1584.43	0.61	3001.53	23.15	0.01	0.00	0.18	0.00	2.79	2.98
	rs	927.71	0.00	3039.77	11.21	3245.04	29.84	0.14	0.00	0.00	0.01	22.04	22.04
24	st	-	-	-	-	-	-	0.04	-	0.05	0.00	-	2.78
	rs	-	-	-	-	-	-	1.28	-	0.00	0.02	-	22.39
32	st	-	-	-	-	-	-	0.05	-	0.08	0.00	-	3.47
	rs	-	-	-	-	-	-	2.28	-	0.12	0.02	-	24.98

Table 2.1: Average run times in seconds and average gap in percent for high-overlap-setting

		CPLEX						*ALG*				*GRE*		
		U_A		$U_{1.5}$		U_Σ								
requests	sett	s	GAP $C^*(\%)$	s	GAP $C^*(\%)$	s	GAP $C^*(\%)$	s	GAP $C^*(\%)$	GAP LB(%)	s	GAP $C^*(\%)$	GAP LB(%)
4	st	0.01	0.00	0.47	0.00	1.13	0.00	0.00	0.00	0.02	0.00	2.83	2.85
	rs	0.02	0.00	1.02	0.00	1.79	0.00	0.00	0.00	0.03	0.00	4.12	4.15
8	st	0.12	0.00	6.89	0.00	10.67	0.00	0.00	0.00	0.06	0.00	5.25	5.31
	rs	0.21	0.00	16.07	0.00	75.07	0.00	0.00	0.00	0.23	0.00	6.86	7.10
16	st	0.86	0.00	715.22	0.14	2051.01	9.82	0.00	0.00	0.00	0.00	1.63	1.63
	rs	15.62	0.00	1534.50	1.61	2005.62	9.05	0.02	0.00	0.00	0.00	5.41	5.41
24	st	-	-	-	-	-	-	0.01	-	0.03	0.00	-	2.23
	rs	-	-	-	-	-	-	0.07	-	0.02	0.00	-	13.30
32	st	-	-	-	-	-	-	0.03	-	0.00	0.00	-	3.37
	rs	-	-	-	-	-	-	0.20	-	0.14	0.01	-	8.09

Table 2.2: Average run times in seconds and average gap in percent for low-overlap-setting

ALG in terms of run times at the cost of a gap to optimality. While *GRE* achieves feasible schedules in virtually zero time we see that the loss in terms of solution quality is significant when a simple rule of thumb is applied. The more elaborate heuristic *ALG* unlocks the potential of better schedules without significantly increasing run times.

It can be seen that the average time it takes to obtain a solution is larger for all three approaches when the overlap of jobs is high. This is in line with our intuition: due to the higher interference potential, and, in terms of the graphical model, more obstacles, the effort for constructing the obstacles and finding a path increases. Similarly, we see that the reshuffle (rs) setting takes longer to solve than the *st* setting. Here, again, the potential interference is higher. Further, during the pickup of a container at the handover bays, the twin-cranes cannot interfere which reduces the amount of potential conflicts for a given number of requests. We observe a similar effect when analyzing the average relative gap of *GRE*. It increases with potential interference resulting from the respective test bed settings.

As expected the run times increase with the number of jobs per crane to be handled. What is interesting to see, is, that the average gap of *GRE* does not increase drastically when increasing the number of requests to be handled, and it even decreases for some settings. While this seems to be surprising at first it actually follows from the fact that the gap between any feasible solution's makespan to the optimum makespan is bounded from above by $\sum_{c \in C} l(n_c) / \max\{l(n_c) \mid c = 1, 2, 3\} \leq 3$ for most instances since one crane is always processing its non-delay routing. Hence, optimality-gaps are very unlikely to increase above 3 but, still, a gap of less than 25% for large instances is quite good for a simple rule of thumb. This is good news for practitioners who prefer to have simple rules applied in the real world. However, we can unlock the remaining potential with virtually no increment in run times applying our admittedly more complicated approach.

We observe that the lower bound is very close to the optimal solution. When analyzing the pairwise-optimal routings of the cranes we see that out of 4240 instances in total, the largest pairwise-optimal makespan was obtained from

the pair of twin-cranes in 2656 cases, while it was obtained by a twin-crane and crane 1 in 1584 cases. This indicates that in the majority of instances the conflicts between twin-cranes affect the solution the most.

When comparing C_{ALG} for 24 and 32 requests with the lower bound we see that the average gap is below 0.2 percent for every combination of parameters. This leads to the assumption that the solution quality of ALG is high, for even larger instances also.

Finally, we want to compare instances differing in the workload setting or in the overlap setting. Table 2.3 outlines the average value for C_{ALG}. We see that on average a larger makespan is obtained for the (st) setting, where jobs are picked up at either side of the yard and transported to a storage location. This comes at no surprise since cranes have longer average travel distances. Similarly, the average objective value is higher for a high overlap.

Table 2.4 details the average relative gap between C_{ALG} and the longest non-delay routing as $(C_{ALG}/\max\{l(n_c) \mid c = 1, 2, 3\}) - 1$ and is shown in percent. We see that in the (rs) setting, where jobs are reshuffled within the yard, the average relative gap is larger than in the storage setting, again pointing at a potentially higher interference between cranes. This assumption is supported by the larger gap of instances with a high overlap of jobs. Nonetheless, the average relative gap is generally decreasing with the number of requests per crane. The reason for this might be as follows. As described above it seems that the twin-cranes are the crucial pair of cranes when it comes to interferences. Now, once one of twin-cranes has waited for the other (or made a detour for the other) there is a good chance they move back and forth synchronously for a while. In the special case where both have to constantly move between bays 0 and B and $B + 1$ and 1, respectively, letting a crane wait once suffices to perfectly synchronize them. Thus, waiting time may not be linear but less than linear in the number of requests which explains our decreasing relative gap.

Summarizing our findings, ALG provides schedules very close to the optimum in very short time (only in two settings the average run time is above 0.2

		Requests				
	sett	4	8	16	24	32
high overlap	st	90	167	304	469	599
	rs	78	132	235	344	434
low overlap	st	81	146	275	396	520
	rs	63	108	186	271	366

Table 2.3: Average value for C_{ALG}

		Requests				
	sett	4	8	16	24	32
high overlap	st	1.13	0.69	0.52	0.47	0.46
	rs	9.17	7.77	9.08	6.53	6.12
low overlap	st	0.16	0.22	0.23	0.20	0.19
	rs	2.12	2.3	1.63	2.12	1.61

Table 2.4: Average relative gap (in percent) between $\max\{l(n_c) \mid c = 1, 2, 3\}$ and C_{ALG}

seconds). Since run times for both approaches are rather small, we do not see an advantage of GRE over ALG."

2.2 Interference aware scheduling of triple-crossover-cranes

The content presented in this section is as well depicted in Briskorn and Zey [7] and all quotes are taken from this very article.

In this section, a holistic scheduling approach is developed, tackling the container job to crane assignment, the job sequence construction as well as the determination of conflict free routings.

"The section has the following structure. We provide a formal definition as well as a MIP model formulation in Section 2.2.1. Further we determine the computational complexity of the problem. The B&Bs are detailed in Section 2.2.2 and a computational study is outlined in Section 2.2.3.

2.2.1 Problem definition and model formulation

In our problem setting we have three RMGs that work jointly on a container yard block. The block is segmented into a set of bays $Q = \{0, 1, ..., B + 1\}$ whereas 0 and $B + 1$ denote a handover bay on the land- or seaside of the block. Here, the cranes can pick up and drop off containers for exchange of containers with transport vehicles.

The cranes are denoted by $C = \{1, 2, 3\}$ with 1 denoting the larger crossover-crane. It moves on a separate pair of rails along the block and has a larger height and width, such that it can cross the twin-cranes when its spreader is completely lifted. Cranes 2 and 3 are the twin-cranes that share a pair of rails and have equal height and width such that they can not cross each other. Nevertheless, they can move below the larger crane when its spreader is up. In the section at hand we denote 2 as the crane that is located in a smaller bay than crane 3.

We start from the premise that the time horizon is continuous and we refer to time interval $[t - 1, t]$ as period $t \in \mathbb{N}\backslash\{0\}$. We consider a set $J = \{1, \ldots, |J|\}$ of transport jobs to be conducted. In order to conduct a job $j \in J$, a crane has to conduct a *pick up* (U) request in bay b_j^U and transport that container to bay b_j^O where it gets *dropped off* (O). The respective duration for which a crane has to stay in the bay while conducting a request are d_j^U and d_j^O, implying all movements of the spreader necessary to pick up or drop off a container. Note that it is implied that the type of crane handling a job does not affect the duration of a request. Nonetheless, the pick up and drop off durations differ between jobs and requests, reflecting the amount of time it takes to adjust the spreader in the respective container location.

At the beginning of the planning horizon the cranes $c \in C$ are positioned in bay b_c^0. They can travel between bays with a velocity of 1 bay per period. We assume that they always travel with full speed or do not travel at all, thus neglecting acceleration and deceleration. As a result we can assume that each crane is always located in a bay $b \in Q$ at the beginning of a period rather than being located at some position between two consecutive bays. It stays

in that bay in a period while waiting or conducting a request. Whenever it moves to $b+1$ $(b-1)$ its position changes according to its progress when approaching that bay. Hence, at any point in time $t \in [t'-1, t']$ it is positioned in $b+(t-t'+1)$ $(b-(t-t'+1))$.

The problem is to make a three part decision, namely the *assignment* of jobs to cranes, the *sequencing* of jobs assigned to the same crane, and the interference free *routing*. We decide each part under the objective of minimizing the makespan and describe them in the following.

1. We have to assign each job to exactly one crane implying the crane a job is conducted by.

2. For each crane and the jobs it gets assigned we have to decide the sequence of picking up in which the crane conducts the jobs. Regarding requests, we then can interpret the sequence of jobs as a sequence of requests to be conducted by having the pickup request related to a job immediately before its drop off request.

3. Given the sequence of requests to be conducted by each crane we, finally, have to decide the position of each crane over time such that it can process its sequence. For picking up (dropping off) of a container j a crane has to be present for d_j^U (d_j^O) periods in b_j^U (b_j^O). The position of a crane, thus, is affected in exactly one of the following ways in each period.

   - The crane moves from bay b to bay $b-1$ or $b+1$ with $0 \leq b-1$ and $b+1 \leq B+1$.
   - The crane stays in bay b while either waiting or (partially) conducting a pick up (drop off) request.

   Another requirement is that if c is located in bay b at the end of period t, then the next activity starts from or takes place in b. While conducting jobs the cranes are not allowed to interfere. This translates to the following more precise requirements.

- If crane 1 is conducting request $o \in \{U, O\}$ of job j in b_j^o in time interval $[t, t + d_j^o]$, then no twin-crane can be positioned in a bay b, with $b_j^o - 1 < b < b_j^o + 1$.
- At each point of time cranes 2 and 3 have to be in positions b and b' such that $b \leq b' - 1$. Note that this requirement implies that a container that has to be picked up or dropped off in bay 0 ($B+1$) cannot be assigned to crane 3 (2).

A routing σ_c of crane c implies a duration it takes crane c to carry it out and we refer to it as its length $l(\sigma_c)$.

A feasible schedule is an assignment of each job to exactly one crane, a sequence of jobs assigned to each crane, and an interference free routing such that the requirements detailed above are met. We represent a feasible schedule by a triple $(\sigma_1, \sigma_2, \sigma_3)$ reflecting the routings and, therefore, implying the assignment and the job sequences. Consequently the makespan of a schedule $(\sigma_1, \sigma_2, \sigma_3)$ is $\max\{l(\sigma_1), l(\sigma_2), l(\sigma_3)\}$. The triple-crossover-crane scheduling problem (TRCSP), then, asks for a feasible schedule with minimum makespan. In the following we settle the computational complexity of TRCSP.

Theorem 1. *The problem to decide whether a feasible schedule to TRCSP not exceeding a given deadline exists is strongly NP-complete.*

Proof. Nossack et al. [37] prove that the corresponding problem in a two crane crossover setting is strongly NP-hard. We can design a very similar reduction mechanism to TRCSP by enlarging the block and assigning an initial position to the third crane which is sufficiently far from the pickup and drop off positions. As a consequence, the third crane becomes irrelevant for finding short schedules and we can apply the same reasoning as in the proof of Nossack et al. [37]. □

Next, a MIP model representing TRCSP is developed. We introduce binary variable $y_{j,c}$ that equals 1 when a job j is assigned to crane c. Furthermore,

binary variable $x^o_{t,j,c}$ equals 1 when conducting request $o \in \{U, O\}$ of job j, performed by crane c is completed in period t (and, thus, on point of time t). Finally, binary variable $z^o_{j,c}$ with $j \in J, o \in \{U, O\}$ indicates whether crane $c \in \{2, 3\}$ is positioned in a larger ($z^o_{j,c} = 1$) or smaller ($z^o_{j,c} = 0$) bay while crane 1 conducts request o of job j. We describe the position of crane c at the end of period t (and, thus, at point of time t) by $p_{c,t}$.

We narrow down the time interval $[e^o_j, l^o_j]$ where request o of job j may be completed. It is not hard to see that we can choose $e^U_j = \min\{|b^0_c - b^U_j| c = 1, 2, 3\} + d^U_j$ and $e^O_j = e^U_j + |b^U_j - b^O_j| + d^O_j$ for each request of a job. Analogous, we can derive latest possible completion time l^U_j and l^O_j for a given upper bound on the makespan. A trivial upper bound is given by assigning all jobs to the crossover-crane in arbitrary processing order. The makespan for this schedule can then easily be computed assuming that crossover-crane can process the corresponding sequence of requests. Possibly, we have to add a single period to that makespan when a twin-crane prevents the crossover-crane from starting the first request immediately. This period, then, accounts for the twin-crane to move out of the way.

$$\text{Min } C_{max} \tag{2.15}$$

$$C_{max} \geq \sum_{t=e^O_j}^{l^O_j} x^O_{t,j,c} \cdot t \qquad \forall c \in C, j \in J \tag{2.16}$$

$$\sum_{c \in C} y_{j,c} = 1 \qquad \forall j \in J \tag{2.17}$$

$$\sum_{t=e^o_j}^{l^o_j} x^o_{t,j,c} = y_{j,c} \qquad \forall j \in J, o \in \{U, O\}, c \in C \tag{2.18}$$

$$\sum_{t=e_j^U}^{l_j^U} t \cdot x_{t,j,c}^U \leq \sum_{t=e_j^O}^{l_j^O} (t - d_j^O) \cdot x_{t,j,c}^O \qquad \forall j \in J, c \in C \quad (2.19)$$

$$\sum_{q=t}^{min\{t+d_j^o-1, l_j^o\}} x_{q,j,c}^o + \sum_{o' \in \{U,O\}} \sum_{i \in J \setminus \{j\}} x_{t,i,c}^{o'} \leq 1 \qquad (2.20)$$

$$\forall t = e_j^o - d_j^o, ..., l_j^o, j \in J, o \in \{U, O\}, c \in C$$

$$\sum_{q=1}^{t} \sum_{j \in J} (x_{q,j,c}^U - x_{q,j,c}^O) \geq 0 \qquad \forall t = 1, ..., T, c \in C \quad (2.21)$$

$$\sum_{q=1}^{t} \sum_{j \in J} (x_{q,j,c}^U - x_{q,j,c}^O) \leq 1 \qquad \forall t = 1, ..., T, c \in C \quad (2.22)$$

$$(B+2) \cdot \left(2 - \left(z_{j,c}^o + \sum_{t'=t}^{min\{t+d_j^o, T\}} x_{q,j,1}^o\right)\right) \geq b_j^o + 1 - p_{c,t} \qquad (2.23)$$

$$\forall c \in \{2, 3\}, j \in J, o \in \{U, O\}, t = e_j^o - d_j^o, ..., l_j^o$$

$$(B+2) \cdot \left(2 - \left((1 - z_{j,c}^o) + \sum_{t'=t}^{min\{t+d_j^o, T\}} x_{q,j,1}^o\right)\right) \geq p_{c,t} - (b_j^o - 1) \qquad (2.24)$$

$$\forall c \in \{2, 3\}, j \in J, o \in \{U, O\}, t = e_j^o - d_j^o, ..., l_j^o$$

$$(B+1) \cdot \left(1 - \sum_{t'=t}^{min\{t+d_j^o, l_j^o\}} x_{t',j,c}^o\right) + b_j^o \cdot \left(\sum_{t'=t}^{min\{t+d_j^o, l_j^o\}} x_{t',j,c}^o\right) \geq p_{c,t} \qquad (2.25)$$

$$\forall t = e_j^o - d_j^o, ..., l_j^o, j \in J, o \in \{U, O\}, c \in C$$

$$b_j^o \cdot \left(\sum_{t'=t}^{min\{t+d_j^o, l_j^o\}} x_{t',j,c}^o \right) \leq p_{c,t} \qquad \forall t = e_j^o - d_j^o, ..., l_j^o, j \in J, o \in \{U, O\}, c \in C \tag{2.26}$$

$$p_{c,0} = b_c^0 \qquad \forall c \in C \tag{2.27}$$

$$p_{c,t} - p_{c,t-1} \leq 1 \qquad \forall c \in C, t = 1, ..., T \tag{2.28}$$

$$p_{c,t-1} - p_{c,t} \leq 1 \qquad \forall c \in C, t = 1, ..., T \tag{2.29}$$

$$p_{2,t} \leq p_{3,t} - 1 \qquad \forall t = 1, ..., T \tag{2.30}$$

$$x_{t,j,c}^o \in \{0, 1\} \qquad \forall t = 0, ..., T, j \in J, o \in \{U, O\}, c \in C \tag{2.31}$$

$$z_{j,c}^o \in \{0, 1\} \qquad \forall j \in J, o \in \{U, O\}, c \in \{2, 3\} \tag{2.32}$$

$$y_{j,c} \in \{0, 1\} \qquad \forall j \in J, c \in C \tag{2.33}$$

The objective function (2.15) represents the goal to minimize the makespan. Constraint (2.16) bounds the makespan from below while (2.17) ensures that each job is assigned to exactly one crane. Constraint (2.18) forces each job to be processed exactly once by the respective crane while (2.19) ensures that the pick up and dropp off requests of a job start after another. Similar, constraint (2.20) ensures that requests of different jobs conducted by the same crane are not carried out in parallel. Constraints (2.21) and (2.22) ensure that a container can only be dropped off when it has been picked up before and a crane can carry at most one container at a time. Constraints (2.24)

and (2.23) make sure that no twin-crane is present in a bay b while crane 1 is carrying out a request in b. Constraint (2.30) prevents conflicts between the twin-cranes. Constraints (2.25) and (2.26) make sure that a crane is present for d_j^o periods in the respective bay b_j^o while conducting request o of job j. Further, these constraints ensure that the position of each crane at each point of time is in Q. The starting position of the cranes are implemented in (2.27). Finally, (2.28) and (2.29) restrict the moving speed of each crane to one bay per period. The domains of the binary decision variables are defined in (2.31), (2.32) and (2.33).

2.2.2 Branch and Bound

In this section we develop two B&Bs in which the problem differing in the sequence particular decisions are made. In the first algorithm we sequentially take all three parts of the decision, that is in the branching process first all jobs are assigned to cranes, then for each crane a sequence of those cranes assigned to it is determined, and finally the routing decision is taken. In the second algorithm assignment and sequences are determined simultaneously. In both cases we use a static branching scheme. We outline details on how the decisions are taken in Sections 2.2.2.1 to 2.2.2.3. Finally, in Section 2.2.2.4 we outline different node order strategies as well as a heuristic in order to determine initial upper bounds.

2.2.2.1 Sequential Assignment and Sequencing

In this section, we treat the algorithm to take assignment and sequencing decisions sequentially. Throughout the first phase, covered by the first $|J|$ levels of the decision tree, jobs are assigned to cranes. Assigning is done one by one and, hence, on level k of the tree (with the root node being on level 0) k jobs are assigned to cranes. Nodes on level $|J|$, then, represent the first part of the decision. The second phase, again, consists of $|J|$ stages and comprises levels $|J| + 1$ to $2|J|$ of the tree. In each stage of this phase we

define the position of a previously assigned job in the sequence of a crane starting at the last position and ending at the first. Clearly, the (partial) sequences corresponding to each node have to be in line with the assignment represented by the node's ancestor on level $|J|$. Nodes on level $2|J|$, then, represent the first two parts of the decision. For such a node the routing is still to be decided. We, thus, evaluate the node by applying the approach by Briskorn and Zey [6] in order to find a good routing for the given assignment and sequences. The routing, then, complements assignment and sequences to a feasible schedule and, therefore, implies its makespan. In the exact variant of our B&B we possibly use a standard solver to determine the optimum routing.

The reason for separating the assignment and the sequencing in our branching scheme is that the assignment decisions alone enable us to find valuable bounds already. We give details on these bounds later. An alternative strategy that comes to mind (and in our opinion is used more often in the literature) is to simultaneously build assignments and sequences. We take into account such an algorithm in Section 2.2.2.2 in which we simultaneously build assignments and sequences followed by the routing. However, it proved to be inferior to branching by separating assignment and sequencing decisions which is why we describe this version in detail in the following.

Figure 2.17 depicts the structure of the resulting search tree for an instance with $|J| = 5$. In levels 1 to $|J|$ jobs are assigned one by one to cranes 1, 2 or 3 while in levels $|J| + 1$ to $2|J|$ jobs are sequenced one by one. Finally, nodes on level $2|J|$ are complemented by a routing decision.

