Journal of Scheduling (2022) 25:3–34
https://doi.org/10.1007/s10951-021-00710-w

Twin-crane scheduling during seaside workload peaks with a dedicated handshake area

Lennart Zey[1] · **Dirk Briskorn**[1] · **Nils Boysen**[2]

Accepted: 7 September 2021 / Published online: 15 October 2021

Abstract
To enable the efficient division of labor in container yards, many large ports apply twin cranes, two identical automated stacking cranes each dedicated to one of the transfer zones on the seaside and landside. The use of a handshake area, a bay of containers that separates the dedicated areas of the two cranes, is a simple means to avoid crane interference. Inbound containers arriving in the transfer zone of one crane and dedicated to a stacking position of the other crane's area are placed intermediately in the handshake area by the first crane and then taken over by the second crane, and vice versa for outbound containers. Existing research only evaluates simple heuristics and rule-based approaches for the coordination of twin cranes interconnected by a handshake area. For this setting, accounting for precedence constraints due to stacking containers in the handshake area, we derive exact solution procedures under a makespan minimization objective. In this way, a comprehensive computational evaluation is enabled, which benchmarks heuristics with optimal solutions and additionally compares alternative crane settings (i.e., without workload sharing and cooperation with flexible handover). We further provide insights into where to position the handshake area. Our results reveal that when planning is too simple (i.e., with a rule-based approach), optimality gaps become large, but with sophisticated optimization, the price of a simplified crane coordination via a handshake area is low.

Keywords Port operations · Container logistics · Twin cranes · Crane scheduling

1 Introduction

Economies of scale in transportation coupled with modern, more fuel-efficient engines have led to increased ship sizes. In 2017, the largest container ships surpassed the 20,000 TEU (twenty-foot equivalent unit) capacity mark, and the currently largest vessels, the MSC Gülsün and its sister ships, transport up to 23,756 TEU (Schuler, 2019). Any additional hour of berth time in a port causes tremendous opportunity costs for these mega vessels. Thus, to keep these profitable ships returning, there is fierce competition among ports to offer the best cost–performance ratio to the shipping companies. To enable efficient port processes, all subsystems, starting with the huge quay cranes at the seaside, the intermediate storage yards for containers, up to the hinterland access (e.g., toward the railway system), and the automated guided vehicles (AGVs) or yard trucks connecting these subsystems, have to contribute. These subsystems, arranged in the so-called European setup with container blocks positioned perpendicular to the quay (Carlo et al., 2014a), are depicted in Fig. 1.

This paper focuses the storage yard subsystem, which has to ensure that it is not the bottleneck of the whole process, especially during peak hours with huge vessels berthed. Note that the great impact of an efficient crane operations schedule on total performance has for instance been shown by the extensive simulation study provided by Speer and Fischer

The authors have been supported by the German Research Foundation (DFG) through the grant "Scheduling mechanisms for rail mounted gantries with respect to crane interdependencies" (BR 3873/7-1).

✉ Lennart Zey
zey@wiwi.uni-wuppertal.de
http://www.prodlog.uni-wuppertal.de/

Dirk Briskorn
briskorn@wiwi.uni-wuppertal.de
http://www.prodlog.uni-wuppertal.de/

Nils Boysen
nils.boysen@uni-jena.de
http://www.om.uni-jena.de/

[1] Bergische Universität Wuppertal Professur für BWL, insbesondere Produktion und Logistik, Gaußstraße 20, 42119 Wuppertal, Germany

[2] Friedrich-Schiller-Universität Jena Lehrstuhl für Operations Management, Carl-Zeiß-Straße 3, 07743 Jena, Germany

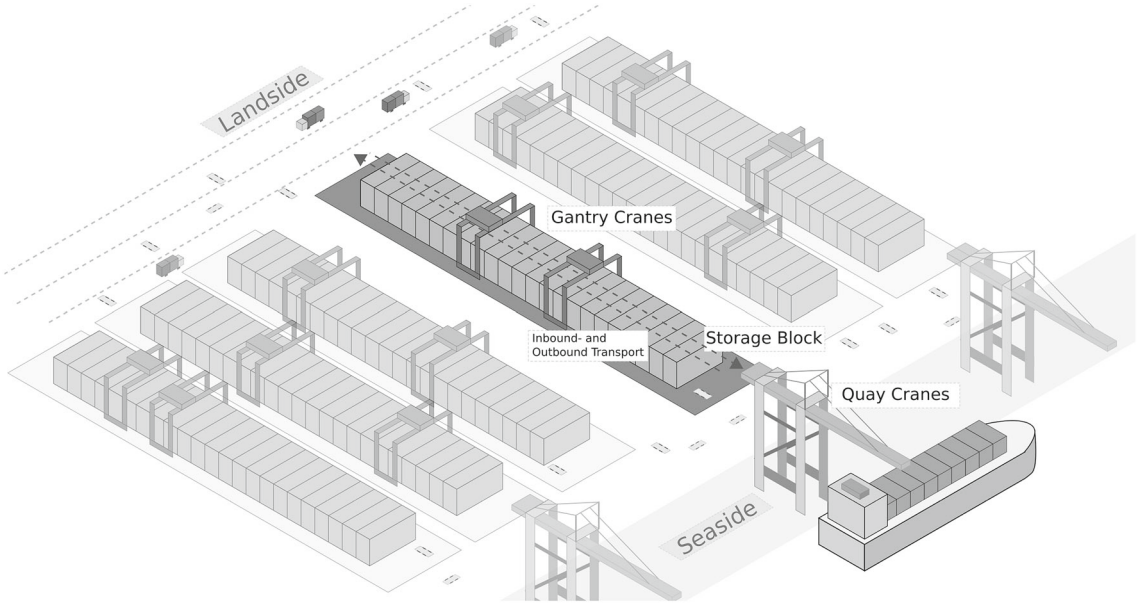

Fig. 1 Schematic layout of a container terminal

(2017). A storage yard is subdivided into multiple blocks of containers each spanned by one or multiple automated gantry cranes. Figure 1 depicts container blocks each operated by two identical twin cranes, which is a common configuration in many ports worldwide [e.g., at Euromax terminal (Rotterdam) (Carlo et al., 2014a), Port of Virginia (Norfolk) (Briskorn et al., 2016), or Yangsha Terminal (Shanghai) (Hu et al., 2016)]. In this paper, we focus on a single container block operated by twin cranes during peak hours. In this crane configuration, the two identical cranes cannot pass each other, so that each of them exclusively serves one of the access points, also known as transfer points or input/output (I/O) points, at the seaside and landside, respectively.

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 direct access to the seaside transfer point, should support the seaside crane and share some of the workload. Cooperation between 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 toward its dedicated storage position in the block, places the box in an intermediate storage position between the 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 box 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 by Briskorn et al. (2016), Jaehn and Kress (2018) and Kress et al. (2019).

Because of the interference of the twin cranes when handing over boxes in arbitrary bays, these scheduling approaches are very challenging optimization tasks, especially when executed in a real-time environment where any change in the input data requires an (almost) instantaneous plan adaptation. Furthermore, monitoring the execution of such schedules in the real world, for example to avoid collisions, is more complicated if neither crane has a dedicated operating area. Consequently, in order to reduce this type of organizational complexity, a separation of blocks into dedicated crane areas interconnected by a so-called *handshake area* (see Fig. 2) has been discussed in the literature, e.g., Gharehgozli et al. (2017), XiaoLong et al. (2019), and is reported to be applied, for instance, in the port of Rotterdam (Carlo & Vis, 2010) and in practice in general (Dell et al., 2009). A handshake area restricts the handover of containers to a fixed, predefined bay within each block. Each crane has its dedicated area, which it operates exclusively without interference, and only when entering the handshake area to hand over or take over a container is it necessary to ensure that the cranes try not to do so simultaneously. Most scheduling approaches prioritize cranes simultaneously claiming access to the handshake area by simple decision rules, for example, by giving preference to the crane with the larger remaining workload (Gharehgozli et al., 2017). Consequently, existing

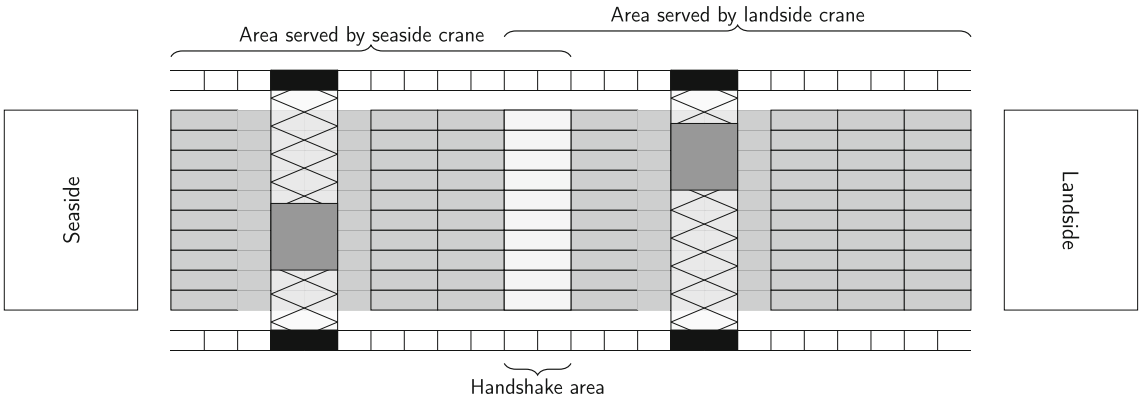

Fig. 2 Schematic layout of a container block with twin cranes and a handshake area

research on handshake areas has mainly evaluated different rules for prioritizing cranes (Carlo & Martínez-Acevedo, 2015) embedded into simple heuristics for sequencing the container moves per crane in simulation studies (Gharehgozli et al., 2017; XiaoLong et al. 2019).

The organizational complexity of container handling is reduced when restricting handovers to a handshake area, because collisions can occur only in this area. This facilitates collision avoidance and interference handling. The scheduling task is further simplified because there is much less flexibility in the choice of where to hand over a container from one crane to another. Hence, the solution space decreases. Even with a handshake area, however, scheduling cooperative twin cranes and avoiding their interference in the shared area remains a complex and challenging optimization task. This paper introduces three alternative branch and bound approaches to solve the resulting optimization problem to optimality. Once a suitable exact solution procedure is available (and proven to solve instances of practical size), for the first time, we can quantify the price for planning simply. 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 under high competitive pressure to ensure fast and reliable container handling processes, especially during seaside workload peaks, receive some decision support on how to operate their twin cranes efficiently.