In order to represent the information carried in an arbitrary node we denote the set of jobs assigned to crane c so far by $A_c \subseteq J$. Sequences of jobs are constructed by filling positions in the sequence in decreasing order. For a given *sequence* n_c we can derive the minimum timespan c needs to conduct all jobs, by assuming that it starts in the pick-up bay of the job having the lowest index in n_c. This timespan is denoted as $l(n_c)$. Here, we disregard interference between cranes and assume that crane c can process n_c without any waiting time or detours. Bay b_{n_c} is the bay of the job in n_c having the

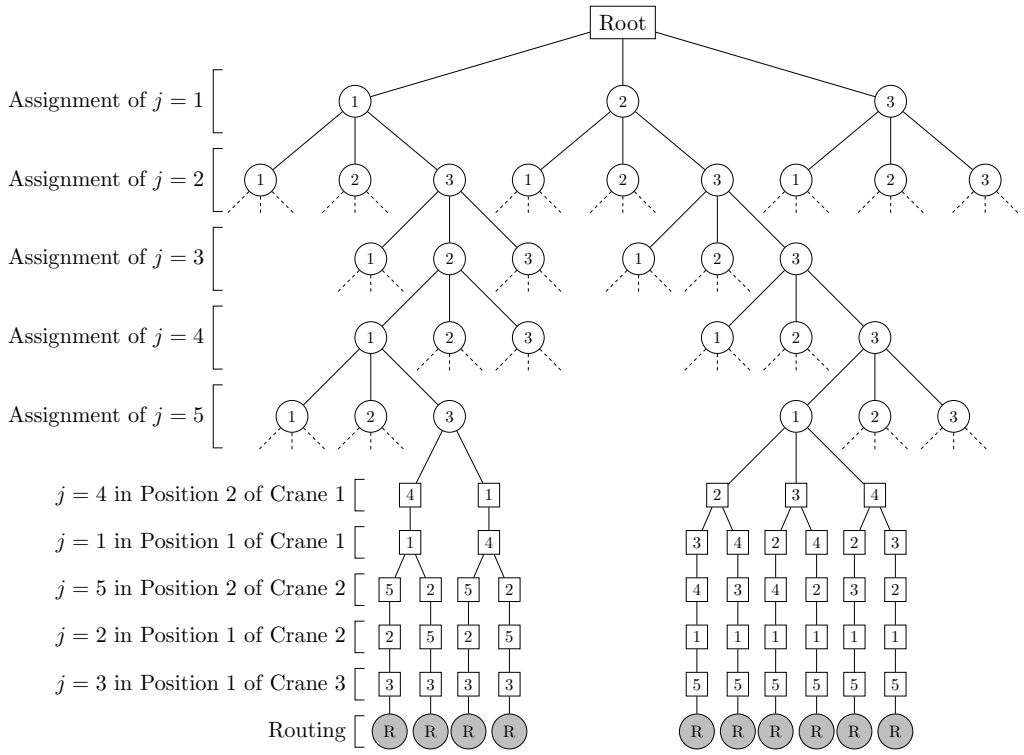


Figure 2.17: Branching example for $|J| = 5$

lowest index.

Naturally, one may wonder if it would be more promising to employ a dynamic branching scheme rather than a static one. We investigated several relaxations we might use for determining promising branching decisions to be taken next. However, in each attempt the potential benefit of dynamic branching decisions was counterbalanced by the higher effort of solving the relaxation and/or maintaining the branching tree.

In the following we outline details of the assignment phase and the sequence phase in Sections 2.2.2.1.1 and 2.2.2.1.2. The routing decision making is presented in Section 2.2.2.3.

2.2.2.1.1 Assignment The assignment phase is outlined in the section at hand. We describe the branching mechanism first and the bounds afterwards.

Branching As mentioned before, the assignment phase consists of $|J|$ stages and on each stage one job gets assigned to each of the cranes. Naturally, we decompose the solutions space appropriately if we create a child node by assigning the current job j to each crane that might conduct it. Hence, if $b_j^U = 0$ or $b_j^O = 0$ ($b_j^U = B + 1$ or $b_j^O = B + 1$) we create no child node for the assignment of j to crane 3 (2).

We consider jobs in non-increasing order of their implied workloads. The workload of job j is $w_j = d_j^U + |b_j^O - b_j^U| + d_j^O$, that is the total duration of pick up and drop off request plus the travel time from its origin to its destination. In the following, we assume that $w_j \geq w_{j+1}$, $j = 1, \ldots, |J| - 1$ which can be achieved by renumbering. Then, nodes on level j are derived from nodes on level $j - 1$ by assigning job j to each crane that can handle it. Doing so leads to potentially tighter lower bounds on low levels already (see Section 2.2.2.1.1 for details on bounds). As a result, there are up to $3^{|J|}$ nodes on level $|J|$ of the tree.

In Figure 2.17 the first $|J| = 5$ levels represent the assignment phase. In stages 1 to $|J|$ jobs are assigned one by one to cranes 1, 2 or 3 as indicated by the numbers in the nodes. The left subtree in stages $|J| + 1$ to $2|J|$, thus, is based on jobs 1 and 4, job 3, and jobs 2 and 5 being assigned to cranes 1, 2, and 3. The right subtree is based on job 5, job 1, and jobs 2, 3, and 4 being assigned to cranes 1, 2, and 3.

Bounding In order to derive lower bounds we first consider the set A_c assigned to crane c. We derive a lower bound for processing all jobs in A_c first and consider those jobs not assigned to any crane yet later on in order to further tighten the lower bound.

For a given set A_c of jobs assigned to crane c, the minimum duration it takes for that crane to conduct the jobs is affected by three factors: the total *workload* of jobs in A_c, the *empty travel time* between drop off position of a job and pick up position of the next job, and the delay due to waiting or even detours implied by the routing. Since total workload $\sum_{j \in A_c} w_j$ assigned to

crane c is a constant empty travel times and routing remain as the influenceable factors. We decided to disregard the latter when deriving bounds. Note that, when interferences are ignored we can derive a lower bound for each crane c independently by bounding the empty travel time implied A_c from below.

We will consider a slightly more general version of the problem than actually necessary at this point where we consider a node of the tree in level k with $k = 1, \ldots, 2|J|$. Hence, n_c might not be empty. While n_c is empty throughout the assignment phase this allows us to fall back on this description when we describe the bounding mechanism in the sequencing phase in Section 2.2.2.1.2.

A lower bound for empty travel time can be found by determining a sequence of jobs in A_c but not in n_c such that total empty travel time is minimized. This problem is very closely related to the one considered in Gilmore and Gomory [25]. We shortly introduce a special case of the problem in Gilmore and Gomory [25] serving our purpose. Here, jobs (corresponding to transport jobs in A_c but not in n_c) have to be processed on a machine (corresponding to the crane). The machine is in a certain state at each point of time (corresponding to the position of the crane over time). An initial *state* (corresponding to the starting position b_c^0 of crane c) is given and so is a final state the machine has to be left in after processing all jobs.

Jobs change the machines state from a start state (corresponding to the jobs origin position) to an end state (corresponding to the jobs destination position). After completing a job the machine has to be brought from this job's end state to the following job's start state before the following job can be processed (corresponding to the empty drive of the crane from a job's destination position to the next job's origin position). Gilmore and Gomory [25] develop a polynomial time algorithm determining the job sequence with minimum changeover costs (minimum empty travel time). Note that the jobs workloads do not have a counterpart in the problem considered by Gilmore and Gomory [25]. However, since total workload is a constant once the assignment is fixed, a sequence having minimum empty travel time has

minimum makespan, as well.

We adapt this algorithm in the following. For final position b' we consider the subproblem $\underline{\text{TRCSP}}(n_c,A_c,c,b')$ to determine the sequence of jobs in $A_c \setminus n_c$ implying minimum travel time for crane c under the assumption that the crane starts in b_c^0 and ends up in b'. Even though TRCSP has no final position required for a crane (which would correspond to the final state of the machine), we can restrict the set of final positions that a crane can take to b_j^O for all $j \in A_c$. Note that we can indeed assume that there is a job j in A_c such that the crane has final position b_j^O since otherwise the crane conducts a superfluous empty drive after completing the last job. We end up with a time complexity of $O(|A_c|^3)$ to determine the minimum empty travel time sequence of all jobs in A_c when conducted by crane c starting from b_c^0. We select the sequence with minimum empty travel time and add the total workload $\sum_{j\in A_c} w_j$, in order to obtain the minimum makespan denoted by $\underline{C^a_{\max}}(c, A_c)$. The lower bound determined for a node is

$$lb_1 = \max\{\underline{C^a_{\max}}(c, A_c) \mid c = 1, 2, 3\}.$$

We can potentially strengthen this lower bound by taking into account the workload implied by those jobs not assigned yet. We derive a lower bound on the total workload plus empty travel time to be managed by the crane system. Obviously, crane c cannot finish before $\underline{C^a_{\max}}(c, A_c)$. Furthermore, these values do not account for the pickup times or drop off times of those jobs not in A_c. Note that the *loaded travel time* $|b_j^U - b_j^O|$ of job $j \notin A_1 \cup A_2 \cup A_3$ may not increase the travel time of a crane c (since it goes empty from b_j^U to b_j^O according to $\underline{C^a_{\max}}(c, A_c)$) it necessarily increases its travel time when crane c does not reach every bay in the interval $\left[\min\left\{b_j^U, b_j^O\right\}, \max\left\{b_j^U, b_j^O\right\}\right]$ of bays. Let b_j^D be the number of bays in $\left[\min\left\{b_j^U, b_j^O\right\}, \max\left\{b_j^U, b_j^O\right\}\right]$ not reached by any crane according to $\underline{C^a_{\max}}(c, A_c)$. Note that this number does not depend on the very schedule assumed for $\underline{C^a_{\max}}(c, A_c)$ but only on the set of jobs in A_c since c reaches the interval $\left[\min\left\{b_j^0, \min\left\{b_j^U, b_j^O \mid j \in A_c\right\}\right\}, \max\left\{b_c^0, \max\left\{b_j^U, b_j^O \mid j \in A_c\right\}\right\}\right]$ due

to the optimality reached by applying the algorithm by Gilmore and Gomory [25]. We, hence, obtain lower bound

$$lb_2 = \left\lceil \frac{\sum_{c \in C} \{\underline{C^a_{\max}}(c, A_c)\} + \sum_{j \notin A_1 \cup A_2 \cup A_3} (d^U_j + d^O_j + b^D_j)}{3} \right\rceil$$

by considering the lower bound on the total workload plus empty travel time and distribute it evenly among the three cranes. We consider lower bound

$$lb_A = \max\{lb_1, lb_2\}$$

for each node in the following. Note that for a node on level k, $k = |J|, \ldots, 2|J|$, $lb = lb_1$ since $A_1 \cup A_2 \cup A_3 = J$.

2.2.2.1.2 Sequencing The sequencing phase is outlined in this section. Here, nodes on level $|J|$ and, therefore, representing an assignment are examined by determining sequences of jobs. Again, we describe the branching mechanism first and the bounds afterwards.

Branching The branching scheme is to design sequences by considering cranes one by one and for each crane to assign jobs to positions of the sequence in decreasing order of positions. By applying such a scheme and consequently branching by constructing a cranes sequence from the last position to the first, the computational burden of determining lower bounds decreases, as described in Section 2.2.2.1.2.

We branch such that the sequence of one crane is fully constructed before switching to the next crane. We do so since we can achieve stronger lower bounds on low levels of the search tree. The order in which cranes are regarded is according to non-increasing lower bounds $\underline{C^a_{\max}}(c, A_c)$ as determined in the corresponding ancestor node on level $|J|$. For every node we construct a child node for each allocation of a job in $A_c \setminus n_c$ to the next open position

in n_c. Consequently, the job-sequences of c are then built on $|A_c|$ levels of the search tree.

In Figure 2.17 levels $|J|+1 = 6$ to $2|J| = 10$ represent the sequencing phase. The comments on the left only apply to the left subtree. The lower bounds corresponding to cranes 1, 2, and 3 corresponding to the ancestor node on level $|J| = 5$ are in decreasing order (this is not represented by Figure 2.17) and, hence, the sequence of crane 1 is constructed first on levels 6 and 7, the sequence of crane 2 is constructed second on levels 8 and 9, and the sequence of crane 3 is constructed last on level 10.

Bounding In order to derive lower bounds we once again adapt the approach by Gilmore and Gomory [25]. By sequencing jobs from the last position to the first, we have both, a given initial state b_c^0, being the starting position of crane c, as well as a given ending state b_{n_c}, being the pick up bay of the job having the lowest index in n_c. Hence, we can derive $\underline{\text{TRCSP}}(n_c,A_c,c,b_{n_c})$ as the optimum sequence of jobs in A_c but not in n_c. Since both, a beginning and an ending state, are given we determine the minimum empty travel time with a time complexity of $O(|A_c|^2)$. If we add the workload $\sum_{j\in A_c\setminus n_c} w_j$, we obtain $\underline{C^s_{max}}(c,n_c,A_c)$ as a minimum duration for processing the jobs in A_c that are not yet in n_c. A lower bound on the time necessary to conduct the jobs of crane c can be described as

$$lb_{3,c} = \begin{cases} \underline{C^a_{\max}}(c, A_c) & \text{if } n_c \text{ is empty} \\ l(n_c) + \underline{C^s_{max}}(c, n_c, A_c) & \text{else} \end{cases}$$

Recall that sequences of cranes are constructed one after another, hence $lb_{3,c}$ equals $\underline{C^a_{\max}}(c, A_c)$ if the sequencing has not yet begun, while it equals $l(n_c)$ after a complete sequence is constructed. Consequently, the lower bound for a node on level k, $k = |J|+1,\ldots,2|J|$, can then be described as

$$lb_3 = \max\{lb_{3,c} \mid c = 1,2,3\}.$$

A second bound, lb_4, is applied after having completely built the job sequences of two cranes. We derive a lower bound for the minimum makespan of an interference-free routing by applying the approach proposed in Briskorn and Angeloudis [5]. The authors basically tackle the two crane versions of the problem to find a minimum makespan interference-free routing for a given sequence of jobs. The authors can prove optimality of their polynomial time approach and, additionally, show that run times are short. By using the complete job sequences of the two cranes as input we can determine the optimum interference-free routing for the respective pair of cranes ignoring the third crane. Obviously, such a routing implies a lower bound such that the maximum among the makespan obtained by the pairwise routing and lb_3 acts as a lower bound for nodes implying the sequencing of the last crane.

2.2.2.2 Simultaneous Assignment and Sequencing

In this section, we treat the approach to take assignment and sequencing decisions simultaneously. We branch by deciding the next entry in one crane's job sequence and, consequently, simultaneously build the assignment and sequences. The branching scheme as well as the bounds employed are detailed in Sections 2.2.2.2.1 and 2.2.2.2.2.

2.2.2.2.1 Branching Throughout the B&B we successively build the job fulfillment sequences of the cranes, by appending an unassigned job to the current partial sequence, again being denoted by n_c, of a crane c. Consequently the phase consists of $|J|$ stages, with a root node on stage 0, such that leafs on the final stage represent a node with complete assignments and sequences.

The set of jobs that have yet to be assigned and sequenced in a node is given by $J \setminus n_1 \cup n_2 \cup n_3$. The trivial approach would be to create a child node for each job in $J \setminus n_1 \cup n_2 \cup n_3$ being assigned to each crane. This, however, leads to identical assignments and sequences achieved by jobs being assigned in different order or, in other words, different leaf nodes representing the same

assignment and sequences. Hence, we employ a variant of this branching scheme avoiding duplicate leafs.

Consider an arbitrary node with j^* being the job with the highest index among all assigned jobs and c^* being the crane that this job has been assigned to. We allow then to assign a job $j \in J \setminus n_1 \cup n_2 \cup n_3$ to any crane, only if $j^* < j$. If we do so, we update j^* and c^* respectively afterwards. Whenever $j < j^*$ we only allow to append that job to n_{c^*}.

The following theorem states that we indeed prevent creating multiple identical leafs using this variant.

Theorem 2. *For each assignment of jobs to cranes and triple of corresponding sequences there is exactly one order of jobs being assigned to cranes in the branching scheme yielding it.*

Proof. The proof consists of two steps. First, we consider an arbitrary assignment of jobs to cranes and triple (n_1, n_2, n_3) of corresponding sequences constructed using our branching scheme. We show that there is exactly one branching order, that is an order in which the branching scheme assigns jobs to cranes, such that the assignment of jobs to cranes and triple of corresponding sequences under consideration is yielded. We conduct the first step by reconstructing the sequence σ representing the branching order. We construct σ from start to end by adding one job at a time and we will see that there is exactly one such sequence.

Let j_c be the first job in n_c that has not yet been appended to σ. Clearly, the next job assigned to a crane must be in $\{j_1, j_2, j_3\}$. We claim that the job assigned next is $j_{c'}$ with $c' = \arg\min\{j_i \mid i = 1, 2, 3\}$ and show it by contradiction. Assume that $j_{c''}$ with $c'' \neq c'$ and, thus, $j_{c''} > j_{c'}$ is assigned next. We distinguish two cases.

- If $j_{c''} < j^*$, then $j_{c''}$ is assigned to c^* by the branching scheme and, thus, $c^* = c''$. Hence, $j_{c'}$ cannot be assigned next to c' by the branching scheme which is the case, however, in the assignment of jobs to cranes and triple of corresponding sequences under consideration.

- If $j_{c''} > j^*$, c^* is set to c'' after assigning $j_{c''}$ to c''. Again, then, $j_{c'}$ cannot be assigned next to c' by the branching scheme.

Second, we show that for an arbitrary assignment of jobs to cranes and triple (n_1, n_2, n_3) of corresponding sequences there is indeed a branching sequence leading to (n_1, n_2, n_3). Let n'_c for each $c \in C$ be a subsequence of n_c as follows. Subsequence n'_c contains the kth element of n_c if and only if it is the largest element in positions $1, \ldots, k$ of n_c. The elements in n'_c are in the same relative order as in n_c. We, then, construct the branching sequence σ as follows. Let j'_c be the first job in n'_c that has not yet been appended to σ. Clearly, the next job in n'_c assigned to a crane is j'_c. In each step we choose $j'_{c'}$ with $c' = \arg\min\{j_i \mid i = 1, 2, 3\}$ to be the next job in σ. Note that at this point $j'_{c'}$ is the largest job in σ. We additionally, append all jobs in $n_{c'}$ between $j'_{c'}$ and its successor in $n'_{c'}$ to σ. Note that the branching scheme assigns these jobs also to c'. □

Figure 2.18 depicts the structure of the resulting search tree for an instance with $|J| = 5$. The nodes indicate job j which is appended to crane c's job sequence. Each workplan represented by a leaf is complemented by a routing decision. On the righthand side of the tree we put job 5 in the first position of the sequence of crane 3. Hence, in all branches resulting from this node we can assign the remaining (smaller) jobs only to crane 3. On the lefthand side we assign job 1 to crane 3 and are consequently allowed to assign any larger job to any of the cranes. In the depicted subtree we append job 3 to the sequence of crane 1 on the second stage, hence, we can then only append job 2 to that crane as well, but can decide freely to which crane we append jobs larger than job 3.

Leafs on the final stage are then evaluated by the routing as depicted in Section 2.2.2.3.

2.2.2.2.2 Bounding Analogously to Section 2.2.2.1.2 the minimum duration of processing partial sequences $l(n_c)$ acts as a lower bound for the

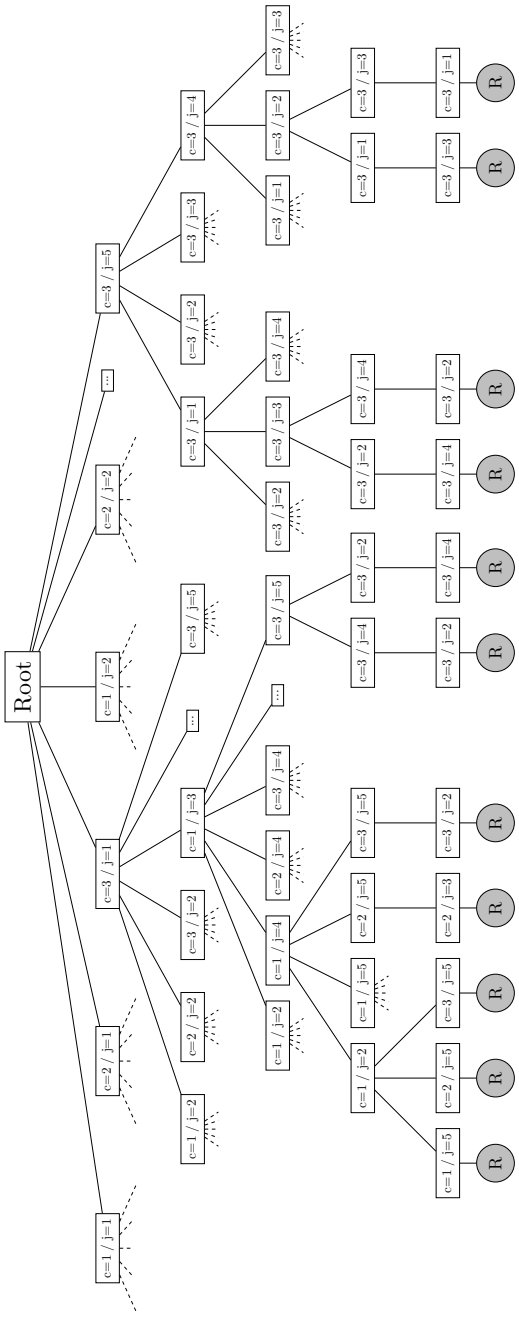


Figure 2.18: Branching example when simultaneously assigning and sequencing jobs for $|J| = 5$

makespan of a crane such that a lower bound on the total makespan can be described by the following term, lb_5. Note that, by building sequences from the first job to the last, $l(n_c)$ can be derived by assuming that c starts in b_c^0 and conducts all jobs in n_c as early as possible.

$$lb_5 = \max\{l(n_c) \mid c = 1, 2, 3\}.$$

We can potentially strengthen the bound by considering unassigned jobs. We do so by incorporating the workload w_j of each job j not assigned yet. Furthermore, bay b_{n_c} is the drop off bay of the job in n_c having the largest index with $b_{n_c} = b_c^0$ if n_c is empty. We incorporate lower bound

$$b_j^e = \min\left\{\min\{|b_{n_c} - b_j^U| c = 1, 2, 3\}, \min\{|b_k^O - b_j^U| k \notin n_1 \cup n_2 \cup n_3, k \neq j\}\right\}$$

of the empty travel time to reach job j. Analogously to lb_2 we, then, obtain lower bound

$$lb_6 = \left\lceil \frac{\sum\limits_{c \in C} l(n_c) + \sum_{j \notin n_1 \cup n_2 \cup n_3} (w_j + b_j^e)}{3} \right\rceil.$$

The lower bound for a node is then the maximum among lb_5 and lb_6.

2.2.2.3 Routing Phase

The final phase of the B&B determines the routing decision. Here, no branching is involved anymore but it can be rather seen as evaluating leaf nodes resulting from branching in the previous phases." Since this leaf nodes represent complete assignments of jobs and the respective sequences in both B&Bs, the routing approach from Section 2.1 is applicable for both algorithms. We present a heuristic routing approach, based on Section 2.1 in the following and show how we extend this approach in order to determine exact solutions for a leaf node.

"First, we determine the interference free routings (and its makespans) for

all pairs of cranes, analogously to in lb_4. If the maximum among these makespans is lower than the current upper bound we determine the heuristic routing as in Section 2.1 in a second step. If the makespan determined by the heuristic routing equals the makespan of any pairwise routing, we obviously found the routing with minimum makespan based on the leaf node. If this is not the case, we determine an optimum interference-free routing using CPLEX applied to the MIP model from Section 2.1.1. We apply the minimum among the (feasible) makespan derived in Step 2. and the current best upper bound as length of the planning horizon. Note that this significantly impacts run times since the MIP model is time-indexed.

Usually, we can prove optimality applying the first two steps only. In Section 2.1.1 it is shown that on average the results of the heuristic approach are less than 0.2% larger than the lower bound obtained by the pairwise routing. Hence, we have to solve the MIP only in a relatively small number of leafs.

2.2.2.4 Node Order Strategies and Upper Bound Heuristic

We can determine an initial upper bound for any of the B&Bs making use of lb_A and the heuristic routing approach presented in prior sections, by means of a simple heuristic, $STRT$. Here, jobs are assigned to cranes in a greedy fashion first. For each unassigned job and crane we compute lb_A, as stated in Section 2.2.2.1.1, under the assumption that the respective job gets assigned to the respective crane. We select the unassigned job and crane combination that increases the lower bound the least, with ties broken arbitrarily and update the respective assignment. The procedure is repeated as long as there are jobs to assign. Finally, we evaluate the complete assignment via the heuristic routing, assuming that the cranes conduct the jobs in order implied by $\underline{C}^a_{\max}(c, n_c, A_c)$. This initial solution is improved afterwards by a hill-climbing mechanism. We exchange pairs of jobs between cranes. After each exchange, we determine $\underline{C}^a_{\max}(c, n_c, A_c)$ and evaluate the implied sequences by the routing. We stop the procedure when no exchange yields an improvement. While there is no structural necessity to provide an initial upper bound when

executing the B&Bs, we can make sure that we provide a feasible solution whenever no routing can be determined within a tight run time limit.

We then develop different strategies how to determine the node in the search tree to be investigated next.

1. In the first algorithm we choose the one having the lowest attributed lower bound among all nodes not having been investigated yet. If there are multiple nodes having the same lowest lower bound, we purse the one on the largest level. We denote the sequential assignment and sequencing B&Bs using this node order strategy with $BEST_H^{SEQ}$ and $BEST_E^{SEQ}$ depending on whether heuristic- and exact routing is applied. The B&Bs with simultaneous assignment and sequencing of jobs are then denoted $BEST_H^{SIM}$ and $BEST_E^{SIM}$ accordingly.

In preliminary tests we noticed that the initial upper bound gets improved for the first time rather late in the procedure when the search tree is large. Hence, we obtain rather bad solutions when time limits are tight. As a result we implemented two additional strategies.

2. The second one is a beam search algorithm. Here we successively investigate only the μ nodes having the smallest lower bounds on a level and discard the remaining nodes. Once the procedure reaches level $2|J|$ all leafs developed are evaluated using the heuristic routing. The main motivation for this approach is to reduce run times. Therefore, only the heuristic routing is applied and we refer to this algorithm as $BEAM^{SIM}$ and $BEAM^{SEQ}$.

3. The third algorithm is similar to the beam search algorithm. We, again, branch the μ best nodes on a level, but do not discard the remaining ones. Whenever there are no nodes left on a level to be processed, we return to the previous level and progress the remaining best μ (or less) nodes that we postponed earlier. The algorithm ends when there are no more nodes to be processed. We therefore denote this complete beam

search algorithm by $CBEAM_H^{SIM}$ ($CBEAM_H^{SEQ}$), when the heuristic routing is applied and by $CBEAM_E^{SIM}$($CBEAM_E^{SEQ}$) for the exact routing step.

2.2.3 Computational Results

For our test beds we set the block size to 31 bays, such that the storage space covers bay 1 to 30 with two handover bays on each side of the block. Crane 1 starts in bay 15, the twin-cranes start in the respective handover bays. The pick up- and drop off durations are between 1 and 5 periods and the respective container positions lay on bay interval $[0, 31]$. For an amount of 5, 10, 15 and 20 jobs we each created 50 instances based on the parameters. Hereby we used the generator from [9] as discussed in [11]. All tests were conducted on a System with Intel Core i7-4790 CPU with 3.6 GHz and 32 GB of RAM running Windows 7. The B&Bs were implemented in Java 8.

In a first step we evaluated the performance of the MIP model as proposed in Section 2.2.1 using CPLEX 12.6.3 with standard settings and compared it to the model formulation by Kemme [29]. Note that the problem tackled in [29] differs from ours only in the objective, which is to minimize the sum of the cranes' waiting times. Hence, both models have equivalent solution spaces and the MIP model proposed by [29] can be modified to represent the goal of makespan minimization easily without major changes. Both model formulations are period based. Nonetheless the position variables used in the model proposed in Section 2.2.1 are time and crane based and are continuous, while the position variables in [29] are binary and are time, crane and bay based. So, the number of binary variables as well as the total number of variables is smaller in our model.

When comparing the MIP models it became apparent, that we could solve only rather small instances. Thus, we only compared the MIP models using instances with 5 jobs and a runtime limit of one hour. Additionally, we tackled the instances with 10 jobs using our MIP model. The time horizon was obtained by $STRT$. The comparison is depicted in Table 2.5, showing the

number of instances for which we could obtain a feasible solution, the average optimality gap to the optimal solution in percent as well as the average runtimes. We were able to determine optimal solutions for every 5 and 10 job instance in the computational study by solving them with $BEST_E^{SEQ}$, as presented in a later part of this section, thus we did not compare the results from Table 2.5 to a lower bound.

	Formulation from Sec. 2.2.1			Kemme [29]		
Jobs	Feasible	GAP	s	Feasible	GAP	s
5	100	0	12.56 s	100	0.69	1123
10	92	16.37	3559.66	-	-	-

Table 2.5: Comparison of MIP models

It can be seen that feasible solutions for all instances with 5 jobs could be obtained within the runtime limit using both MIP models. However, when providing CPLEX with the model formulation proposed in Section 2.2.1, average runtimes were significantly smaller in comparison to the formulation of Kemme [29]. Further, optimum solutions were obtained (and optimality was proven) for every instance when using the formulation from Section 2.2.1, while the usage of the MIP from Kemme [29] results in a small gap. Tackling instances with 10 jobs, runtimes increase drastically with the runtime limit reached for virtually every instance. Even worse, we can determine a feasible solution only for 92 out of the 100 instances. For the feasible solutions found we have a significant optimality gap of about 16%.

In a second step we evaluated the different B&Bs and apply the node order strategies presented in Section 2.2.2.4. We set a runtime limit of one hour, 10 minutes, and 10 seconds, respectively, for each of the algorithms and, after preliminary calibration, set μ to $0.5 \cdot |J|^2 + 30 \cdot |J|$ for the (complete) beam search algorithms. Hereby we employ a beam-width which depends on the number of jobs, and consequently on the number of possible nodes in the tree. The initial upper bound, again, was determined using $STRT$.

We begin by evaluating the B&Bs with sequential assigning and sequencing of jobs as proposed in 2.2.2.1, first. Table 2.6 outlines the results with dif-

ferent time limits. We show average run times as well as the average gap to the optimal solution. Note that the optimal solutions for all instances were obtained by solving them with an infinite time limit if necessary.

We observe that each variant determines feasible solutions significantly faster than the MIP with regard to instances with 5 or 10 jobs using $STRT$. Further, we see that every algorithm, except for $BEAM^{SEQ}$, determines the optimal solution for every instance for this test bed. On average $BEST_H^{SEQ}$ and $BEST_E^{SEQ}$ have smaller run times when compared with the respective complete beam search algorithm. $BEST_H^{SEQ}$ outperforms all other B&Bs in terms of average gap and run times, regardless of the time limit.

The main advantage of $BEAM^{SEQ}$ can be observed when analyzing the results for instances with 15 and 20 jobs: average run times are significantly lower than those of the other B&Bs while the optimality gap remains moderate.

When comparing $BEST_H^{SEQ}$ and $BEST_E^{SEQ}$ with the complete beam search B&B, we see that the average gap as well as run times are larger for both, $CBEAM_H^{SEQ}$ and $CBEAM_E^{SEQ}$, when the time limit is larger or equal to 10 minutes. This can be explained as follows: When the time limit is sufficiently large, both types of algorithms obtain a solution with the same makespan when applying the same type of routing. Nonetheless, when employing $BEST_E^{SEQ}$ and $BEST_H^{SEQ}$ less nodes will be processed, since only nodes with associated lower bounds not exceeding the optimum makespan will be considered. Hence, run times are larger in comparison. On the other hand, when the time limit is tight, $BEST_E^{SEQ}$ and $BEST_H^{SEQ}$ are more likely to fully explore the search tree (again, since a smaller number of nodes is processed).

The main advantage of $CBEAM_H^{SEQ}$ and $CBEAM_E^{SEQ}$ is that the final stage of the search tree is reached potentially earlier. Consequently, μ promising nodes can be evaluated by the routing early, such that the initial upper bound can potentially be updated. When using the best first search algorithms this is not necessarily the case, especially when bounds on larger stages of the

	$BEST_H^{SEQ}$		$BEST_E^{SEQ}$		$CBEAM_H^{SEQ}$		$CBEAM_E^{SEQ}$		$BEAM^{SEQ}$	
jobs	s	GAP	s	GAP	s	GAP	s	GAP	s	GAP
Time limit: 10 seconds										
5	0.01	0.00	0.03	0.00	0.02	0.00	0.03	0.00	0.02	0.00
10	0.48	0.00	0.71	1.40	0.66	0.00	0.74	0.00	0.19	1.11
15	5.88	1.90	6.26	3.03	6.62	3.01	6.86	5.09	0.71	6.85
20	9.65	22.49	9.68	24.50	9.62	5.21	9.65	7.57	1.44	7.05
Time limit: 10 minutes										
5	0.01	0.00	0.03	0.00	0.02	0.00	0.03	0.00	0.02	0.00
10	0.48	0.00	1.30	0.00	0.66	0.00	0.74	0.00	0.19	1.11
15	28.02	0.08	28.70	0.08	74.06	0.49	80.70	0.49	0.71	6.85
20	207.92	1.49	215.33	2.31	255.16	2.18	256.78	2.75	1.44	7.05
Time limit: 1 hour										
5	0.01	0.00	0.03	0.00	0.02	0.00	0.03	0.00	0.02	0.00
10	0.48	0.00	1.30	0.00	0.66	0.00	0.74	0.00	0.19	1.11
15	37.39	0.00	38.06	0.00	146.28	0.32	153.44	0.32	0.71	6.58
20	475.75	0.80	518.49	0.80	1019.12	1.33	1027.99	1.40	1.44	7.05

Table 2.6: Comparison of average solving times in seconds, average gap in percent

tree are significantly tighter than on early stages and the number of nodes is large. We observe the effect, when setting a time limit of 10 seconds and evaluating our B&Bs using instances with 20 jobs. Here $BEST_H^{SEQ}$ and $BEST_E^{SEQ}$ have an average gap of 22 to 25 percent, since they often do not reach parts of the search tree where good solutions can be found.

What is interesting to see is, that the exact algorithms $CBEAM_E^{SEQ}$ and $BEST_E^{SEQ}$ have on average an equal or higher gap in comparison with $CBEAM_H^{SEQ}$ and $BEST_H^{SEQ}$ respectively, while additionally average run times are slightly larger. We find an explanation in Briskorn and Zey [6]. It is shown that the heuristic routing has on average a gap of 0.2% to the lower bound conducted by routing pairs of only two cranes. Consequently, we obtain solutions that equal the lower bound most of the time, and, hence, do not have to use $CPLEX$ in the three step routing approach, since optimality is proven within the first two steps. Table 2.7 gives some insights, when analyzing all instances solved by $BEST_E^{SEQ}$. We outline the quota of instances that were solved without calling $CPLEX$ at all. Further, we provide the total number of leafs explored by the three step routing approach, the number of leafs evaluated by the heuristic routing approach, and the number of leafs evaluated using CPLEX. Finally, we outline the number of leafs where employing CPLEX actually leads to an improved upper bound. Note, that this improvement of the upper bound does not necessarily mean that the optimal solution for TRCSP was found, but only that CPLEX improved any upper bound.

Quota of instances without CPLEX call	59%
Total leafs evaluated in the routing stage	27331636
Total heuristic evaluations	2964
Total calls of CPLEX	616
Total upper bound improvements by CPLEX	35

Table 2.7: Analysis of the instances solved by $BEST_E$

We observe that only a fraction of the leafs are evaluated by the heuristic routing and most of them are discarded due to an insufficient lower bound obtained by the routing of pairs of cranes, lb_4. The heuristic routing eval-

uation, again, leads to a large portion of leafs not explored further and, hence, CPLEX gets called only for a tiny fraction of leafs. This explains the minor increase in run times comparing $CBEAM_E^{SEQ}$ and $BEST_E^{SEQ}$ to $CBEAM_H^{SEQ}$ and $BEST_H^{SEQ}$, respectively. The gap is similar since heuristic routing only very rarely yields worse solutions than exact routing does. In fact, there is virtually no difference between exact routing and heuristic routing when the search tree is inspected entirely (when we do not reach the runtime limit). For the remaining instances higher average effort per node means a smaller number of nodes explored within the runtime limit which explains the increase of the optimality gap comparing the algorithms with an exact routing approach to the ones with heuristic routing.

For both general branching algorithms, with sequential as well as the simultaneous assignment and sequencing of jobs, leafs on the final stage of the search tree represent workplans that are structurally identical. Hence, when conducting the routing, they are evaluated by the same heuristic or exact routing approaches. We therefore expect similar results regarding the influence of CPLEX on the solution quality which consequently leads us to a reduced study on the simultaneous assignment and sequencing of jobs. We tested $CBEAM_H^{SIM}$, $BEST_E^{SIM}$ and $BEAM^{SIM}$ for the same test bed and parameters as for the sequential assignment and sequencing of jobs. When employing $BEST_E^{SIM}$ we ran out of memory for most of the instances with 15 jobs or larger, hence, we cannot provide an extensive comparison. The results are presented in table 2.8.

	$BEST_H^{SIM}$		$CBEAM_H^{SIM}$		$BEAM^{SIM}$	
jobs	s	GAP	s	GAP	s	GAP
Time limit: 10 seconds						
5	0.07	0.00	0.08	0.00	0.07	0.27
10	5.19	10.13	2.14	0.00	0.19	20.30
15	-	-	10.00	6.32	0.40	24.42
20	-	-	10.00	16.09	1.00	27.02
Time limit: 10 minutes						
5	0.07	0.00	0.08	0.00	0.07	0.27
10	13.24	0.00	2.15	0.00	0.19	20.30
15	-	-	511.08	0.50	0.40	24.42
20	-	-	600.00	8.04	1.00	27.02
Time limit: 1 hour						
5	0.07	0.00	0.08	0.00	0.07	0.27
10	13.24	0.00	2.15	0.00	0.19	20.30
15	-	-	1958.04	0.10	0.40	24.42
20	-	-	3600.00	5.02	1.00	27.02

Table 2.8: Comparison of average solving times in seconds, average gap in percent

When looking at the 5 job test bed we observe that the B&Bs have short run times and determine the optimal solution for each instance. However, in comparison to the counterpart algorithms making use of the sequential assignment and routing of cranes they are slower.

When we increase the number of jobs, it becomes apparent that all of the algorithms are outperformed in comparison to the B&Bs with sequential assignment and sequencing of jobs. Either, the search tree becomes too large such that we run out of memory when employing of $BEST_H^{SIM}$ or it takes significantly longer to solve an instance. Additionally the gap to the optimal solution is larger for the 15 and 20 job test bed. We can further observe that $BEAM^{SIM}$ has fast run times in comparison to $CBEAM_H^{SIM}$ as well as $BEAM^{SIM}$ but does not outperform $BEAM^{SEQ}$ in terms of gap, resulting from potentially weak bounds and consequently poor information quality when progressing the μ best nodes on a level of the tree. Nonetheless $BEAM^{SIM}$ is the only algorithm in our study being able to solve the largest instances not slower than one second on average.

What is interesting to see, is that $BEST_H^{SIM}$ takes longer to solve than $CBEAM_H^{SIM}$ and, thus, the results are in contrast to the earlier findings. We analyzed this circumstance by tracking the average number of nodes in the search tree for $BEST_H^{SIM}$, $BEST_H^{SEQ}$ and $CBEAM_H^{SIM}$ when solving the 10 job test bed and detail the findings in the following: when solving instances with $BEST_H^{SEQ}$ the average number of nodes in the tree is 375, for $CBEAM_H^{SIM}$ it is 4438 and for $BEST_H^{SIM}$ it is 296601. The results indicate that the bounds obtained by simultaneously assigning and sequencing jobs are potentially weaker than the bounds obtained by the sequential algorithms. Hence, more nodes have to be processed and stored in the search tree which consequently affects memory requirements and runtimes.

Summarizing our findings, we see that the sequential assignment and sequencing of jobs yields more promising results in comparison to the simultaneous assignment and sequencing, mainly due to better bounds. When analyzing each of the conducted algorithms, we observe that $BEAM^{SEQ}$ provides moderate solution quality with relatively small run times, while $BEAM^{SIM}$ has faster run times with a relatively large gap. We can further observe that $BEST_H^{SEQ}$ and $BEST_E^{SEQ}$ outperform $CBEAM_H^{SEQ}$ and $CBEAM_E^{SEQ}$. Additionally, the solution quality of the heuristic routing approach from Briskorn and Zey [6] is again validated, such that node order strategies making use of only the heuristic obtain solutions faster than with the exact approach while the optimality gap remains small."

2.3 Summary

In this chapter, a holistic scheduling approach for triple-crossover-cranes was developed. First, in Section 2.1, the triple-crossover-crane interference resolving problem (TRCIRP) is presented, aiming at minimum makespan, interference-free routings for a set of three cranes working on a container block. A MIP model, depicting the problem accurately is presented. Afterwards an heuristic approach, based on a graphical representation of the

TRCIRP, is developed, allowing to determine high quality schedules (most of them optimum) in short time.

The proposed heuristic is then employed as a working horse in the holistic scheduling framework presented in Section 2.2, namely the triple-crossover-crane scheduling problem (TRCSP). The problem aims at determining minimum makespan schedules for a set of triple-crossover-cranes by deciding about job assignments, -sequences and conflict free routings of the cranes. It is proved that TRCSP is NP-hard and a MIP model, depicting the problem, was provided. In a second step two B&Bs decomposing the problem were developed. They consist of assigning jobs and deciding about sequences sequentially or of taking both decisions at once, by simultaneously fixing the job assignment and position in the job sequence of a crane. In a final step, job sequences are then evaluated by the routing approach from Section 2.1 in order to obtain feasible schedules. Further, if necessary, the MIP model is solved in order to obtain minimum makespan routings. The sequential assignment and sequencing of jobs yields more promising results in comparison and instances with up to 10 jobs can be solved in under a second on average.

Chapter 3

Scheduling of cooperating Twin-Cranes

The content presented in this chapter is as well depicted in Zey et al. [43] and all quotes are taken from this very article.

"During seaside workload peaks with huge vessels berthed, efficiently storing and retrieving inbound and outbound containers unloaded from ships and to be loaded onto them, respectively, is of utmost importance. To increase the container throughput during these peak times, the landside-crane, although being blocked from a direct access to the seaside transfer point, should support the seaside-crane and share some of the workload. Cooperation among twin-cranes is enabled, if the seaside-crane takes over an inbound container at the seaside access point, but, instead of directly delivering the container towards its dedicated storage position in the block, places the box in an intermediate storage position in between seaside access point and final storage position. Then, the seaside-crane can prematurely return to the seaside access point, whereas the landside-crane completes the previous container move and delivers the container from its intermediate storage position to its dedicated storage position. We call this type of cooperation, where any open storage position is a potential intermediate storage position for a container move subdivided into two legs operated by different cranes, *any-*

bay handover. Sophisticated scheduling procedures coordinating twin-cranes with any-bay handover have, for instance, been introduced in Briskorn et al. [8], Jaehn and Kress [27], Kress et al. [31].