The remainder of the paper is structured as follows. Section 2 reviews the related literature. Section 3 defines our optimization problem (i.e., twin crane scheduling with a handshake area) and states its computational complexity. Three alternative branch and bound procedures are introduced in Sect. 4. Our computational study presented in Sect. 5 is subdivided into three parts. First, we evaluate the computational performance of our solution algorithms (see Sect. 5.1). We also explore where to position the handshake area under seaside workload peaks (see Sect. 5.2). We then benchmark our sophisticated optimization approaches with simple rule-based solution methods taken from the literature and with any-bay handover in Sect. 5.3. Finally, Sect. 6 concludes the paper.

2 Literature review

Ocean-based container transport is the backbone of a globalized society. Therefore, it is anything but surprising that a huge body of scientific literature on container transport and port operations has accumulated during the past decades. Instead of trying to summarize this vast field, we only refer to the latest survey papers on port operations in general (Stahlbock & Voß, 2008; Steenken et al., 2004; Vis & de Koster, 2003), and on seaside operations (Bierwirth & Meisel, 2015; Carlo et al., 2015), intermediate storage in container yards (Carlo et al., 2014b), ground container transport in ports (Carlo et al., 2014c), and hinterland railway access (Boysen et al., 2013) in particular.

We concentrate our literature survey on workload sharing of multiple cooperative cranes in container yards. Cooperation is, obviously, not given if only a single gantry crane operates a container block. Crane scheduling approaches for this setting are provided for instance by Gharehgozli et al. (2014b) and Boysen and Stephan (2016). But twin cranes (without handshake area or any-bay handover) where one crane is fixedly assigned to all seaside-related container moves and the other to all landside-related moves, so that crane scheduling has to avoid interference whenever both cranes meet (e.g., see Boysen et al., 2015; Briskorn & Angeloudis, 2016; Gharehgozli et al., 2014a; Kovalyov et al., 2018), also do not cooperate in the strict sense we presuppose in this paper. Cooperation between cranes requires that there is (at least some) flexibility in the assignment of container

moves to cranes, so that by varying these assignments the division of labor between cranes can be altered. This kind of cooperation, which is especially valuable during seaside workload peaks, can be achieved by the following settings:

- *Front evasion* By extending the transfer points at both ends of a container block (and the rail tracks rail-mounted gantry cranes move on), transfer point access even for the remote twin crane is enabled (see Fig. 1(right)). This requires that the blocking crane always gives way and moves to the foremost position of the transfer point whenever the remote crane has to receive or supply a container to a subsequent position that is still part of the transfer point. This type of workload sharing has rarely been studied. The simulation study of Gharehgozli et al. (2017) where cranes are steered by simple rule-based heuristics indicates that a front evasion leads to a slightly better throughput performance than applying a handshake area. More elaborate heuristics and a mixed-integer programming (MIP) model are presented by Emde and Boysen (2014).
- *Cross-over cranes* Instead of applying two identical-sized twin cranes, some terminals apply two different-sized cranes, for example, CTA Altenwerder terminal (Hamburg), or even triple (i.e., two small twins spanned by one large crane) cross-over cranes, such as CTB Burchardkai terminal (Hamburg) (Kemme, 2012). A crossing of different-sized cranes is possible if the trolley of the larger crane is moved to the crossing position located at the side of the large crane, beyond the profile of the small one. In this way, both transfer areas can be reached by multiple cranes, and workload sharing is enabled. A simulation study with simple rule-based approaches comparing these different crane configurations is provided by Kemme (2012). More sophisticated optimization approaches coordinating two cross-over cranes are, for instance, introduced by Briskorn (2021), Briskorn and Angeloudis (2016) and Nossack et al. (2018). Scheduling procedures for triple cross-over cranes are provided by Briskorn and Zey (2018) and Dorndorf and Schneider (2010).
- *Any-bay handover* Workload sharing between twin cranes can be enabled by a preemptive container processing, which means that one crane receives a container at its dedicated access point and moves the box to an intermediate storage position where the other crane takes over. Any-bay handover allows intermediate storage wherever empty stacking space in the block can be found. Thus, optimization procedures scheduling a preemptive container processing of cooperative twin cranes must additionally decide on appropriate intermediate storage positions or alternatively whether a single crane should execute the move un-preempted. A heuristic procedure based on the famous bucket-brigade protocol (Bartholdi & Eisenstein, 1996) is provided by Briskorn et al. (2016). Their approach is extended by Jaehn and Kress (2018). They consider the case where in addition to a given sequence of seaside-related container moves, a sequence of landside-related operations occurs. An exact solution procedure for this optimization task is introduced by Kress et al. (2019).
- *Handshake area* In this paper, we only allow intermediate storage of containers processed in a preemptive mode in a dedicated area, the handshake area. Carlo and Martinez-Acevedo (2015) consider 14 different priority rules for avoiding interference of cranes in the handshake area if both cranes claim access simultaneously. They compare these rules with a branch and bound procedure and identify some rules that lead to small optimality gaps well below 5%. Their study, however, assumes given sequences of container moves, which removes much of the complexity from the scheduling task. Deciding on the sequences of container moves is integrated into the solution approaches of Gharehgozli et al. (2017) and XiaoLong et al. (2019). Both apply a rule-based avoidance of crane interference in the handshake area and integrate them into straightforward heuristics determining the sequences of container moves. The optimality gaps of these approaches are not quantified.

This paper is the first to derive exact solution approaches for scheduling the cooperation of twin cranes with a handshake area. The finding of our own literature search is supported by the recent survey paper on crane scheduling with interference by Boysen et al. (2017), who also establish a gap in the literature here.

Note that another lever to equally distribute the workload between cranes is the storage assignment decision, which either assigns each inbound container an initial storage position in the block after arrival or reassigns a container during restacking, typically executed during off-peak hours. By assigning storage positions closer to either the seaside or landside, the workload division between twin cranes is affected. A survey paper on container stacking and the resulting decision problems is provided by Lehnfeld and Knust (2014). We assume that the dedicated stacking positions of all containers are fixed and given. Only the decision on the intermediate stacking positions in the rows of the handshake area is part of our optimization problem. A handshake area confines intermediate container storage to a restricted area, so that reshuffles may become necessary if the other crane does not remove some boxes early enough. Therefore, we also integrate the storage decision within the handshake area into our optimization task. Such an integrated approach of stacking and crane scheduling was recently introduced

by Galle et al. (2018), although only in a single-crane setting.

3 Problem description

For defining our twin crane scheduling problem (TCSP) in the 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 (see Fig. 2). The slots in the first dimension are passed by the complete crane body, and we refer to them as bays $b = 1, \ldots, B$. The slots in the second dimension, passed by a crane's trolley and container spreader, 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 multiple slots aligned at the smallest bay. They are located in positions $(0, 1), \ldots, (0, R)$ and determine the seaside access point. 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 realized 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), $r = 1, \ldots, R$, of the handshake area has a free capacity C^h_r defined by 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 $H_1 = \{(b^h, 1), (b^h, 3), \ldots, \left(b^h, 2\lceil R/2 \rceil - 1\right)\}$ and $H_2 = \{(b^h, 2), (b^h, 4), \ldots, \left(b^h, 2\lfloor R/2 \rfloor\right)\}$ being used exclusively by cranes 1 and 2 for dropping off a container.

We only consider moves to and from the seaside during workload peaks caused by one or multiple berthing vessels, such that we have two sets of container moves, *inbound containers* I^i and *outbound containers* I^o, all being available at the beginning of the planning horizon. Note that crane scheduling is typically executed under a rolling planning horizon, so this assumption does not mean that we consider an unrealistically long time span. 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 is transported to destination position $d_i = (d_i^b, d_i^r)$, where it is dropped off. For an inbound container i, the origin position o_i is located at the seaside access point, such that it can be defined 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 its destination position at the seaside access point, i.e., in $(0, d_i^r)$.

Two identical gantry cranes operate on the yard block, each moving with the whole crane body along the bays of the first dimension. To reach a 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 containers at a specific slot. We refer to the seaside crane and landside crane as crane 1 and crane 2 and assume that the 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 1 (2) has to be located in a bay $b \leq b^h - 1$ ($b \geq b^h + 1$) while crane 2 (1) operates in b^h.

Because of the separate areas in which the two 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_1. It consists of two *operations*, the pickup operation in position $\hat{o}_{j(i)} = (\hat{o}^b_{j(i)}, \hat{o}^r_{j(i)}) = o_i$ and the drop-off operation 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 $j^1(i) \in J_1$ and $j^2(i) \in J_2$ ($j^1(i) \in J_2$ and $j^2(i) \in J_1$), 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 $\hat{d}_{j^1(i)} = (b^h, \hat{d}^r_{j^1(i)})$, $\hat{d}^r_{j^1(i)} \in H_1$ ($\hat{d}^r_{j^1(i)} \in H_2$). 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 but is part of the optimization.

At the beginning of the planning horizon, crane $c \in \{1, 2\}$ is located in $o_c^0 = (o_c^{0,b}, o_c^{r,0})$. 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 $\max\left\{|b - b'|, |r - r'|\right\}$ periods. The spreader can be lowered only after the gantry and trolley are in their intended position, and it has to be fully up before the 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 Sect. 1, i.e., given handshake area and container movement related to the seaside only), which need further justification.

(a) The partition of the handshake area into two subsets of stacking positions to be exclusively used for either inbound or outbound containers reduces the computational and organizational burden. While it decreases the flexibility in where to intermediately stack containers, deadlock prevention in particular becomes much easier. We exclude, furthermore, the possibility of multiple alternative handshake bays the cranes may choose from, e.g., depending on the current workload. Also, mixed policies are excluded, where, for instance, most of the time a handshake area is applied, but exceptionally a direct delivery into the other cranes area is allowed. We leave these extensions to future research.