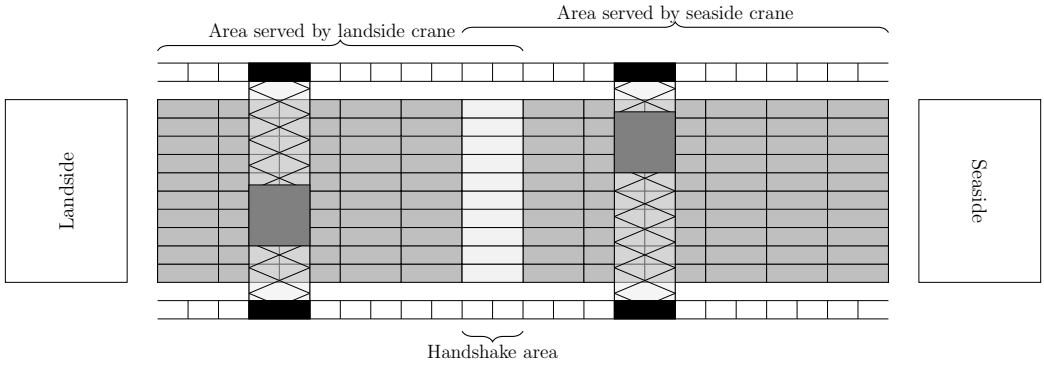


Figure 3.1: Schematic layout of a container block with twin-cranes and a handshake area

Due to the interference of the twin-cranes when handing over containers in facultative bays, these scheduling approaches are very challenging optimization tasks, especially when executed in a real-time environment where any change of the input data requires an (almost) instantaneous plan adaption. Consequently, many real-world terminals prefer to avoid these complexities and separate blocks into dedicated crane areas interconnected by a so-called *handshake area* (see Figure 3.1). A handshake area restricts the handover of containers to a fixed, predefined bay within each block. Both cranes have their dedicated areas, which they operate exclusively without interference, and only when entering the handshake area to handover or takeover a container it has to be ensured that the cranes try not to do so simultaneously. Thus, a handshake area considerably facilitates collision avoidance, and in most real-world terminals cranes simultaneously claiming access to the handshake area are prioritized by simple decision rules, e.g., prefer the crane with the larger remaining workload.

Even with a handshake area, however, scheduling cooperative twin-cranes and avoiding their interference in the shared area still constitutes a complex and challenging optimization task. This section introduces three alternative

branch & bound approaches to solve the resulting optimization problem to optimality. Once a suited exact solution procedure is available (and proven to solve instances of practical size), for the first time, we can quantify the price for planning simple. By benchmarking the simple rule-based approaches commonly applied in real-world terminals with our optimal algorithms the optimality gaps of these simple rules can be quantified. Furthermore, we benchmark the application of a handshake area with an any-bay handover. In this way, practitioners being under high competitive pressure to ensure fast and reliable container handling processes especially during seaside workload peaks receive some decision support on how operate their twin-cranes efficiently.

The remainder of this chapter is structured as follows. Section 3.1 defines the optimization problem (i.e., twin-crane scheduling with a handshake area) and states its computational complexity. Three alternative branch & bound procedures are introduced in Section 3.2, followed by some benchmarking algorithms, being heuristics for the introduced problem as well as an any-bay handover approach, in Section 3.3. The approaches are tested with regard to their computational performance in Section 3.4, in both, a static as well as a rolling horizon planning environment. Finally, Section 3.5 concludes the chapter.

3.1 Problem description

For defining our twin-crane scheduling problem (TCSPH) in presence of a dedicated handshake area, we consider a single yard block, two identical twin-cranes, and two access points from sea- and landside, respectively. The storage positions of the yard block are arranged according to a two-dimensional grid. The slots in the first dimension, we refer to as bays $b = 1, \ldots, B$. The slots in the second dimension are referred to as rows $r = 1, \ldots, R$. Consequently, each *storage position* can be identified by a tuple $(b, r) \in P := \{1, \ldots, R\} \times \{1, \ldots, B\}$. Additionally, we have sea-

side access points aligned at the smallest bay, that is located in positions $(0,1),\ldots,(0,R)$. The handshake area is given by a single bay located at predefined position b^h, $1 \leq b^h \leq B$ and, consequently, consists of positions $(b^h,1),\ldots,(b^h,R)$. In the handshake area, containers may be handed over from one crane to the other. This is implemented by one crane setting down the container in a position (b^h,r), $r=1,\ldots,R$, and the other crane picking up the container in (b^h,r) later on. Each stacking position (b^h,r), $r=1,\ldots,R$, has a *capacity* C^h_r amounting to the maximum stacking height minus the number of containers fixedly stacked in this position. We assume that we have a partition of the positions in the handshake area into two subsets $H_2 = \{(b^h,1),(b^h,3),\ldots,\big(b^h,2\lceil R/2\rceil-1\big)\}$ and $H_3 = \{(b^h,2),(b^h,4),\ldots,\big(b^h,2\lfloor R/2\rfloor\big)\}$ being used exclusively by crane 2 and 3 for dropping off a container.

We only consider moves to and from the seaside, such that we have two sets of *inbound containers* I^i and *outbound containers* I^o all being available at the beginning of the planning horizon. Recall that focusing on such a set of containers is usually done during workload peaks caused by major vessels to be unloaded and loaded. Each container i is associated with two positions. The *origin position* $o_i = (o_i^b, o_i^r)$ is where the container needs to be picked up by a crane. Afterwards it gets transported to *destination position* $d_i = (d_i^b, d_i^r)$ where it gets dropped off. For an inbound container i the origin position o_i is located at the seaside access point, such that it can be described by $(0,o_i^r)$, $o_i^r \in \{1,\ldots,R\}$. The respective destination position d_i is then a position in P. An outbound container i has its origin position in P and their destination position in $(0,d_i^r)$.

Two identical gantry cranes operate on the yard block, which move as a whole along the bays of the first dimension. In order to reach each storage position, a trolley runs along the horizontal beam of the gantry and passes the rows of the second dimension. A spreader can be lowered from the trolley to pick up or drop off a container. We refer to the seaside-crane and landside-crane as crane 2 and crane 3, hence, picking up the notation from Chapter 2, and assume that operation areas are separated by the handshake area, so that

they operate in bays $0, \ldots, b^h$ and $b^h, \ldots, B+1$, respectively. The only shared bay is the handshake area where, however, both cranes cannot be present at the same time. Thus, crane 2 (3) has to be located in a bay $b \leq b^h - 1$ ($b \geq b^h + 1$), while crane 3 (2) operates in b^h.

Due to the separate areas in which the cranes operate, it is implied for each container whether or not it is intermediately dropped off in the handshake area. We, consequently, refer to the set of inbound (outbound) containers that need to be handled by the landside-crane as $I^{i,l} \subseteq I^i$ ($I^{o,l} \subseteq I^o$) and to the set of remaining containers as $I^{i,s} = I^i \setminus I^{i,l}$ ($I^{o,s} = I^o \setminus I^{o,l}$).

For each container, we derive one or two transport *jobs* that are necessary to transport it from its origin to its destination, depending on whether it is intermediately stored in the handshake area. Let J_c be the set of transport jobs of crane c. For each container i in $I^{i,s}$ and $I^{o,s}$, we have a single transport job $j(i)$ in J_2. It consists of two *requests*, being the pick up request in position $\hat{o}_{j(i)} = (\hat{o}^b_{j(i)}, \hat{o}^r_{j(i)}) = o_i$ and the drop off request in $\hat{d}_{j(i)} = (\hat{d}^b_{j(i)}, \hat{d}^r_{j(i)}) = d_i$. For each container i in $I^{i,l}$ ($I^{o,l}$), we have two transport jobs $j^1(i) \in J_2$ and $j^2(i) \in J_3$ ($j^1(i) \in J_3$ and $j^2(i) \in J_2$), referred to as the storage job and the retrieval job. Storage job $j^1(i)$ corresponds to the transport from $\hat{o}_{j^1(i)} = o_i$ to $\hat{d}_{j^1(i)} = (b^h, \hat{d}^r_{j^1(i)})$, $\hat{d}^r_{j^1(i)} \in H_2$ ($\hat{d}^r_{j^1(i)} \in H_3$). Retrieval job $j^2(i)$ has its pickup position in $\hat{o}_{j^2(i)} = \hat{d}_{j^1(i)}$ and the drop off position in $\hat{d}_{j^2(i)} = d_i$. Note that the row $\hat{d}^r_{j^1(i)} = \hat{o}_{j^2(i)}$ of the intermediate storage position in the handshake area is not given in advance.

At the beginning of the planning horizon, crane $c \in \{2, 3\}$ is located in $o^0_c = (o^{0,b}_c, o^{r,0}_c)$. We assume that both trolleys can move one row per period and both gantries can move one bay per period, and they can do so simultaneously. Hence, if no interference of cranes occurs, then moving gantry and trolley from position (b, r) to position (b', r') takes $\max\{|b - b'|, |r - r'|\}$ periods. The spreader can be lowered only after gantry and trolley are in their intended position and it has to be fully up before trolley and gantry can move. We assume that lifting and lowering takes p time periods independent of the current stacking height.

In the setup described above, we have four simplifying assumptions (in addition to those already explained in Section 3, i.e., given handshake area and container movement related with the seaside only), which need further justification.

1. The partition of the handshake area into two subsets to be exclusively used for either inbound or outbound containers aims at reducing the computational and organizational burden. While it reduces the flexibility how to conduct transport jobs, in particular deadlock prevention becomes a lot easier if positions in the handshake area are dedicated to one direction of container transport each.

2. The time for moving a gantry by one bay and a trolley by one row is identical for both cranes. In the current crane generation, the gantry is much faster than the trolley [42]. This, however, is counterbalanced by the larger distance gantries have to cover when moving along the length of a container of a bay. The real-world time difference is, thus, negligible, and this assumption is often applied in crane scheduling research, see [28, 4, 21].

3. The spreader can only be lifted or lowered while gantry and trolley stand still, and, vice versa, they can only start moving once the spreader is back in its upmost position. This is a technical prerequisite of many cranes and a safety restriction in most yards. Note, however, that even if a simultaneous movement is possible and allowed, this feature can hardly be used in practice, because block utilization is usually high and crowded bays can only be passed with a lifted spreader [21]. Consequently, this assumption is often applied in crane scheduling research [27, 31].

4. Duration of pickup and drop off may be position dependent, because positions may differ in the current stacking height. However, significant portions of these durations account for accelerating and slowing down the spreader, adjusting it to the container, and locking or unlocking the

spreader to/from the container. Since these portions do not depend on the very positions, real-world differences are negligible and setting p constant is a pardonable simplification, see also [2].

We consider a continuous time horizon where both cranes can move to adjacent bays in a single time unit and refer to the time interval $[t-1, t]$ with $t \in \mathbb{N}$ as period t.

Within this problem setting, we seek crane schedules defining the activities of both cranes for any period. In order to constitute a crane schedule, we have to make a four-part decision.

1. For each container $i \in I^{i,l} \cup I^{o,l}$ that requires a hand-over between seaside- and landside-crane, we have to decide in which row it is intermediately stored in the handshake area. Thus, we define the positions for the respective jobs in J_2 and J_3.

2. We have to determine a sequence of jobs in J_3 implying the order in which the landside-crane conducts the corresponding transport jobs.

3. We have to determine a sequence of transport jobs in J_2 implying the order in which the seaside-crane conducts the corresponding transport jobs.

4. We have to determine a sequence of pickups and dropoffs in the handshake area, because no two such requests can be carried out in parallel, due to interference constraints between cranes. This sequence has to be consistent with the sequences in 2. and 3., that is the sequences in 2. and 3. imply the order of requests carried out by the same crane. Furthermore, retrieving a container $i \in I^{i,l} \cup I^{o,l}$ from the handshake area has to succeed previous storage and can only be conducted as long as no other containers are placed on top of it. Finally, at each point in time and in each position (b^h, r) in the handshake area, there must not be more than C_r^h containers intermediately stored simultaneously.

We aim at a minimum makespan schedule, that is the point in time when the last container is dropped off at its destination position should be as early as possible. We can assume that each crane proceeds along its sequence of requests (see points 2. and 3. above) as fast as possible (given the travel times between each pair of positions), which may be delayed by interference with the other crane only. That is, crane 2 (3) may have to wait for crane 3 (2) to complete one or more requests in the handshake area before proceeding to the handshake area itself. Moreover, it may be necessary to leave the handshake area in between two requests in order to prioritize the other crane. However, these two types of delay are determined by the sequence of decision 4. Thus, once all four parts of the decision are made, we can easily determine the minimum makespan. The TCSPH is to make the four-part decision, such that an overall minimum makespan can be achieved.

Theorem 3. *The TCSPH is strongly NP-hard even if* $I^{i,l} = I^{o,l} = \emptyset$ *and* $p = 0$.

We abstain from a formal proof and refer to Gharehgozli et al. [22] instead. Note that the problem setting in [22] differs from ours since it has $p = 0$, covers only a single crane, and has access points on landside and seaside of the block. However, our setting with $I^{i,l} = I^{o,l} = \emptyset$ renders crane 3 irrelevant, which leaves us with crane 2 only. Furthermore, Gharehgozli et al. [22] show (although they do not emphasize it) that their problem is NP-hard even if access points on one side of the block are used only. This setting is equivalent to a special case of our problem with $I^{i,l} = I^{o,l} = \emptyset$ and $p = 0$. Clearly, our problem setting is more involved due to the need to coordinate the requests of both cranes with respect to both, physical and temporal interference in the handshake area. Obviously, these issues do not simplify the problem setting.

3.2 Branch & Bound Approaches

We present three branch & bound approaches (B&Bs) in order to solve the TCSPH. The approaches decide the stacking positions of jobs in $I^{i,l} \cup I^{o,l}$

and the sequences of jobs for both cranes (parts 1. to 3. of the decision) in the course of branching.

In the approach in Section 3.2.1 we simultaneously decide about the job sequences as well as possible stacking positions of requests. The approach in Section 3.2.2.1 proceeds by determining job sequences completely before deciding stacking positions while the approach in Section 3.2.2.2 determines the decisions in reverse.

3.2.1 Simultaneous Sequencing and stacking position determination

We propose a B&B which simultaneously constructs job sequences and decides stacking positions (part 1. to 3. of the decision) by branching in this section. As we will show in Section 3.2.1.1 taking these parts of the decision partially implies part 4. of the decision. However, some portion of part 4. remains to be taken. We introduce a strongly polynomial routing approach that determines the optimum sequence of requests in the handshake area (part 4. of the decision) for given sequences of requests for both cranes in Section 3.2.1.2. Finally, in Section 3.2.1.3, we describe how we determine lower bounds.

3.2.1.1 Branching

The branching scheme in this section follows the common idea of building sequences of jobs from start to end by appending jobs one by one. Consequently, we maintain a *(partial) sequence* n_c of jobs for each crane c, $c \in \{2, 3\}$, in a node. We branch by deciding the next job to be handled by one of the cranes and if it is a job with an request in the handshake area we also decide the stacking position of the container in b^h. For containers that have to cross the handshake area it suffices to explicitly decide about the stacking position for either of the resulting storage and retrieval jobs and we do so for the first job being appended. Since a container can only be retrieved

from a position where it gets stored, we implicitly determine the stacking position for both jobs. Consequently, our search tree might reach a depth of $|J_2| + |J_3|$ and each node may have have up to $|I^{i,s} \cup I^{o,s}| + \lceil R/2 \rceil \cdot |I^{i,l} \cup I^{o,l}|$ child nodes.

While the branching scheme explicitly considers parts 1. to 3. of the decision it is open so far how the remaining fourth part is to be taken. In the following we distinguish between the order of requests in the handshake area that take place in the same row and the order of those that occur in different stacking positions. With respect to the former, the following lemma implies a strong connection between parts 1. to 3. and part 4. of the decision.

Lemma 1. For parts 1. to 3. of the decision taken, a sequence of requests is implied for each pair of containers intermediately stored in the same stacking position.

Proof. For the proof we focus on a single stacking position and the requests of jobs taking place in that stack. We consider two containers a and b that need to be intermediately stored in the same stacking position with $j^1(a)$, $j^1(b)$, $j^2(a)$, and $j^2(b)$ being the corresponding storage and retrieval jobs. Clearly, a and b can be picked up only after they have been dropped off.

Let us assume, without loss of generality, that $j^1(a)$ precedes $j^1(b)$ in the job sequence of the crane handling both. Now, we distinguish two cases. First, if container $j^2(a)$ precedes $j^2(b)$ in the other crane's job sequence, as well, then a must be retrieved before b gets stored since otherwise b needs to be retrieved before a can be retrieved. Second, if container $j^2(a)$ succeeds $j^2(b)$ in the other crane's job sequence, then b gets stored before a is retrieved. In both cases we have a distinct order in which the requests in the handshake area related to a and b are conducted. □

In the following it will be more handy to talk about *precedence relations between jobs* which we interpret with regard to requests as follows. For a pair of jobs j and j', j is a predecessor of j' if

- both, j and j', are in n_c, $c \in \{2, 3\}$, and j precedes j' in n_c which means that the drop off request of j has to be conducted before the pickup request of j' or
- j and j' are jobs in different job sequences, have requests in the same stacking position and, among these requests, j's request has to be conducted first according to Lemma 1.

In the first case j is conducted completely before the first request of j' begins. In the second case only the two requests within the stacking position are affected by the precedence relation. The other requests of j and j' having a position outside of the handshake area, however are not directly affected by the precedence relation. Note that we have cyclic (acyclic) precedence relations between jobs if and only if we have cyclic (acyclic) precedence relations between requests.

While Lemma 1 states that there is a sequence of requests for each pair of containers the entirety of sequences of requests in the same stack might be in conflict. This is the case if they imply cyclic precedence constraints between jobs. Otherwise, they imply a unique sequence of all requests for each stacking position in the handshake area, referred to as *stacking position sequence* in the following. For such a sequence we can then determine the number of containers in a stacking position at any time and hence check if the capacity of the position is violated or not. We say a stacking position sequence is feasible if capacity constraints are not violated.

If we apply the proposed branching scheme and append jobs one by one and successively define storage positions in the handshake area, we can construct every pair of job sequences with different stacking positions for jobs. However, such a brute-force branching scheme potentially yields duplicate nodes in the search tree and requires extensive feasibility checks in each step. Therefore, we introduce branching rules restricting the number of child nodes to be created while implicitly ensuring feasibility. Moreover, we employ an additional rule that enables us to prevent duplicate nodes in the search tree, that is distinct nodes representing the same (partial) job sequences for cranes.

In order to specify these rules, let us first refine the types of jobs in J_c, $c \in \{2, 3\}$ that we consider in a node. We say the jobs in $N_c \subseteq J_c$ are the jobs that are handled by crane c only. We introduce N_3 for notational convenience only (note that $N_3 = \emptyset$). Further, the jobs in $S_c \subseteq J_c$ are the jobs that imply the storing of a container in the handshake area, while $R_c \subseteq J_c$ are the retrieval jobs. We decide about stacking positions by branching and, hence, the origin position of some retrieval jobs and the destination position of some storage jobs may not have been decided yet in a node of the search tree. We focus on a single node in the following and say that the jobs in $S_c^k \subseteq S_c$ ($R_c^k \subseteq R_c$) have a row position in the handshake area determined while the jobs in $S_c^u = S_c \setminus S_c^k$ ($R_c^u = R_c \setminus R_c^k$) do not.

The basic idea for implicitly ensuring feasibility while branching is to make sure that by building job sequences from start to end all implied sequences of requests for stacking positions are build from start to end, as well. We, first, introduce the rules and show that we actually reach our goal afterwards.

Rule 1. A retrieval job in R_c can be appended to n_c only if the associated storage job is already appended to the sequence of the other crane $5 - c$.

Rule 1. then allows to append retrieval jobs only after the respective storage job is appended. Consequently, we allow only jobs from R_c^k to be appended. Rule 1 is obviously in line with our goal since a container can be retrieved from the handover bay only after it has been stored there.

Rule 2. Retrieval job $j^2(i)$ can be appended to n_c only if for each storage job $j^1(i')$ succeeding $j^1(i)$ in the same stacking position in n_{5-c} the corresponding retrieval job $j^2(i')$ is in n_c.

Rule 2. is also in line with our goal since a container can be retrieved from the handover bay only if it is the top container at the moment. We will give an explanatory example of Rules 1 and 2 in the following.

Example: Let us consider four containers d, e, f and g and their respective storage and retrieval jobs. Assume that they get stored in the same stacking position by crane c, $j^1(d)$, $j^1(e)$ and $j^1(f)$ are already in n_c and d is the

first container to be stored followed by e and finally f. Further assume that they are the only jobs that get stored in the stacking position and that only $j^2(d)$ is in n_{5-c}. This implies that d is stored first and retrieved first which means that d is retrieved before e is stored. The next allowed retrieval job addressing the stacking position then is $j^2(f)$. For $j^1(e)$ there exists a job ($j^1(f)$) succeeding $j^1(e)$ in n_c with its retrieval job not yet in $n_{c'}$. If $j^2(e)$ precedes $j^2(f)$, then e is retrieved before f is stored and we do not construct the sequence of requests in that stack from start to end. Hence, Rule 2 demands that among $j^2(e)$ and $j^2(f)$ only $j^2(f)$ can be appended next. For $j^2(g)$, the respective storage job is not yet in n_{5-c}, such it cannot be appended either according to Rule 1. Consequently, when resolving the stacking position sequence, e and f get stored, such that container f indeed is the top container for the given sequence pair. Note further that in the example $j^2(d)$ must have been appended to $n_{c'}$ during branching before $j^1(e)$ and $j^1(f)$ got appended to n_c, again due to Rule 2.

We will now show that Rules 1. and 2. indeed allow us to construct the implied sequences of requests in stacking positions.

Lemma 2. By appending jobs with regard to Rules 1. and 2., the order in which requests within the same stacking position get carried out is exactly the order in which the respective jobs get appended to the job sequences.

Proof. We consider two arbitrary containers a and b that get intermediately stored in the same stacking position with $j^1(a)$, $j^1(b)$, $j^2(a)$, and $j^2(b)$ being the corresponding storage and retrieval jobs. Due to Rule 1., $j^1(a)$ or $j^1(b)$ is appended first. Without loss of generality, we assume that $j^1(a)$ is appended first. Again, due to Rule 1., $j^2(a)$ or $j^1(b)$ is appended next.

- If $j^2(a)$ is appended next, then the retrieval request of $j^2(a)$ precedes the storage request of $j^1(b)$ in the stacking position, due to Lemma 1 and the order in which requests related to a or b get carried out coincides with the order in which the respective jobs get appended to the job sequences.

- If $j^1(b)$ is appended next, there is then a storage job ($j^1(a)$) in the stacking position sequence preceding $j^1(b)$ without the respective retrieval job ($j^2(a)$) in the sequence. Then, due to Rule 2., $j^2(b)$ must precede $j^2(a)$, such that $j^1(b)$ precedes $j^2(a)$ as well and , again, both orders coincide.

Summarizing, Rules 1. and 2. allow only two orders in which jobs are appended for two arbitrary containers intermediately stored in the same stacking position. For both the order coincides with the order in which the corresponding requests are carried out. □

Lemma 2 implies that there cannot be cyclic precedence constraints among jobs related to the same stacking position and allows us to define a *state* of each stacking position in a node, as being the current fill level and current stacking order of containers in a position. Such a state allows us to define an earliest possible starting time of the next request in the stacking position. Further, we define precedence relations related to containers that are stored in a stack for a given state. That is, if containers are stored on top of each other we can retrieve the lower container not before the upper container has been retrieved. We incorporate both in the determination of bounds as detailed in Section 3.2.1.3.

Continuing the previous example, the stack is filled with container d after appending $j^1(d)$ and gets emptied after appending $j^2(d)$. Then e and f get stored before they get retrieved in reverse order.

The preceding rules ensured acyclic precedence relations for jobs implying operating within the same stacking position. However, even if we obtain stacking position sequences we have to maintain overall feasible pairs of job sequences. That is, we have to make sure that the entirety of precedence relations between jobs is acyclic.

Let us therefore show, that the proposed branching scheme ensures acyclic precedence relations. We consider the set of precedence relations among jobs given by crane sequences and implied by Lemma 1.

Lemma 3. By applying Rules 1 and 2 we obtain acyclic precedence relations.

Proof. Assume that the precedence relations contain a cycle. Consider job j being the immediate predecessor of job j' in the cycle but j' being appended to its crane's job sequence first. Such a pair exists since the order in which jobs are appended is well-defined. Then, j and j' are jobs of different cranes since precedence relations are not in conflict with job sequences of cranes and requests related to j and j' are in the same stacking position. However, due to Lemma 2 the precedence relations between requests in the same stack are in line with the order in which corresponding jobs are appended. □

The definition of a state resulting from Rules 1. and 2. allows us to define the third rule regarding the jobs that can be appended to n_c in a node.

Rule 3. When appending a storage job we can only select stacking positions with sufficient capacity.

It is obvious that Rule 3. results in only a subset of stacking positions that can be selected for a storage job to be appended to n_c and, hence, allows us to refrain from further capacity checks.

Lastly, we say a pair of (partial) job sequences is feasible if

- all (partial) stacking position sequences are feasible and
- precedence relations among jobs are acyclic.

Due to Lemma 1, Rules 1. and 2. ensure unique stacking sequences. Using Rule 3., obviously, only feasible stacking sequences are constructed. Lemma 3 shows that due to Rules 1. and 2. we obtain acyclic precedence relations. Hence, Rules 1. to 3. ensure feasible (partial) job sequences.

All feasible pairs of sequences can now be constructed as follows. Based on an ancestor node, Rules 1. and 2. define a subset of jobs to be appended while Rule 3 limits the stacking positions to be selected. We create all child nodes, such that in each we

- append a job from N_2 not in n_2 to n_2, that, consequently is not related to a stacking position,
- append a job from R_c^k not in n_c to n_c, $c \in \{2, 3\}$ according to Rule 1 and 2, and
- append a job from S_c^u, $c \in \{2, 3\}$ to n_c, storing a container in every stacking position from H_c as long as Rule 3 is maintained.

However, if we apply the branching scheme as described, we potentially create duplicate nodes in the search tree, e.g. if we first append a job from N_2 to n_2 and afterwards append a job from S_3 to n_3, we construct the same node as if we branch in reverse order. Hence, we propose a final rule in order to prevent this type of redundancy.

Rule 4. If the last job assigned to a crane was appended to n_3, we can append a job to n_2 only if both jobs imply operating in the same stacking position.

We will show that applying Rules 1. to 4. indeed allow us to construct every feasible pair of sequences and there are no two different nodes in the search tree representing the same pair of sequences. For the proof we introduce branching order σ, being the sequence in which jobs are appended one by one by branching, which yields a pair of sequences (n_2, n_3).

Theorem 4. *For each pair of sequences (n_2, n_3), there is exactly one σ following Rules 1. to 4. yielding it.*

Proof. The proof is two-fold. First, we show that for any pair of job sequences constructed by our branching scheme σ is unique. Second, we show that following the scheme we can indeed construct any arbitrary pair of schedules.

For the first part let us consider a pair (n_2, n_3), constructed by the proposed branching scheme. Assume that n_2 and n_3 are constructed up to a certain point and let k_c, with $c \in \{2, 3\}$, be the next job in n_c that is not yet in σ.

Obviously, by appending one job at each level in the search tree, a job from $\{k_2, k_3\}$ is the next job in σ.

We, first, show that, following Rules 1. to 4., only one job in $\{k_2, k_3\}$ can be the next job in σ. We do so by distinguishing the following cases.

- If $k_2 \in N_2$, then k_2 is the next job in σ since k_2 cannot follow a job from J_3 in σ due to Rule 4.
- If $k_2 \notin N_2$ and k_2 and k_3 imply requests in the same stacking position then (n_2, n_3) together with Rules 1. and 2. implies the next job in σ due to Lemma 2.
- If $k_2 \notin N_2$ and k_2 and k_3 imply requests in different stacking positions, jobs are appended as follows.
  - If any k_c, $c \in \{2, 3\}$, has a predecessor in n_{5-c} not yet in σ (all predecessors are implied by (n_2, n_3) due to Lemma 1), this job cannot be the next job in σ due to Lemma 2. Hence, only k_{5-c} can be the next job in σ.
  - If both, k_2 and k_3, have all their predecessors in σ, then Rule 4. implies that k_2 is the next job in σ. Assume that k_3 is the next job in σ. Then, Rule 4. requires that k_2 follows immediately after a job in J_3 implying a request in the same stacking position. However, such a job would be a predecessor of k_2 which we assumed does not exist.

Second, we show that for any feasible pair of job sequences (n_2, n_3) there is a branching sequence σ leading to (n_2, n_3). We do so by giving a procedure constructing σ from (n_2, n_3) and obeying Rules 1. to 4.

1. Let σ be empty.
2. Append jobs from n_2 not in σ yet in the same order until the next job k_2 in n_2 has a predecessor not in σ yet or all jobs in n_2 are appended. Go to 2.

3. Append jobs from n_3 not in σ yet in the same order until all jobs in n_3 are appended or the predecessor of k_2 is appended. Go to 1.

This procedure is in line with the branching scheme. Obviously, the job sequence for each crane is regarded by keeping jobs in the same relative order as in n_2 and n_3. Furthermore, Rules 1. to 4. are accounted for.

- Rules 1. and 2. are being followed since a job follows all its predecessors in σ. This is explicitly ensured by Step 2. for jobs in n_2. Since jobs in n_3 are appended to σ only if necessary to proceed with n_2 we do not have to ensure this in Step 3. due to acyclic precedence relations in feasible pairs of job sequences (n_2, n_3).
- Rule 3. is followed since feasible pair of job sequences (n_2, n_3) implies a unique stacking position sequence for each stacking position accounting for capacity constraints.
- Rule 4. is obeyed since we switch back to Step 2. immediately after appending the predecessor of k_2 to σ.

Concluding, the branching scheme ensures that each feasible pair of job sequences (n_2, n_3) can be constructed. □

The above enables us to fully specify a node in the search tree as the pair of (partial) sequences (n_2, n_3) which allows us to derive all information necessary for applying the branching scheme. However, as the branching scheme only partially covers part 4. of the decision, namely sequences of requests in the same stacking position. Hence, it remains to decide the sequences of requests in different stacking position and conducted by different cranes. We propose a strongly polynomial routing approach in Section 3.2.1.2 and incorporate this approach in Section 3.2.1.3 when determining bounds. Consequently, a leaf in the search tree represents a feasible solution for the TCSPH. We denote this B&B as SIM throughout the computational study from Section 3.4.

We branch then following a best-first-search order based on the lower bounds presented in Section 3.2.1.3. Among nodes having the same lower bound we select the one on the largest level in the search tree. In order to avoid out of memory errors, we deviate from the best-first-search order whenever the size of the search tree exceeds 150000 nodes. Then, we branch only the five best nodes on subsequent levels while keeping all remaining ones, and return to previous levels only when no nodes are left on the current level. Such an approach resembles a beam search and keeps the number of nodes in the tree relatively stable.

We provide an initial upper bound on the makespan by applying a simple nearest neighbor heuristic and we determine upper bounds for nodes by a more sophisticated heuristic. Both approaches are presented in Section 3.3.1.

3.2.1.2 Routing

In this section we describe how the routing of cranes is determined. For two (not necessarily complete) sequences of transport jobs n_2 and n_3, we can determine the routing with minimum makespan by means of a dynamic program (DP) resembling the one in Briskorn and Angeloudis [5] for twin-cranes. Here, we consider rather the sequence of pick up or drop off requests than the sequence of jobs. We number the requests of a crane in increasing order according to n_c. For two such sequences (and implied stacking position sequences) given, we determine the order of requests in the handshake area but in different stacking positions. This is sufficient to deduce an accurate routing for both cranes since all other requests can be conducted by the cranes independently and, thus, without any waiting time or detours.

We have a state s for conflicting pairs of requests within the handshake area. Such a state is specified by (c, f_2, f_3), with f_2 and f_3 being the two conflicting requests of cranes 2 and 3 and c being the crane that gets prioritized with respect to this conflict. It implies that crane c has just finished request f_c and crane $5-c$ is positioned right next to the handshake area, with its trolley in the position according to f_{5-c}. Further we have an initial state s^i where

the cranes are in their initial positions and a final state s^f where the cranes have just conducted their very last request in n_c.

Obviously we can restrict ourselves to only a subset of states, when determining a routing, due to existing precedence relations. Hence, for two conflicting requests f_c and f_{5-c}, we have only a single state if one is a (not necessarily immediate) predecessor of the other.

A transition (s, s') from one state $s = (c, f_2, f_3)$ to another state $s' = (c', f'_2, f'_3)$ then implies that

- crane c' conducts all not yet conducted requests in $[f_{c'}, \ldots, f'_{c'}]$ as early as possible without waiting, starting from its implied position in s and
- if $f_{5-c'}$ is $f'_{5-c'}$, crane $5-c'$ simply stays in the bay next to b^h. Otherwise it conducts all not yet conducted requests in $[f_{5-c'}, \ldots, f_{5-c'}-1]$ without waiting, moves to the bay next to b^h and positions the trolley according to $f'_{5-c'}$.

Obviously, we cannot have a transition for every pair of states. A transition (s, s') exists if and only if

- $f_c < f'_c$ and $f_{5-c} \leq f'_{5-c}$ holds and
- assuming that cranes 2 and 3 process n_2 and n_3 from s on only interrupted by waiting due to precedence constraints they would be present in b^h simultaneously for the first time when conducting f'_2 and f'_3.

The first requirement simply states that no crane c goes backward in n_c and the crane getting priority with respect to the second conflict makes actual progress. The second requirement states the conflict between f'_2 and f'_3 is the first one that actually materializes and, thus, has to be resolved by the routing procedure. Here, we assume that a crane moves to the handshake area only after bringing the trolley next to or in the right position. This ensures that cranes interfere as little as possible without delaying the actual request.

Note that we cannot benefit from letting a crane wait for the other crane to work in the handshake area first if it could have worked there first without delaying the other crane. Since for a transition (s, s') the cranes would interfere without deciding about prioritization with respect to f'_2 and f'_3, it is implied that crane $5 - c'$ can indeed move to the bay next to b^h and position its trolley according to $f'_{5-c'}$ while c' is conducting the request.

The duration $t(s, s')$ associated with transition (s, s') is the timespan crane c' requires to conduct all not yet conducted requests in $[f_{c'}, \ldots, f'_{c'}]$ starting from its progress implied by s. Now, we are ready to define the makespan $t(s')$ associated with state s' as $t(s') = \min\{t(s) + t(s, s') | s \in P(s')\}$ with $P(s')$ being the set of states from which a transition to s' exists.

For the problem at hand we have $O((|I^{i,l}| + |I^{o,l}|)^2)$ potential states and transitions. We can determine if a transition exists (and compute the corresponding duration) in $O(|I^{i,l}| + |I^{o,l}|)$ steps by iteratively checking if the respective pairs of requests are in conflict to each other. Hence we obtain a complexity of $O((|I^{i,l}| + |I^{o,l}|)^3)$.

3.2.1.3 Bounding

In this section we describe how we determine lower bounds on the makespan in a node, that is a makespan for the pair of partial sequences corresponding to the node. A bound consists of two parts, being a lower bound on the lengths of the partial sequences and the duration necessary to conduct the remaining jobs, that are not in the sequences yet. In the following we detail how we account for the second part and how we then use the DP approach presented in Section 3.2.1.2 in order to derive a lower bound for each node.

For the second part, we determine two lower bounds on the time necessary to conduct the non sequenced jobs in A_c, being the jobs in J_c but not in n_c for each crane c. The time necessary depends then on four factors.

1. the *workload* of a job, being the laden travel that is necessary to transport jobs $k \in A_c$ from its origin $\hat{o}_k$ to its destination $\hat{d}_k$ plus the time

necessary to conduct both requests

2. the *empty travel* that can occur between dropping off a container in its storage position and picking up the next container its origin position.

3. waiting times due to precedence relations between jobs

4. waiting times due to interference between cranes

We focus only on the first two factors when determining bounds. Then, we can determine a lower bound $2 \cdot p + \max\{|\hat{o}_k^b - \hat{d}_k^b|, r_k\}$ on the workload w_k for each job $k \in A_c$ with

$$r_k = \begin{cases} |\hat{o}_k^r - \hat{d}_k^r| & \text{for } k \in N_c \cup S_c^k \cup R_c^k \\ 1 & \text{else.} \end{cases} \tag{3.1}$$

Here r_k is a lower bound on the time necessary for the trolley to reach the row where the container gets dropped off after picking it up. Note that for jobs in $S_c^u \cup R_c^u$ the row in the handshake area is not decided yet so we have to assume that the closest row is chosen ultimately." Note further, that in the TCSPH we do not have reshuffling jobs, such that even if $|\hat{o}_k^r - \hat{d}_k^r|$ could be 0 for jobs with unknown stacking positions, the crane would have to travel at least 1 bay in between $\hat{o}_k$ and $\hat{d}_k$. Hence, bounding the time necessary to position the trolley from below by 1 results in a feasible lower bound after all. "Finally, we can define $W_c = \sum_{k \in A_c} w_k$ as a lower bound on the total workload resulting from jobs in A_c of crane c.

Furthermore, we develop two lower bounds on the total empty travel time necessary to conduct the jobs in A_c.

- We consider a bipartite graph where nodes in the first set correspond to drop off requests of jobs in A_c or the last job in n_c and nodes in the second set correspond to pickup requests of jobs in A_c or a dummy end job. An edge between a drop off request k and a pickup request k' represents k' being carried out immediately after k and, therefore,

implies a lower bound (similar to the one in (3.1)) on the empty travel duration. This lower bound is represented by the edge's weight. Note that n_c and n_{5-c} might imply precedence relations between jobs in A_c or between jobs in A_{5-c}. If the pickup request $k \in A_c$ is a predecessor of drop off request $k' \in A_c$ then we can drop the edge connecting k' and k from consideration. Similar as in Gharehgozli et al. [22] we then determine a minimum weight perfect matching. The minimum weight is a lower bound on the total empty travel time. Note that the matching might not imply a proper sequence and we can determine it in $O(|A_c|^3)$.

- Ignoring the time necessary to adjust the trolley position we can derive a lower bound by applying the approach from Gilmore and Gomory [25] in order to determine a sequence for the jobs in A_c with minimum empty travel time. Gilmore and Gomory [25] consider a scheduling problem where a machine (crane) has to process a set of jobs (corresponding to A_c) in a sequence that minimizes the total setup time. Here, every job j has a starting state S_j (corresponding to $\hat{o}_k^b$, with $k \in A_c$) that the machine has to brought in in order to process the job. After finishing j the machine is left in state E_j (corresponding to $\hat{d}_k^b$ for $k \in A_c$). The setup time between jobs j and m amounts to $|E_j - S_m|$. Furthermore the machine has a starting state (being either related to the last job in n_c or o_c^0) and a predetermined ending state that it has to reach after finishing all of the jobs. Gilmore and Gomory [25] develop an optimum algorithm that runs in $O(|A_c|^2)$ time. Note, however, that we do not know the final position of a crane in advance. A straightforward approach would be to fix each job to be the last one in a separate run and end up with a runtime complexity of $O(|A_c|^3)$. This type of approach has been applied in Briskorn and Zey [6] before and performed well. However, we can adapt our branching scheme to construct sequence n_c from end to start which gives us the final position of each crane after the first couple of branching steps. We, then, can straightforwardly apply the approach by Gilmore and Gomory [25] in $O(|A_c|^2)$ time." Note further, that for such a branching scheme all previously defined properties regarding

the stacking positions are still valid, since in between two operations in a stacking position sequence, the number of containers in a stack is identical, regardless whether such a s sequence is conducted forward or backwards.

"Let lb_c^e be the larger of the lower bounds for empty travel for crane c and A_c. A lower bound for the timespan necessary to conduct all jobs in A_c is then given as $lb_c^e + W_c$. We now integrate this lower bound with the routing for partial sequences n_2 and n_3. We apply a slight adaption of the DP approach presented in Section 3.2.1.2 for routing (n_2, n_3) aiming at a minimum makespan assuming that crane c, $c \in \{2, 3\}$, complete its last job $lb_c^e + W_c$ time units later than implied by the routing. This makespan implies a lower bound on the minimum feasible makespan associated with the node of the search tree at hand.

3.2.2 Non-simultaneous sequencing and stacking position determination

In this section we present two B&Bs for which we decide about the job sequences of cranes and stacking positions sequentially. Since most of the ideas presented in Section 3.2.1 carry over straightforwardly we focus on highlighting the differences in the following. The motivation for decoupling both decisions is to keep the width of the search tree small while achieving lower bounds which are similarly tight. Upper bound determination and node ordering is performed analogously to Section 3.2.1.

3.2.2.1 Determining job sequences first

In this section we present a B&B in which we decide about the sequences of jobs of each crane on the first levels of the search tree followed by determining sequences on the last levels. The routing (part 4. of the decision) is determined as described in Section 3.2.1.2.

We append one job to n_2 or n_3 on the first $|J_2|+|J_3|$ levels of the search tree. After having decided about the order of all jobs in the respective sequence, we consecutively decide about the stacking position of each intermediately stored container. Hence the search tree has a depth of $|J_2|+|J_3|+|I^{o,l}|+|I^{i,l}|$. On levels $1,\ldots,|J_2|+|J_3|$ we have up to $|J_2|+|J_3|$ child nodes, on larger levels we have up to $\lceil R/2 \rceil$ child nodes.

On each of the first $|J_2|+|J_3|$ levels of the search tree we append one job to the sequence of a crane. Since we do not determine the stacking positions, we only apply branching Rule 1 from Section 3.2.1.1. Hence, we allow a retrieval job only to be appended to n_c if the corresponding storage job is already in n_{5-c}. Based on Rule 1 only, we can formulate a similar Lemma as in Lemma 3 an show that the obtained sequences contain only acyclic precedence relations. Rule 1 again restricts the set of jobs to be appended, such that on levels $1,\ldots,|J_2|+|J_3|-1$ we create child nodes by

- appending a job from N_2 not in n_2 to n_2,
- appending a job from S_c not in n_c to n_c, $c \in \{2,3\}$, and
- appending a job from R_c not in n_c to n_c, $c \in \{2,3\}$, according to Rule 1.

On levels $|J_2|+|J_3|,\ldots,|J_2|+|J_3|+|I^{o,l}|+|I^{i,l}|-1$ we decide about the stacking positions of containers in $I^{o,l}$ and $I^{i,l}$. On levels $|J_2|+|J_3|,\ldots,|I^{i,l}|-1$ we construct child nodes by assigning containers from $|I^{i,l}|$ to stacking positions in H_2. In a node, we do so in the order in which the storage jobs are sequenced in n_2. On the succeeding $|I^{o,l}|$ levels, we consecutively assign containers from $I^{o,l}$ to stacking positions in H_3 and do so in the order in which the storage jobs are sequenced in n_3.