(b) The time for moving a crane body by one bay and a trolley by one row is assumed to be identical. In the current crane generation, the crane gantry is much faster than the trolley. This, however, is counterbalanced by the larger distance gantries have to cover when moving along the length of a container of a bay (compared to the much shorter container width passed by the trolley). Given the crane speed specifications by Briskorn et al. (2019), for instance, the time difference for crane travel along both dimensions of a TEU container is less than 0.5 seconds. Compared to the much longer pickup and drop-off times, which according to Briskorn et al. (2019) range between 20 and 40 seconds, the time difference is thus negligible, and this assumption is often made in crane scheduling research (see Kellner & Boysen, 2015; Boysen et al., 2017; Ehleiter & Jaehn, 2018).

(c) The spreader can be lifted or lowered only while the gantry and trolley stand still, and vice versa, the latter two can start moving only 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 is seldom used in practice, because block utilization is usually high and crowded bays can only be passed with a lifted spreader (Ehleiter & Jaehn, 2018). Consequently, this assumption is often applied in crane scheduling research (Jaehn & Kress, 2018; Kress et al., 2019).

(d) The duration of pickup and drop-off may be position-dependent, because positions may differ in their current stacking heights. However, large parts 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 parts do not depend on the very positions and their stacking heights, real-world differences are small, and setting pickup and drop-off time p constant seems a pardonable simplification (see also Boysen & Fliedner, 2010).

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. Note that periods refer to parts of the planning horizon but do not imply a discretization of the planning horizon. 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 handover between seaside and landside crane, we have to decide in which row it is intermediately stored in the handshake area. Thus, we define the stacking positions for the respective jobs in J_1 and J_2. Note that tracking the row of each container not only allows us to reduce vertical crane travel across the bays, but is also the prerequisite to track whether a box is actually available at a certain point in time or currently blocked by another container.
2. We have to determine a sequence of jobs in J_2 implying the order in which the landside crane executes the corresponding transport jobs.
3. We have to determine a sequence of transport jobs in J_1 implying the order in which the seaside crane executes the corresponding transport jobs.
4. We have to coordinate the prioritization of pickups and drop-offs in the handshake area, because no two such operations can be carried out in parallel, due to crane interference. Such a prioritization sequence has to be consistent with the sequences in points 2 and 3; that is, the sequences in points 2 and 3 imply the order of operations 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 container is 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 at the same time.

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. This objective frees the cranes as early as possible from the current workload, so that they are readily available to process the next set of container moves of the subsequent planning run. Furthermore, this objective also tends to reduce energy consumption (Speer & Fischer, 2017). Note that He et al. (2015) benchmark time-efficient and energy-efficient schedules and their impact on either objective. Their computational study, which however treats another yard crane setting, concludes that the two objectives are strongly related. Further research is certainly required here. We, however, follow the majority of papers—see the review papers of Boysen et al. (2017),

Carlo et al. (2014a)—and target an efficient crane utilization enabled by the minimum makespan objective. But there are also alternative objectives, such as minimizing the lateness of jobs in relation to predetermined handover times to external transport devices (e.g., AGVs). We focus on the makespan, but would like to emphasize that extending crane scheduling with a handshake area to other objectives is a valid task for future research. We can assume that each crane proceeds along its sequence of operations (see points 2 and 3) as rapidly as possible (given the travel times between each pair of positions), which may be delayed only by interference from the other crane. That is, crane 1 (2) may have to wait for crane 2 (1) to complete one or more operations in the handshake area before proceeding to the handshake area itself. Moreover, it may be necessary to leave the handshake area between two operations in order to prioritize the other crane. However, these two types of delay are determined by the prioritization sequence of decision 4. Thus, once all four parts of the decision are made, we can easily determine the minimum makespan. The TCSP is to make our four-part decision such that an overall minimum makespan can be achieved. We provide a MIP model representing the TCSP in Appendix C.

Theorem 1 *The TCSP 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. (2014b) instead. Note that the problem setting of Gharehgozli et al. (2014b) assumes pickup and drop-off time $p = 0$, considers only a single crane, and addresses container movement via both access points. However, our setting with $I^{i,l} = I^{o,l} = \emptyset$ renders crane 2 irrelevant, which leaves us with crane 1 only. Furthermore, Gharehgozli et al. (2014b) show (although they do not emphasize it) that their problem is NP-hard even if only a single access point on one side of the block is 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 operations of both cranes with respect to both physical and temporal interference in the handshake area. Obviously, these issues do not simplify the problem setting.

Note that the proof of Gharehgozli et al. (2014b) relies on the number of rows being part of the input of the problem. Clearly, in the real world, the number of rows in blocks is rather small. In the following, we thus show that TCSP is strongly NP-hard even for $R = 2$.

Theorem 2 *The TCSP is strongly NP-hard even if* $R = 2$, $p = 0$, *and* $C_1^h = C_2^h > 0$ *is fixed.*

Proof See Appendix A.

Note that Theorem 2 covers the cases where capacity in the handshake bay is not tight or stacking is not allowed due to $C_1^h = C_2^h = 1$.

4 Branch and bound procedures

We present three branch and bound (B&B) approaches in order to solve our TCSP. These approaches determine 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. Decision part 4, i.e., the prioritization of pickups and drop-offs in the handshake area, is determined by a polynomial-time routing algorithm.

In the B&B approach of Sect. 4.1, we simultaneously consider the job sequences and the possible stacking positions of handshake operations. The approach in Sect. 4.2.1 proceeds by determining job sequences completely before deciding stacking positions, while the B&B approach in Sect. 4.2.2 determines these decisions in reverse.

4.1 Simultaneous sequencing and determination of stacking positions

In this section, we propose a B&B algorithm which simultaneously constructs job sequences and determines stacking positions (parts 1 to 3 of the decision) by branching. As we will show in Sect. 4.1.1, taking these parts of the decision partially implies part 4 of the decision, namely the prioritization within the handshake area. However, some portion of part 4 remains to be taken. We introduce a strongly polynomial routing approach that determines the optimal sequence of operations in the handshake area (part 4 of the decision) for given sequences of operations for both cranes in Sect. 4.1.2. Finally, in Sects. 4.1.3 and 4.1.4, we describe how we determine lower and upper bounds, respectively. An overview of the approach is given in Algorithm 1.

4.1.1 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, $c \in \{1, 2\}$, 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 operation in the handshake area, we also determine the stacking position of the container in b^h. For containers that have to cross the handshake area, it suffices to explicitly determine 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 had been stored by the other crane, we implicitly determine the stacking position for both jobs. Consequently, our search tree might reach a depth of $|J_1| + |J_2|$, and each node may 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 regarding how the remaining

Algorithm 1 Simultaneous sequencing and determination of stacking positions

initialize root node with empty job sequences n_1 and n_2 and no container in $I^{i,l} \cup I^{o,l}$ having a stack assigned; determine initial upper bound
while there are nodes in the search tree not yet considered **do**
  select node r
  for each crane c and job j associated with container i of c not in n_c in r **do**
    if $i \in I^{i,s} \cup I^{o,s}$ **then**
      create a child node (append j to n_1) observing Rule 4 (see Sect. 4.1.1)
    else if i is assigned to stack s in b **then**
      create a child node (append j to n_c) observing Rules 1 to 4 (see Sect. 4.1.1)
    else
      for each stack $s \in H_c$ **do**
        create a child node (assign i to s and append j to n_c) observing Rules 1 to 4
      end for
    end if
  end for
  for each child node s **do**
    determine upper bound (see Sect. 4.1.4) and update global upper bound
    determine lower bound (see Sect. 4.1.3) and add s to the search tree as appropriate
  end for
end while

fourth part is to be taken. In the following, we distinguish between the order of operations 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 *With parts 1 to 3 of the decision taken, a sequence of operations is implied for each pair of containers intermediately stored in the same stacking position.*

Proof See Appendix A.

In the following, it will be handier to talk about precedence relations between jobs which we interpret with regard to operations 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, $c \in \{1, 2\}$, and j precedes j' in n_c, which means that the drop-off operation of j has to be conducted before the pickup operation of j' or
- j and j' are jobs in different job sequences, have operations in the same stacking position, and among these operations, j's operation has to be conducted first according to Lemma 1.

In the first case, j is conducted completely before the first operation of j' begins. In the second case, only the two operations within the stacking position are affected by the precedence relation. The other operations of j and j' having a position outside of the handshake area 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 operations.

While Lemma 1 states that there is a sequence of operations for each pair of containers, the entirety of sequences of operations 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 operations 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 whether or not the capacity of the position is violated. 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 [i.e., 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, $c \in \{1, 2\}$, that we consider in a node. We say that the jobs in $N_c \subseteq J_c$ are the jobs that are handled only by crane c. We introduce N_2 simply for notational convenience (note that $N_2 = \emptyset$). Further, the jobs in $S_c \subseteq J_c$ are the jobs that imply the storage of a container in the handshake area, while $R_c \subseteq J_c$ are the retrieval jobs. We determine stacking positions by branching, and hence, the origin position of some retrieval jobs and the destination position of some storage jobs may not yet have been determined 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 $S_c^u = S_c \setminus S_c^k$ ($R_c^u = R_c \setminus R_c^k$) have 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 operations for stacking positions are built from start to end as well. We first introduce the rules and afterwards show that we actually reach our goal.

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 $3-c$.*

This rule allows us 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, because 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_{3-c} the corresponding retrieval job $j^2(i')$ is in n_c.*

This rule is also in line with our goal, because a container can be retrieved from the handover bay only if it is the top container at that 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 are 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 are stored in the stacking position and that only $j^2(d)$ is in n_{3-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_{3-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 operations in that stack from start to end. Hence, Rule 2 demands that between $j^2(e)$ 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_{3-c}, such that it also cannot be appended according to Rule 1. Consequently, when resolving the stacking position sequence, e and f are stored such that container f indeed is the top container for the given sequence pair. Moreover, $j^2(d)$ must have been appended to $n_{c'}$ during branching before $j^1(e)$, and $j^1(f)$ were 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 operations in stacking positions.

Lemma 2 *When appending jobs according to Rules 1 and 2, the order in which operations within the same stacking position are carried out is exactly the order in which the respective jobs are appended to the job sequences.*

Proof See Appendix A.

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 operation 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 cannot retrieve the lower container before the upper container has been retrieved. We incorporate both aspects when determining bounds, as is detailed in Sect. 4.1.3.

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

The preceding rules ensure 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 See Appendix A.