Based on Lemma 1, we define precedence relations for jobs assigned to the same stacking position. If necessary, we discard nodes with cyclic precedence relations or stacking position sequences violating capacity constraints.

Again, such a scheme potentially yields duplicate nodes on levels $1, \ldots, |J_2| + |J_3|$ in the search tree. Hence, we propose the following Rule, closely related to Rule 4 in order to construct unique nodes only.

Rule 5. If the last job on levels $l = 1, \ldots, |J_2| + |J_3| - 1$ was appended to n_3, we can append a job to n_2 only, if the last job in n_3 is a storage job and the next job in n_2 is its retrieval.

We can show that we can indeed construct every pair of job sequences by a unique sequence of branching steps similar to the proof of Theorem 4.

Even if not every row position is defined, we can apply an adaption of the routing approach presented in Section 3.2.1.2 for a pair of job sequences (n_2, n_3). Here, we employ lower bounds on the time necessary for a trolley to reach a row within the handshake area, as presented in Section 3.2.1.3. Additionally, for levels $1, \ldots, |J_2| + |J_3| - 1$ we determine $lb_c^e + W_c$, as detailed in Section 3.2.1.3. The completion time of c's last request in a routing is then increased accordingly in order to obtain a lower bound on the makespan for a node.

A node in the search tree is then defined by (n_2, n_3). We cannot always derive a certain state of the stacking positions as long as not all jobs taking place in H_c, $c \in \{2, 3\}$ have a defined position. However, during the first phase of the branching (n_2, n_3) suffices to determine the set of jobs A_c that has yet to be scheduled. During the second phase we can determine the jobs that have a yet to be defined stacking position. We denote this approach as SEQ_{JS} throughout the computational study from Section 3.4.

3.2.2.2 Determining stacking positions first

In this section we present a B&B for which we decide stacking positions (part 1. of the decision) of jobs in a first phase of the branching scheme. The second phase consists of determining job sequences (part 2. and 3. of the decision) based on the stacking position assignments. Part 4. of the decision is made as described in Section 3.2.1.2.

On the first $|I^{i,l}|+|I^{o,l}|$ levels of the search tree we decide about the stacking positions of containers in $I^{i,l}$ and $I^{o,l}$. On the subsequent $|J_2|+|J_3|$ levels, we construct sequences of jobs with then already defined stacking positions, by appending a job to n_2 or n_3 on each level of the search tree. Hence, in the first level we can have up to $\lceil R/2 \rceil$ child nodes, while we have at most $|J_2|+|J_3|$ child nodes in the second phase.

In the root node we fix arbitrary orders for containers in $I^{i,l}$ and $I^{o,l}$. Then, on the first $|I^{i,l}|$ levels we create child nodes by consecutively assigning a container, following the order defined in the root node, in $I^{i,l}$ to stacking positions in H_2. On levels $|I^{i,l}|$ to $|I^{i,l}|+|I^{o,l}|-1$ we then define the stacking positions for containers in $I^{o,l}$ and create child nodes by assigning them to every position in H_3, again following the predefined order. Note that, by defining the stacking position of a container i we consequently define the stacking positions for jobs $j^1(i)$ and $j^2(i)$ in J_2 and J_3.

Starting from level $|I^{i,l}|+|I^{o,l}|$ we decide about the job sequences. Each node in this phase has an ancestor node on level $|I^{i,l}|+|I^{o,l}|-1$ with already defined rows for each job. Hence, for nodes in this phase R_c^u and S_c^u are empty. We then apply a slight modification of the branching scheme from Section 3.2.1.1 in order to branch while avoiding duplicate nodes. On each level in the second phase we are then allowed to

- append a job from N_2 not in n_2 to n_2,
- append a job from R_c^k not in n_c to n_c, $c \in \{2,3\}$, according to Rule 1. and 2. from Section 3.2.1.1, and
- append a job from S_c^k not in n_c, $c \in \{2,3\}$ to n_c as long as Rule 3. is maintained.

One can easily see that duplicate nodes are prevented when obeying Rule 4, following the proposed scheme.

Bounding and routing is done using identical or similar approaches as in Section 3.2.1. Having all rows of jobs defined, allows us to employ the approach

from Gilmore and Gomory [25] in order to obtain a sequence for jobs in A_c with minimum empty row travel time as an additional lower bound. We denote this B&B by SEQ_{SJ}.

3.3 Benchmarking Algorithms

This section presents approaches that are used in order to benchmark our B&Bs developed in Section 3.2. While the approach in Section 3.3.2 hardly achieves feasible schedules, due to allowed any-bay handover, the approaches in Section 3.3.1 can either guarantee feasibility or achieve feasible schedule with a high probability. Hence, they can be used as stand-alone heuristics for TCSPH.

3.3.1 Heuristics for the TCSPH

In this section we present two approaches allowing us take the first part of the decision, by defining stacking positions of containers based on the work in Gharehgozli et al. [24]. For the first part of the decision taken, we develop three heuristic scheduling approaches eventually taking the second and third part of the decision. The remaining fourth part of the decision is then determined by the routing approach from Section 3.2.1.2.

While Gharehgozli et al. [24] do not distinguish different stacking positions in a handshake area, the authors decide in which of multiple handshake areas a container is stored (if there exist more than one such area). They propose to minimize the laden travel duration associated with either the storage job or the retrieval job by choosing the stacking position. Note that in our case there might be multiple stacking positions minimizing the laden travel duration. In that case we choose the one closest to the middle row $(R/2)$ among the stacking positions minimizing the laden travel duration of the respective crane. By storing the containers closely together we aim at reducing the time necessary to adjust the trolley in between consecutive

requests in b^h of the same crane. We refer to the resulting set of jobs as J^S and J^R depending on whether laden travel duration of storage jobs or retrieval jobs has been minimized.

Regarding the sequencing of jobs with already defined stacking positions for each container, we apply the nearest neighbor scheduling approach from Gharehgozli et al. [24]. We build a sequence of jobs for crane c, by consecutively appending jobs to n_c following the nearest neighbor approach. Here, a job with minimum distance to the drop off position of the last request in the sequence of a crane is conducted next. Afterwards, we determine a sequence for crane $5-c$ from the first position to the last, again, following the nearest neighbor approach. While Gharehgozli et al. [24] do not consider stacking sequences we modify the approach in order to ensure feasible pairs of sequences by requiring jobs corresponding to containers in $I^{i,l} \cup I^{o,l}$ to be handled in the same order by both cranes. We run the approach with $c = 2$ and $c = 3$ in separate runs and choose the better among the resulting schedules. We denote the heuristic by $NN(J^S)$ and $NN(J^R)$, respectively depending on whether laden travel duration associated with the storage job or the retrieval job is minimized. For the B&Bs we determine $NN(J^S)$ and $NN(J^R)$ ahead of branching and which ever yields the lower makespan provides an initial upper bound.

The second adapted scheduling approach is a local search heuristic based on J^S or J^R. We use the solution found by $NN(J^S)$ ($NN(J^R)$) as an initial solution. Then, for α iterations, we choose two jobs randomly in each sequence and swap them. The swap is accepted whenever the pair of resulting sequences is a feasible and the makespan (evaluated by using the routing approach in Section 3.2.1.2) decreases. The respective variants of the local search heuristic are denoted by $2OPT(J^S, \alpha)$ and $2OPT(J^R, \alpha)$.

The third scheduling approach is only loosely based on the work of Gharehgozli et al. [24] and is a more sophisticated nearest neighbor heuristic. Let c be the crane having the larger workload W_c among cranes based on jobs with already defined stacking positions. We determine sequences with minimum empty travel time with respect to either bay distances or row distances

employing the approach by Gilmore and Gomory [25] (see Section 3.2.1.3 for details). Among both sequences we choose the one having smaller total empty travel time (with respect to both, bay distances and row distances) as job sequence of crane c. Afterwards, we construct the job sequence of crane $5-c$ using the nearest neighbor approach. Once again, we require containers assigned to the same stacking position to appear in the same order in both job sequences. If multiple jobs can be selected, ties are broken in favor of a job implying operating in the handshake area and among those in favor of the job with the earliest corresponding job in the sequence of crane c. Note that the approach does not guarantee feasible job sequences such that a routing cannot necessarily be determined. We denote the approach by $SNN(J^R)$ and $SNN(J^S)$ respectively depending on which set of jobs is provided.

Within the B&Bs we employ the sophisticated nearest neighbor approach in order to determine upper bounds on the makespan for nodes. Here, we only construct sequences for jobs in A_c and A_{5-c} in a node. Undefined stacking positions of containers are defined based on J^S or J^R, depending on for which the simpler nearest neighbor heuristic yields a smaller makespan. Even though all scheduling heuristics presented in this Section are suitable for determining upper bounds, we select the sophisticated nearest neighbor heuristic since it runs faster than the $2OPT$ heuristics and obtains better solutions than the simpler nearest neighbor heuristic, as we will see in Section 3.4.

3.3.2 Dynamic container handover

In this Section we present an approach in which the cranes are allowed to intermediately store a container in every position within the yard. It is based on Briskorn et al. [8] where a setting is tackled where containers exclusively arrive at the seaside access point and need to be stored within the yard. The block is represented in a one-dimensional model, that is only bays are considered and trolley moves are neglected. Handovers between cranes are not restricted to a handshake area but are allowed in any bay, capacities are

assumed to be infinite, and no stacking sequences are considered. Initially, containers are picked up by the seaside-crane, that afterwards is allowed to hand over the container to a landside-crane. The authors aim at minimizing makespan and develop – among others – a bucket-brigade scheduling approach, see e.g. Bartholdi and Eisenstein [1], that obtains close to optimum results. Here, crane 2 hands over a container to 3 whenever the cranes are positioned close to each other, and 3 is not carrying a container. Afterward 3 transports the container to its final destination.

For the approach proposed in this section, we neglect capacities as well as precedence relations for jobs in the same position and allow containers to be intermediately stored in any position. Obviously, it does not yield feasible solutions for the TCSPH and serves as benchmark only. Hence, we explain the approach only briefly. Clearly, the optimum schedule to this setting constitutes a lower bound to the TCSPH. As opposed to Briskorn et al. [8], we consider two types of containers, namely in I^i and I^o, and moves of the trolley and, therefore, have to adapt the bucket-brigade scheduling approach as presented in the following.

For the proposed approach we omit from determining all parts of the decision in advance and only decide about the sequence $\overline{n}_2$ in which crane 2 handles containers in $I^o \cup I^i$, as detailed at the end of this section. For such a sequence given, we decide dynamically about the very next container to be handled by crane 3. Such a decision is made at the beginning of the planning horizon and every time crane 3 sets down a container.

When deciding about the next container to be handled by crane 3 we consider relocating containers in I^o and receiving a container from crane 2 as options. For receiving a container from crane 2 we assume that crane 3 approaches crane 2 as fast as possible and receives the first container that crane 2 can hand over. We evaluate each of these options according to the benefit of crane 2, that is the reduction in workload of crane 2 by crane 3 carrying a container, as compared to the cost of crane 3, that is the empty travel time for crane 3 to approach the container. Each option is evaluated by benefit minus cost and we choose the option having the highest evaluation. Following

the proposed approach, the positions of containers provided for the decision making potentially deviate from the original container positions, e.g. if a container has been relocated by crane 3. Hence, during decision making we use the incumbent container positions as $\overline{o}_i = (\overline{o}_i^b, \overline{o}_i^r)$ and $\overline{d}_i = (\overline{d}_i^b, \overline{d}_i^r)$.

We aim at handing over only containers for which crane 2 can actually gain a benefit. Hence, for two consecutive containers i and i' in $\overline{n}_2$, focusing only on bay positions, we say that

- crane 2 is allowed to hand over $i \in I^i$ only if $\overline{o}_{i'}^b < \overline{d}_i^b$ holds and
- crane 3 is allowed to relocate $i' \in I^o$ only if $\overline{d}_i^b < \overline{o}_{i'}^b$ holds.

If $\overline{o}_{i'}^b \geq \overline{d}_i^b$ with $i \in I^i$, then crane 2 moves beyond $\overline{d}_i^b$ in order to pick up i' and, hence, passes $\overline{d}_i^b$. If $\overline{d}_i^b \geq \overline{o}_{i'}^b$ with $i' \in I^o$, then crane 2 moves beyond $\overline{o}_{i'}^b$ in order to deliver i and, hence, passes $\overline{o}_{i'}^b$ on its way back. Whenever a container is intermediately stored or relocated, we say that the crane carrying it begins to set it when the cranes are positioned less or equal than $p+1$ bays apart.

While carrying a container, crane 3 is prioritized in case of a conflict, while crane 2 gets prioritized otherwise. The makespan of a hereby determined schedule equals then the point of time when all containers in $\overline{n}_2$ are dropped off in their destination positions.

What now remains to detail is the movement of the cranes's trolleys. Clearly, for containers not being handed over, crane 2 positions its trolley according to the pick up request as early as possible." For container i, destined to be handed over to crane 3 we proceed as follows. Let r_c and b_c be the row and bay position of crane c at the beginning of a period. If $|\overline{d}_i^b - b_2| > |r_2 - \overline{d}_i^r| + 1$ holds, $\overline{d}_i^r$ can be reached by 2's trolley before 2 reaches $\overline{d}_i^b$. Further, it can even be moved away from $\overline{d}_i^r$ without increasing the duration to reach d_i. In such a case we decide that 2 moves the trolley closer to r_3, which potentially reduces empty row travel time of 3, before picking up the container. If $|\overline{d}_i^b - b_2| \leq |r_2 - \overline{d}_i^r| + 1$ holds, 2 moves the trolley closer to $\overline{d}_i^r$. Such a movement brings r_2 potentially away from r_3 but into a position that r_3

has to cross later on in order to store the container in d_i. Knowing the trolley position of 2 at the end of a period, we move crane 3's trolley closer to that row during the current period. After 2 has set down i, 3 moves its trolley to the positions necessary to store it in d_i. Whenever crane 3 is currently relocating a container, it moves the trolley according to d_i , if $|b_3 - d_i^b| \leq |r_3 - d_i^r|$ holds. Otherwise, it moves the trolley closer to $\overline{d}_k^r$, with k being the predecessor job of i in $\overline{n}_2$, potentially reducing the time for 2 to pick up i after setting down k. Crane 2 moves its trolley closer to r_3 in a period, if 2 has already transported k. Otherwise, it moves according to the next operation in $\overline{n}_2$.

"Let us conclusively detail how we determine $\overline{n}_2$. We do so by employing a simulated annealing approach resembling the one in Briskorn et al. [8]. We set the initial temperature T to 3 times the workload assuming that only crane 2 handles containers. In each iteration, we generate $|I^i| + |I^o|$ neighboring solutions for the current sequence $\overline{n}_2$ by randomly swapping the position of two containers. We replace the current solution with a neighboring solution if $rand(0,1) < exp(-\Delta/T)$ holds where Δ is the difference in makespans. After each iteration we set T to $0.99 \cdot T$ and repeat the procedure unless T is smaller than 0.1. Then, the container sequence with the best obtained makespan is returned. The approach is denoted by ABB.

3.4 Computational Study

The computational study presented in this section is twofold, first we present a static analysis of the performance of the algorithms and give some managerial insights into strategies where to locate the handshake area in a block in Section 3.4.1. In Section 3.4.2 we then apply these insights and employ our deterministic approaches in a rolling horizon setting and compare the performance of the algorithms in order to investigate how much we can gain from better solutions to TCSPH.

We conducted the tests on an Intel Core i7-4790 CPU with 3.6 GHz and 32

GB of RAM running Windows 7. All approaches were implemented in Java 8.

3.4.1 Static Analysis

In order to evaluate the approaches we created sets of instances with different parameter combinations. We set the size of the storage block to 30 bays with a seaside transfer bay in bay 0 and to either 6 or 10 rows. Initially, crane 2 is positioned in bay 0 while crane 3 is positioned in bay 30 and the initial trolley positions are in row 1. The time to pick up or drop off a container is $p = 3$.

For every instance $|I^i| = |I^o|$ holds and we have 5, 10, 15 or 20 containers of each type. The bay positions of containers are randomly drawn from $[1, 30]$ while row positions are drawn from $[1, 6]$ ($[1, 10]$) according to a uniform distribution. We vary the location of b^h throughout the study and detail its position throughout this section. The location of the handshake area, then consequently determines the sets of jobs for both cranes. In a real world setting it is rather unlikely that containers get stored permanently in the handshake area, hence, whenever o_i or d_i implies storing a container in b^h we change the bay position to $b^h - 1$ and $b^h + 1$ respectively. We used the instance generator from Briskorn et al. [11] in order to generate 30 container sets for each parameter combination, resulting in 240 container sets. We, then, complement each container set to ten instances of TCSPH by setting b^h to each value in $\{5, 10, 15, 20, 25\}$ and by varying the capacity for each stacking position within the handshake area. We distinguish between a low capacity setting, where the capacity of each position is randomly drawn from $\{1, 2, 3\}$ and a high capacity setting where the capacity is drawn from $\{3, 4, 5\}$. As a result, in total 2400 instances were evaluated.

Obviously, for a given container set the position of the handshake area influences the workload of cranes. Since we expect it to have a major impact on the optimum makespans of instances we have a closer look at it first. We outline the average number of jobs $|J_c|$ for a crane that result from a given

b^h and number of containers as well as the average workload W_c in Table 3.1.

		$b^h = 5$			$b^h = 10$			$b^h = 15$			$b^h = 20$			$b^h = 25$		
$\lvert I^i\rvert = \lvert I^o\rvert$	$\lvert J_1\rvert$	$\lvert J_2\rvert$	W_1	W_2	$\lvert J_2\rvert$	W_1	W_2	$\lvert J_2\rvert$	W_1	W_2	$\lvert J_2\rvert$	W_1	W_2	$\lvert J_2\rvert$	W_1	W_2
5	10	9	108	158	7	147	109	5	177	68	3	198	36	2	210	14
10	20	17	216	323	14	294	225	11	356	144	7	400	77	4	427	31
15	30	26	324	475	21	440	329	16	532	207	10	595	108	5	632	42
20	40	35	432	642	28	589	446	21	712	279	14	798	151	7	849	57

Table 3.1: Effect of b^h: average workload W_c, average number of jobs $|J_c|$

Obviously $|J_3|$ increases, the closer the handshake area is located to the seaside, since potentially more containers have to be transported across that bay in order to reach their final destination. Even though $|J_2|$ is a constant for a given set of containers, the position of the handshake area affects the workload of those jobs. Hence, the closer the handshake area is located to the seaside, the less transport distance does crane 2 have to travel, since more jobs need to be stored in b^h and the distance between b^h and bay 0 decreases. Contrarily, the distances in between the pick up and drop off position increase for crane 3.

For the B&Bs we set the time limit to 300 seconds. In case no B&B guarantees optimality for an instance we consider the largest lower bound determined by a B&B and determine the average relative gap to this lower bound for all approaches. Furthermore, we outline the average run times in seconds in Table 3.2 for a block width of 6 rows and in Table 3.3 for a block width of 10 rows.

For a given width of the storage block, it takes more time on average to determine solutions when the capacity of stacking positions is larger. This comes at no surprise since with larger capacities the solution space increases and potentially more nodes have to be evaluated. Based on the same reasoning, the average run times increase with an increasing width of the storage block, since more stacking positions are available to intermediately store containers in. For an increasing number of containers to be handled, the average gap as well as average run times increases. Next, we analyze the performance of each B&B in detail.

We observe that SEQ_{JS} is able to determine optimum solutions when con-

		Avg. relative Gap LB %					Avg. run time s				
$\|I^i\| = \|I^o\|$	Algorithm / b^h	5	10	15	20	25	5	10	15	20	25
5	SEQ_{JS}	0.03	0.00	0.00	0.00	0.00	17.23	3.60	1.14	0.02	0.02
	SIM	0.00	0.00	0.02	0.00	0.00	32.95	50.86	17.26	0.03	0.01
	SEQ_{SJ}	0.02	0.42	0.02	0.00	0.00	100.09	130.03	30.24	0.03	0.02
10	SEQ_{JS}	7.82	1.22	0.18	0.00	0.00	300.00	159.59	86.63	0.12	0.58
	SIM	2.67	1.78	0.44	0.00	0.01	197.37	203.63	92.67	1.40	10.12
	SEQ_{SJ}	0.49	0.81	0.12	0.00	0.00	127.22	146.20	61.11	0.16	0.45
15	SEQ_{JS}	9.01	5.09	1.04	0.47	0.00	300.00	268.73	200.42	41.98	0.67
	SIM	3.93	2.88	1.43	0.33	0.00	281.64	203.91	222.02	62.35	0.81
	SEQ_{SJ}	4.87	0.48	0.19	0.00	0.00	239.44	138.41	121.27	7.29	0.25
20	SEQ_{JS}	9.95	16.83	8.38	2.41	0.00	300.00	300.00	290.77	166.29	5.43
	SIM	3.83	13.74	5.13	1.61	0.05	287.57	283.11	295.45	190.00	40.02
	SEQ_{SJ}	5.97	0.77	0.19	0.01	0.00	239.71	197.68	150.84	14.16	1.18

(a) Low capacity

		Avg. relative Gap LB %					Avg. run time s				
$\|I^i\| = \|I^o\|$	Algorithm / b^h	5	10	15	20	25	5	10	15	20	25
5	SEQ_{JS}	0.03	0.00	0.00	0.00	0.00	18.71	3.71	1.14	0.02	0.02
	SIM	0.00	0.00	0.02	0.00	0.00	52.76	50.98	20.06	0.02	0.02
	SEQ_{SJ}	0.02	0.42	0.02	0.00	0.00	100.10	130.04	30.27	0.03	0.02
10	SEQ_{JS}	6.76	0.90	0.15	0.00	0.00	300.00	159.87	89.87	0.12	0.60
	SIM	3.45	1.97	0.45	0.00	0.01	203.15	203.89	93.76	2.90	10.53
	SEQ_{SJ}	1.66	1.40	0.12	0.00	0.00	146.67	147.15	64.58	0.20	0.53
15	SEQ_{JS}	7.67	3.95	0.83	0.09	0.00	300.00	270.47	184.53	21.87	0.21
	SIM	5.58	3.85	1.38	0.45	0.00	292.24	221.35	243.85	88.35	1.14
	SEQ_{SJ}	7.64	0.71	0.18	0.01	0.00	259.27	135.87	114.91	11.97	0.28
20	SEQ_{JS}	7.70	13.40	5.45	1.45	0.00	300.00	300.00	290.76	129.18	3.08
	SIM	4.98	14.12	5.32	1.97	0.22	300.00	283.84	300.00	216.61	52.55
	SEQ_{SJ}	7.78	1.54	0.39	0.04	0.00	294.46	211.49	165.16	36.24	2.22

(b) High capacity

Table 3.2: Comparison of B&Bs, storage block width: 6 rows: avg. run times in seconds, average relative gap in percent, low and high capacities

sidering 5 containers of each type for all values of b^h larger than 5. It does so significantly faster than the other approaches. For the remaining instances the picture is different and, therefore, we, first, provide an explanation for SEQ_{JS} being superior for these particular instances.

If the workload of crane 2 is significantly higher than the one of crane 3 (which is the case for $b^h \geq 10$), then in optimum schedules crane 2 is likely to have minimum empty travel distance. This is achieved if only two stacking positions in the handshake area are used and these two positions are next to each other. Typically, crane 3 can support such a schedule even if this means extra effort since its workload is significantly lower. For such a schedule, we obtain tight lower bounds in the first phase of SEQ_{JS} if the pair of

		Avg. relative Gap LB %					Avg. run time s				
$\|I^i\| = \|I^o\|$	Algorithm / b^h	5	10	15	20	25	5	10	15	20	25
5	SEQ_{JS}	0.00	0.00	0.00	0.00	0.00	14.40	0.86	2.83	0.03	0.02
	SIM	0.13	0.11	0.00	0.00	0.00	98.39	50.40	28.87	0.09	0.03
	SEQ_{SJ}	0.02	0.16	0.00	0.00	0.00	84.08	138.17	30.17	1.97	0.11
10	SEQ_{JS}	7.73	2.81	0.17	0.01	0.00	300.00	133.12	79.11	10.22	0.09
	SIM	4.12	1.81	0.68	0.06	0.00	243.03	177.85	136.01	27.22	1.57
	SEQ_{SJ}	1.85	0.52	0.22	0.00	0.00	182.39	96.27	100.32	0.23	1.27
15	SEQ_{JS}	12.73	9.32	4.58	0.03	0.04	300.00	280.29	242.21	25.67	23.08
	SIM	4.69	9.57	5.52	1.55	0.02	260.49	260.00	272.74	144.00	29.34
	SEQ_{SJ}	8.76	0.79	0.16	0.03	0.01	233.03	192.33	102.65	31.21	11.46
20	SEQ_{JS}	15.44	20.55	8.82	3.50	0.04	300.00	300.00	300.00	161.63	31.16
	SIM	7.42	17.35	9.17	5.07	0.55	300.00	300.00	290.52	262.54	76.80
	SEQ_{SJ}	11.34	0.34	0.13	0.03	0.01	290.86	188.70	138.46	44.97	21.40

(a) Low capacity

		Avg. relative Gap LB %					Avg. run time s				
$\|I^i\| = \|I^o\|$	Algorithm / b^h	5	10	15	20	25	5	10	15	20	25
5	SEQ_{JS}	0.00	0.00	0.00	0.00	0.00	15.24	0.88	3.16	0.03	0.02
	SIM	0.51	0.11	0.00	0.00	0.00	103.47	53.33	30.74	0.09	0.03
	SEQ_{SJ}	0.02	0.16	0.00	0.00	0.00	80.06	138.53	30.17	1.98	0.11
10	SEQ_{JS}	6.75	1.74	0.17	0.01	0.00	300.00	131.95	81.24	10.21	0.10
	SIM	4.07	2.02	0.86	0.09	0.00	264.85	180.23	142.09	30.68	2.39
	SEQ_{SJ}	3.39	1.20	0.23	0.00	0.00	198.64	101.84	101.08	0.37	1.34
15	SEQ_{JS}	11.43	6.12	2.43	0.03	0.04	300.00	280.29	235.72	24.84	23.11
	SIM	5.97	10.06	6.22	2.11	0.02	260.52	267.47	279.90	169.05	33.58
	SEQ_{SJ}	10.50	1.47	0.27	0.02	0.01	273.18	194.54	112.11	23.10	11.49
20	SEQ_{JS}	12.86	14.40	6.11	2.24	0.01	300.00	300.00	291.07	155.68	31.21
	SIM	8.93	17.49	9.10	6.00	0.74	300.00	300.00	290.68	271.69	86.32
	SEQ_{SJ}	13.09	2.21	1.05	0.48	0.00	300.00	210.38	194.16	92.06	16.09

(b) High capacity

Table 3.3: Comparison of B&Bs, storage block width: 10 rows: avg. run times in seconds, average relative gap in percent, low and high capacities

sequences constructed in this phase can be complemented to schedules in the second phase. For a small number of containers we often can do so and, hence, the tight lower bounds guide our search very well already in low levels of the search tree. However, the more containers need to be stored in the handshake area, that is the smaller b^h or the larger $|I^i| = |I^o|$, the more likely we cannot complement an arbitrary pair of sequences determined in the first phase of SEQ_{JS} to a feasible schedule due to limited number of stacking positions and limited capacities. Hence, the lower bounds in the first phase become less of a good guide. Note that, as opposed to the other approaches, SEQ_{JS} performes better for larger capacities which is in line with the above reasoning since we can complement an arbitrary pair of

sequences more likely to a feasible schedule. The two other approaches do not suffer from the drawback that decisions made in low levels of the search tree cannot be complemented to feasible schedules and, therefore, perform better for $b^h = 5$ or $|I^i| = |I^o| \geq 10$.

For SIM we see that it is on average the second fastest approach for $|I^i| = |I^o| = 5$ but falls behind SEQ_{SJ} for $|I^i| = |I^o| \geq 10$. While SIM provides the strongest lower bounds on a certain level of the search tree since it incorporates information about both, pair of sequences and stacking positions the search tree grows much faster on low levels. The latter affects run times more for larger $|I^i| = |I^o|$. Nevertheless, for $b^h = 5$ the advantage of having both types of information in nodes compensates for the disadvantage of a wider tree since more containers are stored intermediately in the handshake bay. Recall that on low levels of the search tree nodes in both, SEQ_{JS} and SEQ_{SJ}, lack one type of information.

Finally let us shed light on the performance of SEQ_{SJ}. One key advantage of SEQ_{SJ}, as compared to SEQ_{JS}, is that after completing the first phase of branching stacking positions for all containers are given. This allows us to determine tighter lower bounds employing the approach from Gilmore and Gomory [25], see Section 3.2.1.3. However, for instances with $|I^i| = |I^o| = 5$ SEQ_{SJ} is outperformed by SEQ_{JS} or SIM. For these relatively small number of jobs (see Table 3.1) it seems as if providing partial job sequences early on by branching decisions is more beneficial. For all other settings, with a handshake area located in bay $b^h \geq 10$, SEQ_{SJ} not only yields the shortest run times on average but also the smallest relative gap on average.

Next, we analyze the performance of the heuristics. Again, we compare the approaches with regard to average run times and average relative gap. The relative gaps obtained by the nearest neighbor heuristics are outlined in Table 3.4. The average run times of all nearest neighbor heuristics are virtually zero regardless of the parameter settings and, hence, we omit from including them into Table 3.4. The results for all $2OPT$ algorithms are shown in Table 3.5 and 3.6.

$\lvert I^i\rvert = \lvert I^o\rvert$		Storage block width: 6 rows					Storage block width: 10 rows				
	Algorithm / b^h	5	10	15	20	25	5	10	15	20	25
5	$NN(J^S)$	55.85	62.39	32.42	14.86	6.35	60.01	59.73	30.69	13.99	6.35
	$NN(J^R)$	55.83	62.37	32.40	14.86	6.34	60.14	59.47	31.39	13.43	6.30
	$SNN(J^S)$	48.55	55.07	35.96	32.77	14.01	46.22	56.12	33.39	20.85	21.20
	$SNN(J^R)$	47.17	54.12	36.18	32.74	14.01	46.16	54.94	33.23	21.48	21.21
10	$NN(J^S)$	58.93	73.09	35.66	16.44	7.47	63.11	71.78	37.41	18.16	7.43
	$NN(J^R)$	58.93	72.98	35.68	16.44	7.52	63.28	71.67	37.63	18.22	7.60
	$SNN(J^S)$	41.62	56.54	39.11	34.87	31.48	38.89	42.18	34.02	27.50	34.99
	$SNN(J^R)$	41.11	56.95	39.07	34.97	31.46	39.42	41.31	33.04	27.49	35.12
15	$NN(J^S)$	65.58	72.53	37.80	16.89	6.76	67.10	75.54	37.50	15.74	7.89
	$NN(J^R)$	65.59	72.56	37.72	16.90	6.76	67.28	75.25	37.95	15.64	7.81
	$SNN(J^S)$	47.02	53.33	40.81	37.50	44.94	40.82	50.30	35.72	34.80	41.94
	$SNN(J^R)$	46.14	55.78	39.99	37.64	44.41	43.20	47.74	34.50	34.99	41.48
20	$NN(J^S)$	64.73	77.93	39.09	17.39	7.29	68.11	78.09	40.44	19.73	7.98
	$NN(J^R)$	64.73	77.93	39.08	17.50	7.36	68.48	78.17	39.82	19.20	7.75
	$SNN(J^S)$	33.48	52.45	38.98	38.65	46.27	46.98	47.20	36.70	36.58	46.18
	$SNN(J^R)$	34.20	51.59	39.52	37.41	46.22	47.01	43.13	35.10	36.36	45.97

Table 3.4: Comparison of nearest neighbor heuristics: average relative gap in percent, 6 and 10 row block width

It comes at no surprise that all three approaches yield larger relative gaps on average than the B&Bs while run times are significantly slower. When analyzing the nearest neighbor heuristics, we observe that they are perfectly robust against different capacities. This is due to the strategy to retrieve a container immediately after it has been set down in the handshake bay when constructing a pair of sequences. Hence we outline the relative gaps in Table 3.4 without specifying the capacity setting. The gap of both approaches to the lower bound exceeds 30 percent on average for all instances. We see that $SNN(J^s)$ ($SNN(J^R)$) outperforms the simpler heuristic for $b^h \leq 10$ and that it is the other way around for $b^h \geq 20$. Remarkably, the relative gap tends to decrease (significantly in most cases) with increasing b^h. This again can be explained with the above mentioned strategy which is less restricting if less containers are intermediately stored in the handshake bay.

The $2OPT$ approaches however yield much more promising results. We see that the average relative gap decreases with increasing capacity. This is due to swaps resulting in feasible pairs of sequences more often for higher capacity. When allowing 5000 iterations, the average run times do not exceed 0.15 seconds while the gap does not exceed 10 percent on average for $|I^i| = |I^o| \leq 10$. For $|I^i| = |I^o| \geq 15$ run times increase slightly due to higher

$\lvert I^i\rvert = \lvert I^o\rvert$		Avg. relative Gap LB %					Avg. run time s				
	Algorithm / b^h	5	10	15	20	25	5	10	15	20	25
5	$2OPT(J^S, 5000)$	1.50	3.49	1.64	1.47	1.04	0.05	0.03	0.03	0.03	0.02
	$2OPT(J^R, 5000)$	1.62	3.34	1.64	1.47	1.03	0.05	0.04	0.04	0.03	0.02
	$2OPT(J^S, 50000)$	1.50	3.49	1.64	1.47	1.04	0.40	0.36	0.35	0.26	0.19
	$2OPT(J^R, 50000)$	1.62	3.34	1.64	1.47	1.03	0.41	0.36	0.35	0.26	0.19
10	$2OPT(J^S, 5000)$	7.29	7.26	2.93	1.61	1.14	0.08	0.07	0.07	0.06	0.04
	$2OPT(J^R, 5000)$	7.20	7.13	2.95	1.57	1.19	0.09	0.07	0.07	0.06	0.04
	$2OPT(J^S, 50000)$	2.79	4.44	1.73	1.58	0.98	0.73	0.72	0.70	0.58	0.42
	$2OPT(J^R, 50000)$	2.77	4.43	1.61	1.54	1.03	0.73	0.73	0.68	0.58	0.43
15	$2OPT(J^S, 5000)$	14.77	12.30	3.81	1.41	1.08	0.13	0.12	0.11	0.10	0.07
	$2OPT(J^R, 5000)$	14.19	12.43	3.81	1.36	1.08	0.14	0.11	0.11	0.10	0.07
	$2OPT(J^S, 50000)$	5.89	6.34	2.41	1.03	1.03	1.27	1.12	1.06	0.96	0.69
	$2OPT(J^R, 50000)$	6.05	6.38	1.96	1.00	1.03	1.28	1.11	1.07	0.99	0.69
20	$2OPT(J^S, 5000)$	22.07	20.19	6.08	2.26	1.10	0.20	0.17	0.17	0.14	0.13
	$2OPT(J^R, 5000)$	22.22	19.95	5.98	2.12	1.08	0.21	0.17	0.17	0.13	0.13
	$2OPT(J^S, 50000)$	9.06	8.34	2.31	1.19	0.92	1.94	1.67	1.73	1.39	1.29
	$2OPT(J^R, 50000)$	9.11	8.29	2.31	1.20	0.91	1.96	1.68	1.75	1.35	1.30

(a) Low capacity

$\lvert I^i\rvert = \lvert I^o\rvert$		Avg. relative Gap LB %					Avg. run time s				
	Algorithm / b^h	5	10	15	20	25	5	10	15	20	25
5	$2OPT(J^S, 5000)$	1.57	2.97	2.10	1.50	1.08	0.06	0.05	0.04	0.03	0.02
	$2OPT(J^R, 5000)$	1.80	3.20	2.10	1.50	1.06	0.06	0.05	0.04	0.03	0.02
	$2OPT(J^S, 50000)$	1.50	2.97	2.10	1.50	1.08	0.54	0.48	0.41	0.29	0.20
	$2OPT(J^R, 50000)$	1.72	3.18	2.10	1.50	1.06	0.55	0.48	0.42	0.29	0.20
10	$2OPT(J^S, 5000)$	6.04	7.10	2.22	0.91	1.00	0.11	0.10	0.09	0.07	0.05
	$2OPT(J^R, 5000)$	6.10	7.27	2.23	0.86	0.96	0.12	0.10	0.09	0.07	0.05
	$2OPT(J^S, 50000)$	2.80	3.70	1.70	0.87	0.99	1.01	1.01	0.87	0.67	0.46
	$2OPT(J^R, 50000)$	3.01	3.84	1.58	0.83	0.96	1.04	0.99	0.85	0.66	0.46
15	$2OPT(J^S, 5000)$	12.42	10.07	2.96	1.03	0.91	0.18	0.16	0.15	0.12	0.08
	$2OPT(J^R, 5000)$	12.06	10.31	2.86	1.01	0.91	0.18	0.15	0.15	0.12	0.08
	$2OPT(J^S, 50000)$	5.46	4.67	1.42	0.81	0.88	1.63	1.57	1.51	1.19	0.82
	$2OPT(J^R, 50000)$	5.50	4.57	1.46	0.80	0.88	1.63	1.55	1.53	1.22	0.82
20	$2OPT(J^S, 5000)$	16.69	17.55	4.92	1.87	0.99	0.25	0.23	0.21	0.17	0.14
	$2OPT(J^R, 5000)$	16.81	17.48	4.94	1.63	1.00	0.25	0.23	0.21	0.17	0.14
	$2OPT(J^S, 50000)$	8.07	7.51	1.92	1.09	0.82	2.40	2.31	2.19	1.72	1.41
	$2OPT(J^R, 50000)$	8.15	7.28	1.97	0.96	0.86	2.42	2.33	2.17	1.65	1.43

(b) High capacity

Table 3.5: Comparison of $2OPT$ approaches and storage block width of 6 rows: low and high stacking position capacity, avg. run time in seconds, average relative gap in percent

effort for feasibility checks and routing. Also, the relative gap increases. When allowing 50000 iterations, naturally the average run times increase

$\lvert I^i\rvert = \lvert I^o\rvert$		Avg. relative Gap LB %					Avg. run time s				
	Algorithm / b^h	5	10	15	20	25	5	10	15	20	25
5	$2OPT(J^S, 5000)$	2.05	5.10	2.99	1.91	1.60	0.04	0.04	0.03	0.03	0.02
	$2OPT(J^R, 5000)$	1.59	5.31	2.59	2.06	1.65	0.05	0.04	0.03	0.03	0.02
	$2OPT(J^S, 50000)$	1.59	5.06	2.99	1.91	1.60	0.36	0.37	0.35	0.25	0.20
	$2OPT(J^R, 50000)$	1.53	5.02	2.59	2.06	1.65	0.41	0.34	0.32	0.25	0.20
10	$2OPT(J^S, 5000)$	9.59	9.97	3.84	1.71	1.42	0.08	0.07	0.07	0.05	0.04
	$2OPT(J^R, 5000)$	8.51	8.27	2.73	1.64	1.34	0.08	0.07	0.07	0.06	0.05
	$2OPT(J^S, 50000)$	4.64	6.33	2.83	1.66	1.40	0.69	0.68	0.66	0.53	0.43
	$2OPT(J^R, 50000)$	4.09	5.63	2.27	1.55	1.32	0.75	0.67	0.69	0.55	0.47
15	$2OPT(J^S, 5000)$	17.74	15.64	6.06	2.25	1.35	0.12	0.11	0.11	0.09	0.07
	$2OPT(J^R, 5000)$	16.49	14.14	5.84	1.94	1.33	0.13	0.12	0.11	0.10	0.07
	$2OPT(J^S, 50000)$	7.53	7.50	3.43	1.53	1.18	1.16	1.10	1.11	0.94	0.75
	$2OPT(J^R, 50000)$	6.65	6.12	3.51	1.41	1.20	1.23	1.13	1.07	0.96	0.73
20	$2OPT(J^S, 5000)$	27.63	24.55	8.40	3.61	1.70	0.18	0.16	0.15	0.14	0.11
	$2OPT(J^R, 5000)$	25.84	23.76	8.00	3.47	1.77	0.18	0.16	0.16	0.14	0.11
	$2OPT(J^S, 50000)$	10.74	11.47	4.43	1.63	1.31	1.76	1.58	1.51	1.37	1.13
	$2OPT(J^R, 50000)$	10.96	11.34	3.88	1.59	1.38	1.83	1.59	1.55	1.42	1.11

(a) Low capacity

$\lvert I^i\rvert = \lvert I^o\rvert$		Avg. relative Gap LB %					Avg. run time s				
	Algorithm / b^h	5	10	15	20	25	5	10	15	20	25
5	$2OPT(J^S, 5000)$	2.14	4.26	2.58	1.56	1.60	0.06	0.05	0.05	0.03	0.02
	$2OPT(J^R, 5000)$	1.90	3.80	2.25	1.70	1.68	0.07	0.05	0.04	0.03	0.02
	$2OPT(J^S, 50000)$	1.89	3.96	2.58	1.56	1.60	0.53	0.48	0.44	0.28	0.21
	$2OPT(J^R, 50000)$	1.43	3.72	2.25	1.70	1.68	0.58	0.46	0.42	0.28	0.20
10	$2OPT(J^S, 5000)$	7.65	9.17	2.62	1.58	1.48	0.11	0.10	0.10	0.08	0.05
	$2OPT(J^R, 5000)$	6.60	8.53	2.44	1.54	1.44	0.12	0.10	0.10	0.08	0.05
	$2OPT(J^S, 50000)$	3.58	4.63	1.83	1.55	1.48	0.99	1.01	0.94	0.75	0.53
	$2OPT(J^R, 50000)$	3.46	4.40	1.65	1.48	1.44	1.10	1.00	0.99	0.79	0.55
15	$2OPT(J^S, 5000)$	12.37	12.89	3.54	1.81	1.25	0.17	0.17	0.16	0.12	0.09
	$2OPT(J^R, 5000)$	11.85	11.49	3.47	1.63	1.23	0.18	0.16	0.15	0.12	0.09
	$2OPT(J^S, 50000)$	5.12	5.94	2.19	1.57	1.19	1.65	1.71	1.54	1.25	0.92
	$2OPT(J^R, 50000)$	5.41	4.79	2.17	1.28	1.13	1.67	1.69	1.57	1.18	0.94
20	$2OPT(J^S, 5000)$	19.82	18.99	6.61	2.50	1.51	0.25	0.23	0.21	0.18	0.14
	$2OPT(J^R, 5000)$	18.30	18.13	5.88	2.39	1.56	0.26	0.23	0.23	0.18	0.14
	$2OPT(J^S, 50000)$	9.12	9.09	2.80	1.33	1.25	2.43	2.25	2.05	1.83	1.45
	$2OPT(J^R, 50000)$	7.69	8.51	2.73	1.65	1.32	2.58	2.34	2.10	1.84	1.46

(b) High capacity

Table 3.6: Comparison of $2OPT$ approaches and storage block width of 10 rows: low and high stacking position capacity, avg. run time in seconds, average relative gap in percent

but they do not exceed 3 seconds on average. Simultaneously, the relative gap decreases. For $b^h = 5$, $|I^i| = |I^o| = 15$ and high capacities as well

as for $b^h = 5$, $|I^i| = |I^o| = 20$, 10 rows, and high capacities, even the B&Bs are outperformed. Note, however, that such a setting is unlikely a real world scenario, since usually capacities are tight, and, as we will see later on, locating the handshake area close to the seaside is not a good choice.