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 restricts the 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 of them we

- append a job from N_1 not in n_1 to n_1 that, consequently, is not related to a stacking position,
- append a job from R_c^k not in n_c to $n_c, c \in \{1, 2\}$, according to Rules 1 and 2, and
- append a job from $S_c^u, c \in \{1, 2\}$ to n_c, storing a container in every stacking position from H_c as long as Rule 3 is fulfilled.

However, if we apply the branching scheme as described above, we potentially create duplicate nodes in the search tree; for example, if we first append a job from N_1 to n_1 and afterwards append a job from S_2 to n_2, 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 has been appended to n_2, we can append a job to n_1 only if both jobs imply operating in the same stacking position.*

We will show that applying Rules 1 to 4 indeed allows us to construct every feasible pair of sequences. Furthermore, these rules ensure that no two different nodes in the search tree represent 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_1, n_2).

Theorem 3 *For each pair of sequences (n_1, n_2), there is exactly one σ following Rules 1 to 4 yielding it.*

Proof See Appendix A. □

The above enables us to fully specify a node in the search tree as the pair of (partial) sequences (n_1, n_2), which allows us to derive all information necessary for applying the branching scheme. However, the branching scheme only partially covers part 4 of the decision, namely the sequencing of operations in the same stacking position. Hence, it remains for us to determine the sequences of operations in different stacking positions and processed by different cranes. We propose a strongly polynomial routing approach in Sect. 4.1.2 and incorporate this approach in Sect. 4.1.3 when determining bounds. Consequently, a leaf in the search tree represents a feasible solution for TCSP. We denote this B&B approach as SIM throughout the computational study presented in Sect. 5.

We branch by following a best-first search based on the lower bounds presented in Sect. 4.1.3. Among nodes having the same lower bound, we select the one on the largest level in the search tree. To avoid out-of-memory errors, we deviate from the best-first-search policy whenever the size of the search tree exceeds 150,000 nodes. Then, we branch only the best node 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 depth-first search and keeps the number of nodes in the tree relatively stable. We provide an initial upper bound on the makespan and upper bounds for each node by applying a straightforward heuristic presented in Sect. 4.1.4.

4.1.2 Routing

In this section, we describe how the routing of cranes is determined. For two (not necessarily complete) sequences of transport jobs n_1 and n_2, we can determine the routing with minimum makespan by means of a dynamic program (DP) resembling the one in Briskorn and Angeloudis (2016) for scheduling twin cranes with given job sequences but without workload sharing. Here, we consider the sequence of lifting or releasing operations rather than the sequence of jobs. We number the operations 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 operations in the handshake area but in different stacking positions. This is sufficient to deduce an accurate routing for both cranes, since all other operations can be conducted by the cranes independently and, thus, without any waiting time or detours.

Within our DP, we have a state s for each conflicting pair of operations within the handshake area. Such a state is specified by (c, f_1, f_2), with f_1 and f_2 being the two conflicting operations of cranes 1 and 2 and c being the crane that is prioritized with respect to this conflict. It implies that crane c has just finished operation f_c and crane $3 - c$ is positioned right next to the handshake area, with its trolley in the position according to f_{3-c}. Furthermore, we have an initial state s^i defining the cranes' initial positions and a final state s^f defining crane positions just after having conducted their very last operation in n_c.

Naturally, we can restrict ourselves to only a subset of states when determining a routing, because of existing precedence relations. Hence, for two conflicting operations f_c and f_{3-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 $s = (c, f_1, f_2)$ to another state $s' = (c', f_1', f_2')$ then implies that

- crane c' conducts all operations not yet conducted in $[f_{c'}, \ldots, f_{c'}']$ as early as possible without waiting, starting from its implied position in s, and
- if $f_{3-c'}$ is $f_{3-c'}'$, crane $3 - c'$ simply stays in the bay next to b^h. Otherwise it processes all operations not yet conducted in $[f_{3-c'}, \ldots, f_{3-c'} - 1]$ without waiting, moves

to the bay next to b^h, and positions the trolley according to $f'_{3-c'}$.

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 $f_{3-c} \leq f'_{3-c}$ holds, and
- assuming that cranes 1 and 2 process n_c and n_2 from s onward and are interrupted only by waiting due to precedence constraints, they would be present in b^h simultaneously for the first time when conducting f'_1 and f'_2.

The first requirement simply states that no crane c goes backward in n_c, and that the crane given priority with respect to the second conflict makes actual progress. The second requirement states that the conflict between f'_1 and f'_2 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 crane interference is as small as possible without delaying the actual operation.

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 in the absence of a decision about prioritization with respect to f'_1 and f'_2, it is implied that crane $3 - c'$ can indeed move to the bay next to b^h and position its trolley according to $f'_{3-c'}$ while c' is conducting the operation.

The duration $t(s, s')$ associated with transition (s, s') is the time span required by crane c' to conduct all operations not yet conducted 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 $t(s') = \min\left\{t(s) + t(s, s') | s \in P(s')\right\}$, with $P(s')$ being the set of states from which a transition to s' exists.

This leads to the following runtime complexity of our DP. We have $O((|I^{i,l}| + |I^{o,l}|)^2)$ potential states and transitions. We can determine whether a transition exists (and compute the corresponding duration) in $O(|I^{i,l}| + |I^{o,l}|)$ steps by iteratively checking whether the respective pairs of operations are in conflict with each other. Hence, we obtain a complexity of $O((|I^{i,l}| + |I^{o,l}|)^3)$.

4.1.3 Lower bounds

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, namely a lower bound on the lengths of the partial sequences and the duration necessary to conduct the remaining jobs that are not yet in the sequences. In the following, we detail how we account for the second part and how we then use the DP approach presented in Sect. 4.1.2 to derive a lower bound for each node.

Algorithm 2 The routing algorithm

```
add initial state s^i to the set of states
while there are unhandled states to consider do
    select the next (e.g., in lexigographical order) unhandled state s = (c, f_1, f_2)
    let both cranes process n_1 and n_2 based on s
    if both cranes interfere due to operations f'_1 and f'_2 then
        add (or update, if appropriate) (1, f'_1, f'_2) and (2, f'_1, f'_2)
    else
        add (or update, if appropriate) s^f if appropriate
    end if
end while
```

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 necessary time 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 origin $\hat{o}_k$ to destination $\hat{d}_k$ plus the time necessary to conduct both operations,
2. the empty travel that can occur between dropping off a container in its storage position and picking up the next container in its origin position,
3. waiting times due to precedence relations between jobs, and
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 $2 \cdot p + \max\{|\hat{o}^b_k - \hat{d}^b_k|, r_k\}$ on the workload w_k for each job $k \in A_c$ with

$$r_k = \begin{cases} |\hat{o}^r_k - \hat{d}^r_k| & \text{for } k \in N_c \cup S^k_c \cup R^k_c \\ 1 & \text{else.} \end{cases} \tag{1}$$

Here, r_k is a lower bound on the time necessary for the trolley to reach the row where the container is dropped off after picking it up. Note that for jobs in $S^u_c \cup R^u_c$, the row in the handshake area is not yet determined. Thus, we have to assume that the closest row is chosen ultimately. Further note that in TCSP, we do not have reshuffling jobs, such that $|\hat{o}^r_k - \hat{d}^r_k|$ equals at least one. 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 unavoidable empty travel time to process the jobs in A_c.

- We consider a bipartite graph where nodes in the first set correspond to drop-off operation of jobs in A_c or the last job in n_c, and nodes in the second set correspond to pickup operations of jobs in A_c or a dummy end job. An edge between a drop-off operation k and a pickup operation k' represents k' being carried out immediately after k and, therefore, implies a lower bound (similar to the one in (1)) on the empty travel duration. This lower bound is represented by the edge's weight. Note that n_c and n_{3-c} might imply precedence relations between jobs in A_c or between jobs in A_{3-c}. If pickup operation $k \in A_c$ is a predecessor of drop-off operation $k' \in A_c$, then we can drop the edge connecting k' and k from consideration. Similar to the approach of Gharehgozli et al. (2014b), 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 (1964) in order to determine a sequence for the jobs in A_c with minimum empty travel time. Gilmore and Gomory (1964) 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 $\hat{o}^b_k$, with $k \in A_c$) that the machine needs to be set up to in order to process the job. After finishing j, the machine is left in state E_j (corresponding to $\hat{d}^b_k$ 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 related to the last job in either n_c or o^0_c) and a predetermined ending state that it has to reach after finishing all jobs. Gilmore and Gomory (1964) developed an optimal 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 was applied by Briskorn and Zey (2020) 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 can then straightforwardly apply the approach by Gilmore and Gomory (1964) in $O(|A_c|^2)$ time.

Let lb^e_c be the larger of the lower bounds for empty travel for crane c and A_c. A lower bound for the time span necessary to conduct all jobs in A_c is then given as $lb^e_c + W_c$. We now integrate this lower bound with the routing for partial sequences n_1 and n_2. We apply a slight adaptation of the DP approach presented in Sect. 4.1.2 for routing (n_1, n_2) aiming at a minimum makespan assuming that crane c, $c \in \{1, 2\}$ completes its last job $lb^e_c + W_c$ 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.

4.1.4 Upper bounds

We apply an approach loosely based on the work of Gharehgozli et al. (2017) to determine upper bounds. First, for each container that has no stacking position assigned yet we choose one that minimizes the laden travel duration associated with either the storage job or the retrieval job. In case of a tie, we choose the one closest to the middle row ($R/2$). By storing the containers closely together, we aim to reduce the time necessary to adjust the trolley between consecutive operations in b^h.

Now, for this set of jobs, let c be the crane having the larger lower bound on the workload, that is, $W_c \geq W_{3-c}$ (with an arbitrary tiebreaker).

We determine sequences with minimum empty travel time with respect to either bay distances or row distances using the approach by Gilmore and Gomory (1964). Between the two 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. We then construct the job sequence of crane $3 - c$ using a nearest-neighbor approach. We require containers assigned to the same stacking position to appear in the same order in both job sequences in increasing likelihood of the pairs of sequences being feasible. If we obtain a feasible pair of job sequences, the remaining fourth part of the decision is then determined by the routing approach from Sect. 4.1.2.

4.2 Non-simultaneous sequencing and determining of stacking positions

In this section, we present two alternative B&B approaches where we decide on the job sequences of cranes and stacking positions of containers sequentially. Since most of the ideas presented in Sect. 4.1 carry over in a straightforward manner, we focus on highlighting the differences in the following. The motivation for decoupling the two decisions is to keep the width of the search tree small while achieving lower bounds that are similarly tight. Upper bound determination and node ordering is performed analogously to Sect. 4.1. The approaches are illustrated in Algorithms 3 and 4

Algorithm 3 Determining job sequences first

```
initialize root node with empty job sequences n_1 and n_2 and no container in I^{i,l} ∪ I^{o,l} having a stack assigned; determine initial upper bound
while there are nodes in the search tree not yet considered do
    select node r
    if r is on a level smaller |J_1| + |J_2| (n_1 or n_2 is not complete) then
        for each crane c and job j of c not in n_c in r do
            create a child node (append j to n_c) observing Rules 1 (see Sect. 4.1.1) and 5
        end for
    else if r is on a level smaller |J_1| + |J_2| + |I^{i,l}| (some containers have no stack assigned) then
        for each stack s ∈ H_1 do
            create a child node (assign first container i ∈ I^{i,l} in n_1 with no stack assigned to s)
        end for
    else
        for each stack s ∈ H_2 do
            create a child node (assign first container i ∈ I^{o,l} in n_2 with no stack assigned to s)
        end for
    end if
    for each child node s with feasible (partial) job sequences do
        determine an upper bound (see Sect. 4.1.4)
        determine a lower bound (see Sect. 4.1.3) and add s to the search tree as appropriate
    end for
end while
```

4.2.1 Determining job sequences first

In this section, we present a B&B algorithm where we decide on the sequences of jobs of each crane on the first levels of the search tree, and then determine the stacking positions of containers on the last levels. The prioritization of jobs in the handshake area (part 4 of the decision) is determined as described in Sect. 4.1.2.

We append one job to n_1 or n_2 on the first $|J_1| + |J_2|$ levels of the search tree. After having determined the order of all jobs in the respective sequence, we next determine the stacking position of each intermediately stored container. Thus, the search tree has a depth of $|J_1|+|J_2|+|I^{o,l}|+|I^{i,l}|$. On levels $1, \ldots, |J_1| + |J_2|$, we have up to $|J_1| + |J_2|$ child nodes; on larger levels we have up to $\lceil R/2 \rceil$ successors.

On each of the first $|J_1| + |J_2|$ 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 Sect. 4.1.1. Hence, we allow a retrieval job to be appended to n_c only if the corresponding storage job is already in n_{3-c}. Based on Rule 1 only, we can formulate a similar lemma analogous to Lemma 3 and show that the sequences obtained contain only acyclic precedence relations. Rule 1, again, restricts the set of jobs to be appended, such that on levels $1, \ldots, |J_1| + |J_2| - 1$ we create child nodes by

- appending a job from N_1 not in n_1 to n_1,
- appending a job from S_c not in n_c to n_c, $c \in \{1, 2\}$, and
- appending a job from R_c not in n_c to n_c, $c \in \{1, 2\}$, according to Rule 1.

On levels $|J_1|+|J_2|, \ldots, |J_1|+|J_2|+|I^{o,l}|+|I^{i,l}|-1$, we determine the stacking positions of containers in $I^{o,l}$ and $I^{i,l}$. On levels $|J_1|+|J_2|, \ldots, |I^{i,l}|-1$, we construct child nodes by assigning containers from $|I^{i,l}|$ to stacking positions in H_1. In a node, we do so in the order in which the storage jobs are sequenced in n_1. On the succeeding $|I^{o,l}|$ levels, we consecutively assign containers from $I^{o,l}$ to stacking positions in H_2 and do so in the order in which the storage jobs are sequenced in n_2.

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 $1, \ldots, |J_1| + |J_2|$ 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 $l = 1, \ldots, |J_1| + |J_2| - 1$ has been appended to n_2, we can append a job to n_1 only if the last job in n_2 is a storage job and the next job in n_1 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 3.

Even if not every row position is defined, we can apply an adaptation of the routing approach presented in Sect. 4.1.2 for a pair of job sequences (n_1, n_2). Here, we employ lower bounds on the time necessary for a trolley to reach a row within the handshake area, as presented in Sect. 4.1.3. Additionally, for levels $1, \ldots, |J_1| + |J_2| - 1$, we determine $lb^e_c + W_c$ as detailed in Sect. 4.1.3. The completion time of c's last operation in a routing is then increased accordingly to obtain a lower bound on the makespan for a node.

A node in the search tree is then defined by (n_1, n_2). We cannot always derive a certain state of the stacking positions as long as not all jobs taking place in H_c, $c \in \{1, 2\}$, have a defined position. During the first phase of the branching, however, (n_1, n_2) suffices to determine the set of jobs A_c that have yet to be scheduled. During the second phase, we can determine the jobs that lack a definition of stacking positions.

We denote this B&B approach SEQ_{JS} throughout the computational study of Sect. 5.

4.2.2 Determining stacking positions first

In this section, we reverse both decisions and present a B&B algorithm where we determine stacking positions (part 1 of

Algorithm 4 Determining stacking positions first

```
initialize root node with empty job sequences n_1 and n_2 and no container in I^{i,l} ∪ I^{o,l} having a stack assigned; determine initial upper bound
while there are nodes in the search tree not yet considered do
    select node r
    if r is on a level smaller |I^{i,l}|+|I^{o,l}| (some containers in I^{i,l}∪I^{o,l} have no stack assigned) then
        choose first container i ∈ I^{i,l} ∪ I^{o,l} with no stack assigned
        create child nodes by assigning i to each feasible stack in H_1 (if i ∈ I^{i,l}) or H_2 (else)
    else
        for each crane c and job j of c not in n_c in r do
            create a child node (append j to n_c) observing Rules 1 to 4 (see Sect. 4.1.1)
        end for
    end if
    for each childnode s do
        determine an upper bound (see Sect. 4.1.4)
        determine a lower bound (see Sect. 4.1.3) and add c to the search tree as appropriate
    end for
end while
```

the TCSP decisions) of jobs in a first phase of the branching scheme. The second phase determines job sequences (parts 2 and 3 of the decision) based on the stacking position assignments. Part 4 of the decision is made as described in Sect. 4.1.2.

On the first $|I^{i,l}| + |I^{o,l}|$ levels of the search tree, we determine the stacking positions of containers in $I^{i,l}$ and $I^{o,l}$. On the subsequent $|J_1| + |J_2|$ levels, we construct sequences of jobs with already defined stacking positions by appending a job to n_1 or n_2 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 $|J_1| + |J_2|$ 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_1. 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_2, 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_1 and J_2.

Starting from level $|I^{i,l}| + |I^{o,l}|$, we determine 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 Sect. 4.1.1 in order to branch while avoiding duplicate nodes. Bounding and routing are done using identical (or slightly adapted) approaches as in Sect. 4.1. Having all rows of jobs defined allows us to employ the approach from Gilmore and Gomory (1964) in order to obtain a sequence for jobs in A_c with minimum empty row travel time as an additional lower bound. We dub this B&B algorithm SEQ_{SJ}.

5 Computational study

Our computational study presented in this section consists of three parts. First, we benchmark the solution performance of our three B&B procedures in Sect. 5.1. Here, we determine which of them is best suited for solving real-world TCSP instances. Then, in Sect. 5.2, we explore the best position of the handshake area and its impact on throughput performance. Finally, we challenge our best B&B procedure with simple heuristic approaches taken from the literature. By determining the optimality gaps of these straightforward solution methods, we quantify the price of simple planning. Introducing a handshake area is an operational simplification by itself compared with a more flexible any-bay handover (see Sect. 2), where any available stacking position is a possible intermediate stacking position for a preemptive container processing. Therefore, we also quantify the gap of our optimization approach with handshake area with a previous algorithm from the literature for any-bay handover. Both competitors are benchmarked with our best B&B procedure in Sect. 5.3.

We have executed all tests on an Intel Core i7-4790 CPU with 3.6 GHz and 32 GB of RAM running Windows 7, and all approaches have been implemented in Java 8. All tests in our computational study are based on randomly generated instances, applying the widely accepted data generator of Briskorn et al. (2019) for container yards. According to their definition of widespread real-world block settings, we assume 30 bays and 10 rows. Initially, crane 1 (2) is positioned in bay 0 (30), and the initial trolley positions are in row 1. The time to pick up or drop off a container is $p = 3$ time units.

To obtain different-sized test instances, we vary the number of containers to be processed. They are set to 10, 20, 30, and 40 containers, where we assume an equal share between inbound and outbound containers, i.e., $|I^i| = |I^o|$ holds for any instance. The storage positions of all containers are randomly drawn with uniform distribution from the complete block, except for the handshake area, where no containers can be stored permanently. Note that we also experimented with other test instances where the probability of the bays decreases the farther they are from the seaside access point. This could, for instance, arise in practical situations if outbound containers have been pre-marshaled into advantageous positions the night before (Lehnfeld & Knust, 2014). However, the optimization results obtained for these deviating bay distributions were not substantially different, so we decided to keep the test design as clear as possible and opted for the

Table 1 IDs for instance reproduction via the publicly available instance generator of Briskorn et al. (2019) at URL: instances.de/dfg

$\|I^o\| + \|I^i\|$	10	20	30	40
ID	zBZr2Ll	SRHiXWG	cPIxDoT	ddX4GoY

most basic way of generating data, i.e., based on a uniform distribution of container bays. For each number of containers, instance generation has been carried out 30 times, so that in total 120 different container sets have been obtained. These instances can be retrieved from the internet by replacing placeholder "INSTID" with the IDs provided in Table 1 within the following URL: instances.de/dfg/loadinstance.php?id=INSTID&instances=30.

For further details on the instance generator we refer the reader to Briskorn et al. (2019). Each of the instances is to be processed by the twin cranes with five different positions of the handshake area, i.e., $b^h \in \{5, 10, 15, 20, 25\}$, and in two capacity situations in the handshake area. We distinguish between a low-capacity setting, where the free capacity of each storage position in the handshake area is randomly drawn from $\{1, 2, 3\}$, and a high-capacity setting where the capacity is drawn from $\{3, 4, 5\}$ (with uniform distribution). In total, we have obtained 1200 instances.

Note that in addition to our B&B procedures, we also developed a time-based MIP model and solved it using CPLEX 12.6 (see Appendix C). During some preliminary testing, the model was clearly outperformed by our algorithmic approaches, and optimal solutions were not obtainable within a time limit of 1 hour even for the smallest test instances. Even when providing the default solver with tight upper bounds, this result did not improve. Thus, we abstain from further benchmarking of our model in the following computational study.