Finally, we analyze the effect of the position of the handshake area on the average makespan. In Table 3.7 we outline the average makespan obtained by the B&Bs and, as the dominant heuristics, $2OPT(J^S, 50000)$ and $2OPT(J^R, 50000)$.

$\|I^i\| = \|I^o\|$		Storage block width: 6 rows										Storage block width: 10 rows									
		Low capacity					High capacity					Low capacity					High capacity				
	Algorithm / b^h	5	10	15	20	25	5	10	15	20	25	5	10	15	20	25	5	10	15	20	25
5	SEQ_{JS}	186	164	196	220	237	186	164	196	220	237	189	169	203	228	244	189	169	203	228	244
	SIM	186	164	196	220	237	186	164	196	220	237	189	169	203	228	244	190	169	203	228	244
	SEQ_{SJ}	186	165	196	220	237	186	165	196	220	237	189	169	203	228	244	189	169	203	228	244
	$2OPT(J^S, 50000)$	189	170	199	223	239	189	169	200	223	239	192	177	209	232	247	193	176	208	231	247
	$2OPT(J^R, 50000)$	189	170	199	223	239	189	170	200	223	239	192	177	208	232	247	192	175	207	231	248
10	SEQ_{JS}	395	321	385	436	467	391	320	385	436	467	403	335	394	444	477	399	331	394	444	477
	SIM	377	322	386	436	467	379	323	386	436	467	387	332	396	444	477	388	332	396	444	477
	SEQ_{SJ}	369	319	385	436	467	373	321	385	436	467	379	327	394	444	477	385	330	394	444	477
	$2OPT(J^S, 50000)$	377	331	391	442	472	377	329	391	439	472	390	346	404	451	484	387	341	400	451	484
	$2OPT(J^R, 50000)$	377	331	390	442	472	378	329	390	439	471	388	344	402	451	484	386	340	399	450	484
15	SEQ_{JS}	568	493	570	636	674	560	488	569	634	674	615	529	611	659	706	608	513	599	659	706
	SIM	539	482	572	635	674	548	487	572	636	674	571	530	617	669	706	578	533	621	673	706
	SEQ_{SJ}	543	471	565	633	674	557	472	565	633	674	593	487	585	659	706	602	491	586	659	706
	$2OPT(J^S, 50000)$	550	499	578	640	681	548	491	572	638	680	587	520	604	669	714	573	513	597	669	714
	$2OPT(J^R, 50000)$	551	499	575	639	681	549	490	572	638	680	582	513	605	668	714	575	507	597	667	713
20	SEQ_{JS}	774	729	816	866	903	759	708	794	858	903	820	767	834	892	920	800	728	813	880	920
	SIM	731	710	791	859	904	740	712	793	862	905	761	747	836	905	925	772	748	836	913	927
	SEQ_{SJ}	746	629	754	845	903	758	633	756	845	903	790	638	767	861	920	802	650	774	865	919
	$2OPT(J^S, 50000)$	769	676	770	855	911	762	671	767	854	910	787	709	800	875	932	775	694	787	872	931
	$2OPT(J^R, 50000)$	769	676	770	855	911	763	669	767	853	911	788	709	796	874	932	765	690	787	875	932

Table 3.7: Average makespan obtained by the B&Bs and selected heuristics

First, we observe that the average makespan does not considerably depend on the capacity. Second, we find that the average makespan is lowest for a value of $b^h = 10$. We find a reason by taking into account the results from Table 3.1 and observe that $\max\{W_2, W_3\}$, among different values for b^h is minimized for $b^h = 10$, which intuitively supports short makespans. We like to emphasize that this finding does not hold only for the shortest makespan obtained but also for each single approach individually. In the following, we provide more detailed insights into the effect of the very position of the handshake area.

For the not yet examined values for b^h in interval $[5, 15]$, $|I^i| = |I^o| = 5$, low capacity and a storage block width of 6 rows, we determined the average optimum makespan using SEQ_{JS} as well as the average workload of the cranes.

The results are provided in Table 3.8 and they indeed indicate that the average makespan is lower, the lower the maximum workload among cranes is. For the tested instances, the respective minimum maximum workload is $W_2 = 132$ for $b^h = 8$ and an average makespan of 162 is obtained.

b^h	5	6	7	8	9	10	11	12	13	14	15
W_1	108	116	124	132	139	147	153	159	165	171	177
W_2	158	148	138	129	120	109	101	92	84	77	68
Avg. Makespan	186	175	166	162	162	164	169	176	183	189	196

Table 3.8: Average makespan obtained by SEQ_{JS}, W_2 and W_3 for $|I^i| = |I^o| = 5$, low capacity, a block width of 6 and $b^h = 5, 6, \ldots, 15$

In a final step, we evaluate the loss of performance that comes with using a fixed handshake bay instead of allowing cranes to hand over containers flexibly. We developed ABB as a derivative of the bucket brigade algorithm presented in Briskorn et al. [8] which performed well in a one-dimensional setting with flexible handovers, see Section 3.3.2. Now, we employ the relative gap between the makespans of the schedule obtained by ABB and the best schedule obtained by a B&B as a proxy for the loss of performance. In Table 3.9 we provide the average makespan as well as average run times in seconds yielded by ABB. Furthermore, we outline the average relative gap for $b^h = 10$.

	Storage block width: 6 rows			Storage block width: 10 rows		
$\|I^i\| = \|I^o\|$	ABB	Run times s	Gap $b^h = 10$	ABB	Run times s	Gap $b^h = 10$
5	160	0.24	-2.74	166	0.27	-1.8
10	316	0.71	-0.59	326	0.76	-0.4
15	457	1.42	-2.86	483	1.46	-0.7
20	623	2.37	-0.79	644	2.40	1.0

Table 3.9: Average obtained makespan by ABB, average run times in seconds and average relative deviation in percent to best results obtained by any B&B for $b^h = 10$

We observe that the run times of ABB are relatively small which is no surprise given the findings in Briskorn et al. [8]. The relative deviation is not below -3 percent on average and the B&Bs even obtain better results for

$|I^i| = |I^o| = 20$ and a storage block width of 10 rows. Thus, we conclude that the loss in performance by having a fixed handshake bay is kept in check.

3.4.2 Rolling horizon approach

In this section we present a rolling horizon framework and evaluate the performance of SEQ_{JS}, a $2OPT$ approach and ABB within it. More specifically we employed $2OPT(J^R, 100000)$ which performed best among $2OPT$ approaches in preliminary tests. Within the framework sets of containers become known over time for scheduling. Rather than defining distinct points in time when new containers become known we start with λ available containers at the beginning and add μ new containers whenever μ containers reached their destination. This keeps the workload being added and the workload completed in balance for different approaches in an otherwise identical setting. Note that keeping such a balance is preferable since otherwise the number of containers available to the scheduling mechanism might grow significantly or cranes might be idle due to lack of workload.

Whenever new containers become known we discard all decisions made in the previous scheduling step, regarding containers that are picked up after the μth container was set down. Afterwards, we derive a schedule using $2OPT$ or SEQ_{JS}. Obviously, we have to account for the precedence relations as well as the capacities resulting from previously fixed decisions. We, then, implement this schedule until the μth container has been set down in its final position. For ABB, we proceed analogously and update the positions of containers that were re-positioned within the yard.

For the proposed approach we generated 30 instances, each with $|I^i| = |I^o| = 50$, shuffled the containers and afterwards fixed an order in which they are released. The block has a width of 6 rows, a length of 30 bays (with an additional seaside transfer bay in bay 0) and high capacities, randomly drawn from $\{3, 4, 5\}$ and p equals 3. The cranes begin in bays 0 and 30 respectively with their trolley in position 1. The container sets were again generated with the generator from Briskorn et al. [11].

We tested 4 different settings, three with $\mu = 5$ and $\lambda = 5, 10$ or 15 and one setting with $\mu = 10$ and $\lambda = 20$. We set b^h to $8, 9$ and 10 respectively and we limited the run time for SEQ_{JS} for each horizon to the minimum among 10 seconds or the time necessary to obtain a feasible solution. Additionally, we employed $2OPT$.

The results are detailed in Table 3.10. We depicted the average makespan obtained by ABB as well as for the other two approaches with different positions of the handshake area. Further, among approaches considering a handshake area, we underlined the minimum average makespan achieved and outline the average relative gap to the makespan achieved using ABB in the last column.

		$b^h = 8$		$b^h = 9$		$b^h = 10$		Avg. rel. deviation
Setting	ABB	SEQ_{JS}	$2OPT$	SEQ_{JS}	$2OPT$	SEQ_{JS}	$2OPT$	to ABB in %
$\lambda = 5$, $\mu = 5$	1946	1863	1985	1880	1986	1931	2021	-4.2
$\lambda = 10$, $\mu = 5$	1625	1728	1970	1739	1960	1804	1983	6.3
$\lambda = 15$, $\mu = 5$	1594	1721	2089	1707	2037	1754	2050	7.1
$\lambda = 20$, $\mu = 10$	1591	1724	2130	1707	2149	1742	2097	7.3

Table 3.10: Avg. makespan yielded by approaches for the rolling horizon setting, average relative deviation in percent to the makespan obtained by ABB

Let us first compare the results of $2OPT$ and SEQ_{JS}. Analogously to the results from Table 3.8, we observe that the average makespan tends to be the lowest for $b^h = 8$ or 9. Second, we observe that the average makespan obtained by $2OPT$ is significantly larger than the one obtained by SEQ_{JS}. Third, we can see that only SEQ_{JS} can benefit from having a larger number of containers available: the makespan achieved with SEQ_{JS} tends to decrease with increasing λ while the one achieved with $2OPT$ increases. Solution quality of $2OPT$ significantly diminishes with an increasing number of containers to be considered, see Tables 3.5 and 3.6, and hence we can conclude that solution quality of the B&B carries over to a dynamic setting when employed in a rolling horizon approach. One reason may be that while we can account for predetermined precedence relations and capacities in the B&B easily we encounter a vast number of infeasible sequences in $2OPT$.

Now, we analyze the makespan achieved with ABB as compared to the one achieved with SEQ_{JS}. We observe that the average makespan decreases for both approaches with increasing λ. We see, furthermore, that ABB achieves better makespans. Recall, however, that not only is ABB more flexible with regard to the handover position we also relax several constraints of TCSPH. Given these significant relaxations we (similarly to our conclusion in Section 3.4.1) consider a rather small relative gap of less than 7.5% as an indication for a rather small loss of potential when handovers are restricted to a dedicated handshake bay.

3.5 Summary

Throughout this section, we investigated the problem, namely TCSPH, of determining minimum makespan schedules for a pair of twin-cranes having to progress containers that enter and leave a yard block at the seaside under the presence of a dedicated handshake area, where containers need to be intermediately stored. We presented three branch-and-bound approaches as well as several heuristic approaches that are able to obtain schedules with a relatively small gap to a lower bound. We showed that sequential determination of storage position in the handshake area followed by sequence construction is beneficial in terms of average run times and gap. Further, we embedded the approaches into a rolling horizon framework, where the exact approaches outperform the heuristics."

Chapter 4

Conclusions and Outlook

This work focuses on scheduling rail mounted gantry cranes in container storage yards. In Section 1.3 a literature overview is provided that indicates a research gap regarding the scheduling of triple-crossover-cranes as well as twin-crane scheduling with a dedicated container handover area.
Therefore, through the course of this work, hollistic scheduling approaches for both crane setups are developed. Chapter 2 covers the scheduling of triple-crossover-cranes. First, in Section 2.1, a routing approach under the objective of makespan minimization is developed, that obtains close to optimum interference free routings in short time. Afterwards, this approach is then embedded in a holistic scheduling framework, tackling the assignment of jobs to cranes as well as the construction of job sequences by means of two branch-and-bound approaches. The scheduling problem is proven to be NP-hard, however by decomposing the decision into a job assignment and a sequencing phase during branching, optimum schedules can be obtained for instances of practical size.

Even though the complexity for the holistic scheduling approach could be settled, it remains open whether the routing problem alone is already NP-hard. In further research one could investigate whether the presented approaches can be adapted in order to account for release- and due dates or different objectives such as minimization of total completion times. Further it is unclear

at this point how the holistic scheduling approaches perform within a rolling horizon planning scheme.

Chapter 3 then focuses on the scheduling of twin-cranes in presence of a dedicated container handover area, being a area within the storage block where containers are handed over from one crane to the other in order to share workload. After proving the problem to be NP-hard, three branch-and-bound approaches, allowing to determine minimum makespan schedules while incorporating key constraints such as capacities and precedence relations, are presented. Again, decomposing the decision making into separate phases tends to be most beneficial in terms of computational performance, allowing to solve small instances up to optimality in short time. The approaches are then embedded into a rolling horizon scheduling framework where simpler planning heuristics are clearly outperformed.

It remains open, how large the influence on the achievable makespan is, when allowing container handover in any bay, while regarding stacking position capacities as well as resulting precedence relations. When applying SEQ_{JS}, one could investigate how to avoid nodes within lower levels of the search tree that cannot be resolved to feasible sequences later on. Sophisticated avoidance strategies should improve the run times of this approach such that it possibly outperforms the other approaches. Finally, the effect of mixed stacks could be assessed and compared, when storing and retrieving different types of containers within the same stacking position is allowed.

Summarizing, even though the developed approaches contribute to the literature, there still exists a research gap regarding the scheduling of cranes within the tackled settings.

Bibliography

[1] J. J. Bartholdi and D. D. Eisenstein. A production line that balances itself. *Operations Research*, 44(1):21–34, 1996.

[2] N. Boysen and M. Fliedner. Determining crane areas in intermodal transshipment yards: The yard partition problem. *European Journal of Operational Research*, 204(2):336–342, 2010.

[3] N. Boysen and K. Stephan. A survey on single crane scheduling in automated storage/retrieval systems. *European Journal of Operational Research*, 254(3):691–704, 2016.

[4] N. Boysen, D. Briskorn, and F. Meisel. A generalized classification scheme for crane scheduling with interference. *European Journal of Operational Research*, 258(1):343–357, 2017.

[5] D. Briskorn and P. Angeloudis. Scheduling co-operating stacking cranes with predetermined container sequences. *Discrete Applied Mathematics*, 201:70–85, 2016.

[6] D. Briskorn and L. Zey. Resolving interferences of triple-crossover-cranes by determining paths in networks. *Naval Research Logistics*, 65(6–7): 477–498, 2018.

[7] D. Briskorn and L. Zey. Interference aware scheduling of triple-crossover-cranes. working paper.

[8] D. Briskorn, S. Emde, and N. Boysen. Cooperative twin-crane scheduling. *Discrete Applied Mathematics*, 211:40–57, 2016.

[9] D. Briskorn, F. Jaehn, and A. Wiehl. Test data generator- version 1.08, Accessed: 2017-05-09. URL `http://www.instances.de/dfg`.

[10] D. Briskorn, F. Jaehn, and A. Wiehl. A test suite for scheduling algorithms for cranes in transshipment terminals. *OR Spectrum*, to appear.

[11] D. Briskorn, F. Jaehn, and A. Wiehl. A generator for test instances of scheduling problems concerning cranes in transshipment terminals. *OR Spectrum*, to appear.

[12] P. Brucker. An efficient algorithm for the job-shop problem with two jobs. *European Journal of Operational Research*, 40(4):353–359, 1988.

[13] H. Carlo, I. Vis, and K. Roodbergen. Seaside operations in container terminals: literature overview, trends, and research directions. *Flexible Services and Manufacturing Journal*, 27(2-3):224–262, 9 2015.

[14] H. J. Carlo and F. L. Martínez-Acevedo. Priority rules for twin automated stacking cranes that collaborate. *Computers & Industrial Engineering*, 89:23–33, 2015.

[15] H. J. Carlo, I. F. Vis, and K. J. Roodbergen. Transport operations in container terminals: Literature overview, trends, research directions and classification scheme. *European Journal of Operational Research*, 236(1):1–13, 2014.

[16] H. J. Carlo, I. F. Vis, and K. J. Roodbergen. Storage yard operations in container terminals: Literature overview, trends, and research directions. *European Journal of Operational Research*, 235(2):412–430, 2014.

[17] R. Choe, T. Park, S. M. Ok, and K. R. Ryu. Real-time scheduling for non-crossing stacking cranes in an automated container terminal. In M. A. Orgun and J. Thornton, editors, *AI 2007: Advances in Artificial Intelligence: 20th Australian Joint Conference on Artificial Intelligence, Gold Coast, Australia, December 2-6, 2007. Proceedings*, pages 625–631, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[18] R. Choe, H. Yuan, Y. Yang, and K. R. Ryu. Real-time scheduling of twin stacking cranes in an automated container terminal using a genetic algorithm. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 238–243, New York, NY, USA, 2012. ACM.

[19] C. F. Daganzo. The crane scheduling problem. *Transportation Research B*, 23:159–175, 1989.

[20] U. Dorndorf and F. Schneider. Scheduling automated triple cross-over stacking cranes in a container yard. *OR Spectrum*, 32:617–632, 2010.

[21] A. Ehleiter and F. Jaehn. Scheduling crossover cranes at container terminals during seaside peak times. *Journal of Heuristics*, pages 1–34, 2018.

[22] A. H. Gharehgozli, Y. Yu, R. de Koster, and J. T. Udding. An exact method for scheduling a yard crane. *European Journal of Operational Research*, 235(2):431–447, 2014.

[23] A. H. Gharehgozli, G. Laporte, Y. Yu, and R. de Koster. Scheduling twin yard cranes in a container block. *Transportation Science*, 49(3): 686–705, 2015.

[24] A. H. Gharehgozli, F. G. Vernooij, and N. Zaerpour. A simulation study of the performance of twin automated stacking cranes at a seaport container terminal. *European Journal of Operational Research*, 261(1): 108–128, 2017.

[25] P. C. Gilmore and R. Gomory. Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. *Operations Research*, 12(5):655–679, 1964.

[26] H. Heitmann. *Selected scheduling applications*. Schriftenreihe QM : quantitative Methoden in Forschung und Praxis 38. Hamburg : Kovač, 2015.

[27] F. Jaehn and D. Kress. Scheduling cooperative gantry cranes with seaside and landside jobs. *Discrete Applied Mathematics*, 242:53–68, 2018.

[28] M. Kellner and N. Boysen. Rmg vs. drmg: an evaluation of different crane configurations in intermodal transshipment yards. *EURO Journal on Transportation and Logistics*, 4(3):355–377, 2015.

[29] N. Kemme. *Design and Operation of Automated Container Storage Systems*. Physica-Verlag Heidelberg, 2013.

[30] J. Klaws, R. Stahlbock, and S. Voß. Container terminal yard operations – simulation of a side-loaded container block served by triple rail mounted gantry cranes. In J. W. Böse, H. Hu, C. Jahn, X. Shi, and S. Stahlbock, Robertand Voß, editors, *Computational Logistics: Second International Conference, ICCL 2011, Hamburg, Germany, September 19-22, 2011. Proceedings*, pages 243–255. Springer, Berlin Heidelberg, 2011.

[31] D. Kress, J. Dornseifer, and F. Jaehn. An exact solution approach for scheduling cooperative gantry cranes. *European Journal of Operational Research*, 273(1):82–101, 2019.

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

[33] W. Li, M. Goh, Y. Wu, M. Petering, R. de Souza, and Y. Wu. A continuous time model for multiple yard crane scheduling with last minute job arrivals. *International Journal of Production Economics*, 136(2): 332–343, 2012.

[34] S. Libbey. Rmg vs. rtg, Accessed: 2018-08-04. URL `https://www.konecranesusa.com/resources/lifting-viewpoints/rmg-vs-rtg`.

[35] W. C. Ng. Crane scheduling in container yardswith inter-crane interference. *European Journal of Operational Research*, 164:64–78, 2005.

[36] W. C. Ng and K. L. Mak. An effective heuristic for scheduling a yard crane to handle jobs with different ready times. *Engineering Optimization*, 37:867–877, 2005.

[37] J. Nossack, D. Briskorn, and E. Pesch. Container dispatching and conflict-free yard crane routing in an automated container terminal. *Transportation Science*, 52(5):1059–1076, 2018.

[38] Y. N. Sotskov and N. V. Shakhlevich. Np-hardness of shop-scheduling problems with three jobs. *Discrete Applied Mathematics*, 59:237–266, 1995.

[39] R. Stahlbock and S. Voß. Operations research at container terminals: a literature update. *OR Spectrum*, 30:1–52, 2008.

[40] D. Steenken, S. Voß, and R. Stahlbock. Container terminal operations and operations research – a classification and literature review. *OR Spectrum*, 26:3–49, 2004.

[41] UNCTAD. *Review of Maritime Transport 2018*. New York: United Nations Conference on Trade and Development, 2018.

[42] I. F. Vis. A comparative analysis of storage and retrieval equipment at a container terminal. *International Journal of Production Economics*, 103(2):680–693, 2006.

[43] L. Zey, D. Briskorn, and N. Boysen. Twin-crane scheduling during seaside workload peaks with a dedicated handshake area. working paper.