5.1 Computational performance of our B&B procedures

To benchmark our three B&B procedures, i.e., SIM (simultaneous sequencing and determining stacking positions, see Sect. 4.1), SEQ_{JS} (job sequencing first, see Sect. 4.2.1), and SEQ_{SJ} (determining stacking positions first, see Sect. 4.2.2), we solve our test instances with different runtime limits of 10, 300, and 600 seconds. In Table 2, we report the gaps of each B&B approach with the given runtime to the optimal solution. Whenever optimal solutions are not available because none of the three competitors could prove an optimal solution, we report the gap to the maximum lower bound obtained by our three B&Bs when having a runtime of 600 seconds. These gaps are reported in columns "Avg. relative gap LB %" for five alternative positions $b^h \in \{5, 10, \ldots, 25\}$ of the handshake bay. Furthermore, we report the average runtime in CPU seconds, in columns "Avg. runtime s," and differentiate between high- and low-capacity situations in the handshake area at the left and right of Table 2, respectively. The following conclusions can be drawn from these computational results:

- A first expected result is that all three B&Bs suffer from an increasing solution space. Gaps and runtimes increase the greater the number of containers to be handled and the higher the capacity in the handshake area. A higher capacity increases the alternative intermediate stacking positions to be evaluated for each container move.
- The same impact is seen moving the handshake area closer to the seaside access point. Recall that the distance from the seaside access point to the handshake area, and thus the workload of crane 1, increases with b^h. Our detailed investigation of the position of the handshake area in Sect. 5.2 will show that its optimal location under seaside workload peaks is somewhere between bays 7 and 10. Any (considerable) deviation from this position will cause one of the cranes to become the bottleneck resource, which reduces throughput performance. A handshake position of $b^h > 10$ leads to crane 1 having to shoulder the lion's share of the workload, whereas crane 2 is idle most of the time. In these settings, the main focus is on reducing the empty travel and idle time of crane 1, which obviously facilitates the work of all three B&Bs.
- Naturally, a longer runtime improves the solution quality. Increasing the runtime from 300 to 600 seconds, however, leads to only a minor improvement in solution quality in most cases. Astoundingly, the jump from 10 to 300 seconds also leads to only a relatively small gain, especially for all instances with a badly placed handshake area ($b^h \geq 15$) and few containers ($|I^i| + |I^o| \leq 30$). Only if the handshake area is closer to the seaside, such that the workloads of the two cranes are on a comparable level, and a greater number of container moves increases sequencing flexibility, may more computational time be well spent. We will investigate the development of the solution quality over time in more detail below.
- When benchmarking our three B&Bs, we can observe that SIM, which sequences and determines stacking positions simultaneously, is clearly outperformed by its two sequential competitors SEQ_{JS} and SEQ_{SJ}. Incorporating both types of information, i.e., crane sequences and stacking positions, simultaneously leads to a wider search tree already at the early levels, which obviously makes it more difficult to identify promising parts of the solution space.
- SEQ_{SJ} and its approach of first determining the stacking positions leads to the best results in most cases. One key advantage of SEQ_{SJ}, as compared to SEQ_{JS}, is that after the first phase of branching is completed, stacking positions for all containers are given. This allows SEQ_{SJ}

Table 2 Comparison of B&Bs with regard to average relative gap in percent from best LB and average runtimes in seconds for different capacity situations, numbers of containers $|I^o| + |I^i|$, and positions of the handshake area b^h

$\|I^o\|+$		Avg. relative gap LB %					Avg. runtime s				
$\|I^i\|$	$Algorithm/b^h$	5	10	15	20	25	5	10	15	20	25
Runtime limit: 10s											
10	SEQ_{JS}	0.23	0.02	0.03	0.00	0.00	1.90	0.81	0.73	0.03	0.03
	SIM	2.23	0.35	0.09	0.00	0.00	4.95	3.82	2.03	0.10	0.04
	SEQ_{SJ}	0.26	1.92	0.00	0.00	0.00	2.94	4.60	1.07	0.16	0.05
20	SEQ_{JS}	6.85	4.86	0.75	0.03	0.00	10.00	6.20	4.00	0.58	0.12
	SIM	4.61	3.12	1.64	0.87	0.20	8.40	7.89	6.82	3.09	0.83
	SEQ_{SJ}	4.55	1.82	0.23	0.00	0.00	7.43	5.22	3.53	0.16	0.30
30	SEQ_{JS}	9.18	10.95	5.42	0.66	0.08	10.00	9.62	9.04	3.15	1.20
	SIM	6.73	18.43	9.39	3.87	0.65	8.98	10.00	9.91	7.63	3.36
	SEQ_{SJ}	9.97	0.97	0.21	0.03	0.02	8.88	7.12	5.44	1.75	1.24
40	SEQ_{JS}	10.59	23.25	8.89	6.61	0.58	10.00	10.00	10.00	7.74	2.07
	SIM	8.30	22.97	10.83	6.81	3.05	10.00	10.00	10.00	9.43	5.83
	SEQ_{SJ}	17.27	5.15	1.12	0.36	0.02	10.00	9.18	6.92	3.91	1.94
Runtime limit: 300s											
10	SEQ_{JS}	0.00	0.00	0.00	0.00	0.00	14.45	0.84	2.81	0.03	0.03
	SIM	0.13	0.11	0.00	0.00	0.00	98.93	50.53	29.43	0.10	0.04
	SEQ_{SJ}	0.00	0.12	0.00	0.00	0.00	80.28	104.56	21.37	0.16	0.05
20	SEQ_{JS}	5.69	2.61	0.17	0.01	0.00	300.00	133.06	78.84	10.25	0.12
	SIM	3.55	1.69	0.40	0.10	0.00	242.32	187.54	135.30	23.20	1.69
	SEQ_{SJ}	1.96	0.52	0.22	0.00	0.00	173.98	111.99	100.20	0.16	0.30
30	SEQ_{JS}	9.18	9.58	4.25	0.03	0.04	300.00	280.29	242.11	25.42	23.04
	SIM	4.29	8.65	4.61	1.09	0.02	260.31	259.38	272.31	143.38	29.66
	SEQ_{SJ}	4.79	0.65	0.17	0.03	0.01	226.27	154.04	111.78	30.75	21.28
40	SEQ_{JS}	10.59	18.26	7.18	3.87	0.04	300.00	300.00	300.00	157.04	31.15
	SIM	6.41	16.04	6.40	3.77	0.40	300.00	300.00	290.56	261.96	76.93
	SEQ_{SJ}	7.99	0.31	0.16	0.04	0.02	286.96	160.71	125.30	53.06	30.94
Runtime limit: 600s											
10	SEQ_{JS}	0.00	0.00	0.00	0.00	0.00	14.45	0.84	2.81	0.03	0.03
	SIM	0.07	0.11	0.00	0.00	0.00	144.76	80.53	49.43	0.10	0.04
	SEQ_{SJ}	0.00	0.12	0.00	0.00	0.00	160.28	194.57	41.38	0.16	0.05
20	SEQ_{JS}	5.67	2.22	0.13	0.01	0.00	600.00	247.15	148.84	20.25	0.12
	SIM	3.52	1.61	0.30	0.06	0.00	482.32	367.54	247.35	33.20	1.69
	SEQ_{SJ}	1.85	0.39	0.18	0.00	0.00	335.41	222.00	191.66	0.16	0.30
30	SEQ_{JS}	9.18	9.47	3.87	0.03	0.04	600.00	560.29	482.11	45.42	43.04
	SIM	4.24	8.48	4.13	0.86	0.02	520.31	509.38	539.48	253.38	47.60
	SEQ_{SJ}	3.40	0.64	0.17	0.03	0.01	444.72	304.06	221.79	60.76	41.28
40	SEQ_{JS}	10.59	18.17	7.17	2.64	0.04	600.00	600.00	600.00	297.04	61.15
	SIM	6.34	15.07	5.84	3.43	0.30	600.00	600.00	580.56	491.96	131.17
	SEQ_{SJ}	7.69	0.28	0.16	0.04	0.01	567.03	307.17	245.31	103.06	55.99

Table 2 continued

$\|I^o\|+$		Avg. relative gap LB %					Avg. runtime s				
$\|I^i\|$	$Algorithm/b^h$	5	10	15	20	25	5	10	15	20	25
Runtime limit: 10s											
10	SEQ_{JS}	0.13	0.02	0.03	0.00	0.00	1.94	0.83	0.72	0.04	0.03
	SIM	2.41	0.35	0.09	0.00	0.00	5.06	3.96	2.06	0.10	0.04
	SEQ_{SJ}	0.32	1.88	0.00	0.00	0.00	2.72	4.62	1.08	0.15	0.05
20	SEQ_{JS}	6.39	3.00	0.36	0.04	0.00	10.00	6.12	3.68	0.58	0.11
	SIM	4.49	3.06	1.84	0.96	0.27	9.06	7.97	7.37	3.65	1.08
	SEQ_{SJ}	5.54	2.23	0.25	0.00	0.00	8.34	5.52	4.09	0.31	0.32
30	SEQ_{JS}	8.08	7.51	3.36	0.67	0.17	10.00	9.62	9.02	3.14	1.20
	SIM	6.67	18.38	8.85	4.45	0.72	8.98	10.00	10.00	7.80	3.67
	SEQ_{SJ}	11.23	3.04	1.36	0.28	0.03	9.69	8.24	7.38	2.68	1.29
40	SEQ_{JS}	9.69	17.64	6.56	3.68	0.40	10.00	10.00	10.00	7.66	2.10
	SIM	8.18	22.21	10.00	6.05	3.19	10.00	10.00	10.00	9.43	5.93
	SEQ_{SJ}	17.82	12.79	4.99	2.33	0.38	10.00	9.99	9.52	6.89	2.58
Runtime limit: 300s											
10	SEQ_{JS}	0.02	0.00	0.00	0.00	0.00	19.28	0.88	3.10	0.04	0.03
	SIM	0.59	0.11	0.00	0.00	0.00	103.26	53.28	30.89	0.10	0.04
	SEQ_{SJ}	0.00	0.16	0.00	0.00	0.00	80.05	106.16	21.43	0.15	0.05
20	SEQ_{JS}	5.23	1.15	0.17	0.01	0.00	300.00	128.65	80.81	10.25	0.11
	SIM	3.53	1.69	0.43	0.17	0.00	263.67	188.71	142.46	30.36	2.29
	SEQ_{SJ}	3.14	0.80	0.22	0.00	0.00	194.34	115.01	100.99	0.31	0.32
30	SEQ_{JS}	8.08	5.38	2.15	0.03	0.04	300.00	280.28	235.59	24.69	23.00
	SIM	5.34	8.12	6.14	1.61	0.02	260.32	261.71	277.04	167.47	34.45
	SEQ_{SJ}	7.79	1.21	0.18	0.02	0.01	266.82	166.33	119.46	22.58	21.34
40	SEQ_{JS}	9.69	11.96	5.00	1.95	0.01	300.00	300.00	291.04	150.81	31.22
	SIM	7.26	17.08	6.31	4.32	0.70	300.00	300.00	290.73	271.19	86.81
	SEQ_{SJ}	13.23	1.40	0.67	0.42	0.00	300.00	194.83	185.99	89.13	15.75
Runtime limit: 600s											
10	SEQ_{JS}	0.02	0.00	0.00	0.00	0.00	29.28	0.88	3.10	0.04	0.03
	SIM	0.38	0.11	0.00	0.00	0.00	172.55	83.28	50.89	0.10	0.04
	SEQ_{SJ}	0.00	0.12	0.00	0.00	0.00	160.06	197.81	41.44	0.15	0.05
20	SEQ_{JS}	5.20	0.98	0.13	0.00	0.00	600.00	237.74	150.81	14.48	0.11
	SIM	3.29	1.61	0.36	0.17	0.00	523.67	368.71	260.77	50.36	2.29
	SEQ_{SJ}	2.86	0.76	0.18	0.00	0.00	384.36	225.02	192.71	0.31	0.32
30	SEQ_{JS}	8.08	5.12	2.15	0.03	0.04	600.00	560.28	465.59	44.69	43.00
	SIM	5.33	7.82	5.17	1.28	0.02	520.32	501.71	547.04	296.04	52.99
	SEQ_{SJ}	7.12	1.21	0.18	0.02	0.01	526.86	326.35	229.47	42.58	41.34
40	SEQ_{JS}	9.69	11.94	4.95	1.69	0.01	600.00	600.00	581.04	280.81	61.22
	SIM	7.14	15.47	6.18	3.93	0.59	600.00	600.00	580.73	528.29	156.81
	SEQ_{SJ}	11.56	1.40	0.53	0.35	0.00	600.06	364.86	341.16	159.14	25.75

Table 3 Total number of optimal solutions obtained after 600 CPU seconds for different numbers of jobs $|I^o| + |I^i|$ and positions of the handshake area b^h

$\|I^o\|+$ $\|I^i\|$	Algorithm / b^h	# optimal solutions				
		5	10	15	20	25
10	SEQ_{JS}	59	60	60	60	60
	SIM	56	56	60	60	60
	SEQ_{SJ}	60	54	60	60	60
20	SEQ_{JS}	2	41	46	59	60
	SIM	10	24	38	57	60
	SEQ_{SJ}	35	38	42	60	60
30	SEQ_{JS}	0	4	13	56	56
	SIM	8	11	8	37	58
	SEQ_{SJ}	13	29	38	55	58
40	SEQ_{JS}	0	0	1	33	54
	SIM	0	0	2	13	48
	SEQ_{SJ}	2	29	34	48	57

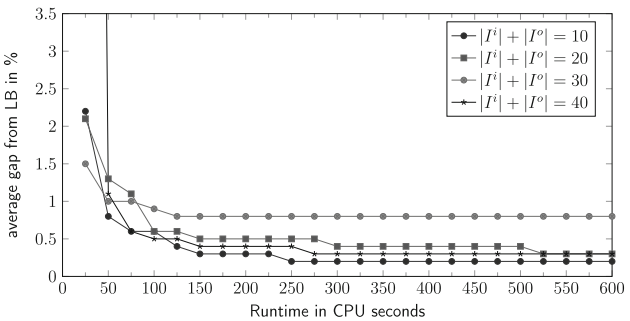

Fig. 3 Average gap of SEQ_{SJ} from lower bound (LB) over time

to determine tighter lower bounds using the approach of Gilmore and Gomory (1964), see Sect. 4.1.3. Only in the case where just a few containers are processed ($|I^i| = |I^o| = 10$) will SEQ_{JS} and its approach of first determining job sequences lead to slightly better results. Due to the relatively small number of precedence relations occurring in these instances, an arbitrary pair of sequences determined in the first phase of SEQ_{JS} can more likely be complemented to a feasible schedule, and the search is well guided, whereas this is less likely for larger instances.

Our main finding of SEQ_{SJ} outperforming both competitors in most cases is confirmed when comparing the number of optimal solutions after 600 CPU seconds in Table 3.

In total, SEQ_{SJ} solves 892 out of the 1200 total instances (i.e., 74.33%) to proven optimality. Thus, SEQ_{SJ} is our best-performing B&B approach, and is applied in all further computational tests.

Finally, we plot the development of the objective values over time in Fig. 3. Specifically, we plot the average gaps of

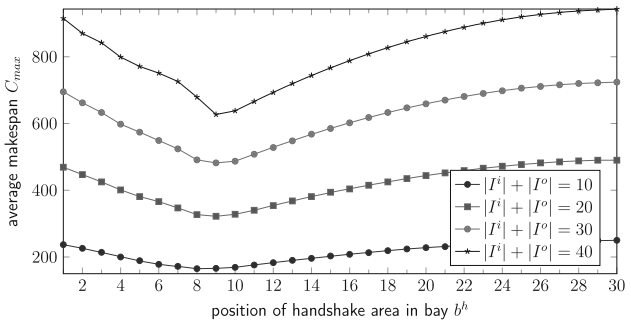

Fig. 4 Average makespan C_{max} for different positions of the handshake area

SEQ_{SJ}'s current objective value from the best lower bound obtained by our three B&Bs after 600 seconds over time. These graphs confirm that high-quality solutions well below a 3% gap can be quickly obtained. Spending more than 50 seconds of computational time barely improves the results. This is good news for the application of our B&B approach in real-world yards. Crane scheduling is an operational decision task where only a few (dozens of) seconds are available before the plan for the next set of container moves has to be available. Thus, there may not be enough time to apply our B&B as an exact approach. However, it seems well suited to deliver solutions of acceptable quality if only very limited time is available, and even better solutions if a bit more computational time is at hand.

5.2 Where to position the handshake area?

This section explores the question of where to position the handshake area. Note that positioning the handshake area is not a strategic decision, but rather a short-term choice that can easily be altered on short notice. Once a large vessel arrives, for instance, a new handover bay valid during the forthcoming seaside workload peak can be fixed wherever enough intermediate stacking space is available. Note that, with all container moves being related with the seaside, it is to be expected that a placement of the handshake area not in the middle of the block but closer to the quay improves performance. However, how far such a movement toward the seaside is advisable is not that obvious and is the subject of the following experiment.

For each of our 600 instances with low-capacity utilization, we place the handshake bay at each possible position $1 \leq b^h \leq 30$, solve the resulting instance with our best-performing B&B procedure SEQ_{SJ}, and determine which of the alternative handover bay positions leads to the lowest makespan. To keep the overall runtime of the experiment within a reasonable time frame, we restrict the runtime of each single optimization run to 150 seconds. Recall that our performance tests in Sect. 5.1 showed that additional time

Table 4 Average makespan C_{max} and workloads W_1 and W_2 of crane 1 and 2, respectively, for different positions of the handshake area

$\|I^o\| + \|I^i\|$	b^h	4	5	6	7	8	9	10	11	12	13	14
5	C_{max}	200	189	178	172	166	**166**	169	176	183	190	196
	W_1	100	109	118	126	**134**	141	148	155	161	167	173
	W_2	167	157	146	137	126	118	107	99	90	81	74
10	C_{max}	401	381	366	347	327	**322**	328	340	354	368	381
	W_1	199	217	234	250	**266**	281	295	309	322	334	345
	W_2	346	324	304	283	263	244	225	208	190	174	159
15	C_{max}	598	574	549	524	491	**482**	487	508	528	548	568
	W_1	299	326	351	376	**399**	421	443	463	483	501	519
	W_2	513	481	447	418	389	360	334	309	284	258	236
20	C_{max}	799	771	751	726	679	**627**	638	666	693	720	744
	W_1	398	435	469	501	**533**	563	591	618	644	669	691
	W_2	687	642	599	558	521	482	447	410	376	343	311

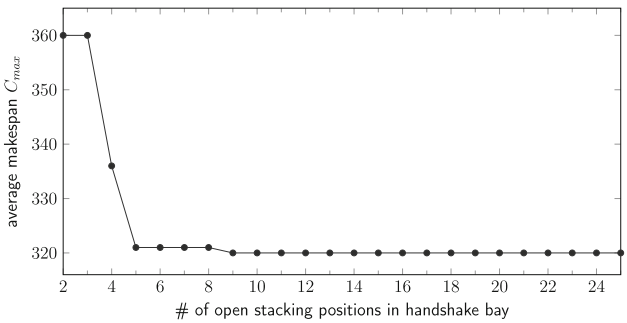

Fig. 5 Average makespan C_{max} for different capacity situations in the handshake area

barely improved the results. The average makespans C_{max} obtained by B&B for the alternative positions of the handshake area are plotted in Fig. 4. These results show that under a seaside workload peak, the optimal handshake position is in either the eighth or ninth bay.

In real-world operations, it seems rather unhandy to determine multiple detailed crane schedules just to (re)position the handshake bay. A simpler rule-based approach seems much more practicable. To derive such an approach, Table 4 lists the minimum average makespan C_{max} for bay positions $4 \leq b^h \leq 14$ and the loaded workloads W_1 and W_2 for both cranes. The loaded workload is schedule-independent and only contains the workload where a crane travels loaded with a container; idle time and empty travel, which are schedule-dependent, are not considered. Thus, the loaded workload for a given set of container moves—for instance, all those container moves of a yard block related to the berthing vessel—can easily be determined for different positions of the handshake bay without having to determine detailed schedules. The results of Table 4 show that the best handover bay is at most a single bay next to the bay where the maximum loaded workload of both cranes $\max\{W_1; W_2\}$ reaches its minimum (marked in bold).

Thus, the following simple rule can be applied to determine an approximated position for the handover bay: Select a bay with sufficient stacking capacity as close to the bay where the maximum loaded workload of both cranes reaches its minimum. Repositioning the handover bay for varying workload situations (e.g., during seaside peaks) according to this rule holds a large potential for performance gains, which is also indicated by the results of Table 4. For instance, a permanent placement of the handover bay in the middle of the container block, which seems a quite natural and obvious choice, increases the workload by about 20% compared to the optimal placement. Note that a position of the handover bay balancing workloads of both cranes has been shown to lead to instances that are hard to solve in the proof of Theorem 2.

Naturally, flexibly repositioning the handshake area requires that sufficient stacking space in the selected handover bay is available. In the following experiment, however, we show that a handful of open stacking positions is sufficient, without substantial slowing of crane operations. Specifically, we set up the following experiment. We solve our instances with $|I^i| + |I^o| = 20$ and the handshake area positioned in bay $b^h = 9$ with our best-performing B&B SEQ_{SJ} and a timeout of 300 seconds. Each of these instances is solved multiple times with varying open stacking capacity in the handshake bay. We start with an almost completely filled handshake bay, where only two open stacking positions are available (i.e., one for inbound and one for outbound containers), and determine the makespan for this setting. Then, we randomly select and remove further containers from the handover bay, one by one, so that additional open stacking positions for intermediate placement of boxes are added until, finally, the handshake area is completely empty. In this way, we explore the impact of open stacking capacity on the makespan, and we can answer the question as to the minimum available capacity a bay must have to be considered a valid candidate for a potential handshake bay.

The average makespan over all instances in relation to the available open stacking capacity in the handshake area is plotted in Fig. 5. Note that we skip the results for more than 25 open stacking spaces, since no further performance gains are realized.

These results indicate that relatively little open stacking space is required to relocate the handover bay to another bay on short notice. More than five open stacking spaces do not improve the performance noticeably. It can be concluded that relocating the handshake area on short notice, even if not much open stacking space is available, seems a good strategy to improve throughput performance whenever there is a structural change in the container workload (e.g., after arrival of a large vessel).

5.3 Benchmark test

In this section, we benchmark our best B&B procedure when sharing workload via a handshake area with three competitors:

- *NO-SHARING* If both cranes are fixedly assigned to their dedicated access point, we have no workload sharing in the strict sense of our definition (see Sect. 2). During seaside workload peaks, the NO-SHARING policy induces that seaside crane 1 has to process all inbound and outbound moves exclusively, while landside crane 2 remains idle. The makespan minimization problem then reduces to a matching of inbound and outbound container moves jointly executed between any successive visits at the access point (see Boysen & Stephan, 2016). Note that, since runtime is less of an issue in this experiment, we can directly apply our B&B procedures for this optimization task and only have to manipulate the input data to a single crane setting. This comparison clarifies research question **RQ1**, whether workload sharing via a handshake area can substantially improve yard throughput compared to the NO-SHARING policy.
- *ANY-BAY* Furthermore, we also compare our optimized crane scheduling with handshake area with an any-bay handover (see Sect. 2). The latter policy allows us to hand over containers between cranes in any bay where free stacking space is available. A previous heuristic algorithm from the literature coordinating twin cranes under the ANY-BAY policy is that of Briskorn et al. (2016). The alterations necessary to adapt their heuristic to our setting with inbound and outbound containers are elaborated in Appendix B. Note, however, that this ANY-BAY heuristic relaxes stacking capacities, so that the results are merely a lower bound and tend to overestimate the advantages of any-bay handover. This benchmark test clarifies research question **RQ2**, whether the simplified crane coordination via a handshake area comes with the price of excessive throughput loss compared to the more flexible (yet more complicated) coordination of any-bay handover.
- *HANDSHAKE-HEU* Previous research on workload sharing in container yards (e.g., Gharehgozli et al., 2017; XiaoLong et al. 2019) mainly focuses on simple heuristic procedures to coordinate container transfers between cranes via a handshake area. Given our sophisticated optimization approach, we benchmark these approaches. In this way, we clarify research question **RQ3**, whether simple heuristics are sufficient to schedule container transfers via a handshake area or will produce large optimality gaps, such that exact solution approaches like our best-performing B&B approach are the better choice. Specifically, we evaluate two approaches from the literature: a straightforward nearest-neighbor heuristic, where jobs are assigned in a greedy manner while minimizing the empty travel between two consecutive container transports (dubbed HANDSHAKE-HEU-NN), and a local search procedure, which improves the initial solution obtained with HANDSHAKE-HEU-NN by interchanging jobs in the cranes sequences (dubbed HANDSHAKE-HEU-LS). Both approaches are based on the work of Gharehgozli et al. (2017), and we describe them in more detail in Appendix B.

We apply our best-performing B&B procedure SEQ_{SJ} (with a timeout after 150 CPU seconds) as the optimization approach for crane coordination via a handshake area. In the following, we dub this approach HANDSHAKE-OPT. The results of this benchmark test over our 1200 total data instances are summarized in Table 5. Specifically, this table reports the average relative gaps (in percent) in relation to the results of our HANDSHAKE-OPT approach for a handshake area optimally placed in bay $b^h = 9$. In relation to this benchmark, we report the relative performance loss of all approaches listed above once for an optimally placed handshake area in bay $b^h = 9$ and for a handshake area in the middle of the container block in bay $b^h = 15$. Note that, naturally, the approaches NO-SHARING and ANY-BAY do not apply a handshake area, so their results are not impacted by the placement of the handshake bay. Further note that we execute heuristic HANDSHAKE-HEU-LS with 10,000 local search moves, which is considerably more than the 1000 moves applied by Gharehgozli et al. (2017). The results of this experiment are summarized in Table 5 and suggest the following answers to our research questions:

- **RQ 1** Workload sharing via a handshake area has the potential to improve yard throughput substantially compared to the NO-SHARING policy where both cranes are fixedly assigned to their designated access points. The

Table 5 Average relative gap (in percent) to the makespan optimized according to HANDSHAKE-OPT with the handshake area positioned in bay $b^h = 9$

	Algorithm	$\|I^i\| + \|I^o\|$			
		10	20	30	40
	NO-SHARING	50.01	52.33	51.02	50.80
	ANY-BAY	0.04	1.35	0.83	3.00
$b^h = 9$	HANDSHAKE-HEU-NN	65.53	76.44	78.05	81.88
	HANDSHAKE-HEU-LS	4.67	8.24	7.80	19.25
	HANDSHAKE-OPT	0	0	0	0
$b^h = 15$	HANDSHAKE-HEU-NN	59.58	67.69	67.40	71.18
	HANDSHAKE-HEU-LS	26.24	25.16	28.51	28.08
	HANDSHAKE-OPT	22.93	22.53	22.15	22.72

makespan of NO-SHARING is more than 50% higher than our optimization approach.

- **RQ 2** On first sight, the benchmark test between workload sharing with handshake area and its more flexible counterpart based on any-bay handover delivers a surprising result. Although it provides greater flexibility, any-bay handover leads to a slightly higher makespan than its competitor. This is due to the heuristic gap of our solution procedure applied for the ANY-BAY policy. We adapt the heuristic procedure of Briskorn et al. (2016) to our problem setting, which is based on the bucket-brigade protocol (see Appendix B for more details). Obviously, this heuristic is not able to deliver optimal results, because optimal objective values for the ANY-BAY policy cannot exceed those obtained with a handshake area, due to the larger solution space. Thus, it remains a future research task to benchmark HANDSHAKE-OPT with optimal solutions for the ANY-BAY policy. Our results suggest that a handshake area placed in optimal position (i.e., in bay $b^h = 9$ in the case of seaside workload peaks) leads to a comparable yard throughput to that of ANY-BAY.
- **RQ 3** Coordinating workload sharing via a handshake area with simple decision rules seems a bad idea. Applying a straightforward nearest-neighbor heuristic such as HANDSHAKE-HEU-NN leads to an increase in makespan of between 65.53 and 81.88% depending on the number of container moves. Thus, planning that is too simple comes at the price of considerable performance loss. More elaborate heuristics based on local search such as HANDSHAKE-HEU-LS reduce the gap, but still reach a performance loss of 10% and greater if many container moves need to be processed ($|I^i| + |I^o| = 40$) and/or not enough local search moves are executed. Note that increasing the number of local search moves for HANDSHAKE-HEU-LS to 100,000 moves did not significantly improve the results. Further note that HANDSHAKE-HEU-LS applies our routing procedure of Sect. 4.1.2 based on DP to coordinate the cranes for given container sequences and stacking positions, so that it already contains sophisticated optimization and goes beyond those approaches presented in the literature.

To conclude, simplifying the coordination of twin cranes via a handshake area does not lead to excessive performance loss, as long as the position of the handshake area is (re)located to an appropriate position that (roughly) halves the workload for both cranes. Planning that is too simple, however, may deteriorate throughput performance considerably. Simple heuristics are clearly outperformed by sophisticated optimization procedures.

6 Conclusions

In this paper, we treat crane scheduling in a container block where twin cranes share container processing during seaside workload peaks via a handshake area. We develop three alternative branch and bound approaches to minimize the makespan of container processing. One of them in particular is shown to deliver either optimal solutions for instances up to 40 container moves if enough computational time is available, or near-optimal solutions if less time is at hand. This paper is the first to derive optimal solution approaches for a handshake setting, which allows us to derive the following take-home messages from our computational study:

- Introducing a handshake area is an organizational decision to ease the coordination of twin cranes jointly operating a container block. If the handshake bay is positioned in the optimal position, such that both cranes have a similar workload, and their schedules are optimized (e.g., by our B&B approaches), the price for this organizational simplification is low. Compared to an any-bay handover, where each available stacking position is a potential handover position between cranes, the loss in throughput performance is small. Note, however, that our computational test applies a heuristic approach for scheduling

the cranes under any-bay handover. Thus, future research must determine whether this finding still holds if two exact solution procedures are benchmarked.
- This finding, however, is no longer true if the handshake area is poorly placed, e.g., always in the middle bay of the container block. In this case, one crane easily becomes the bottleneck resource and slows container processing considerably. This is especially painful under seaside workload peaks if large container vessels are to be processed under great time pressure.
- Performance losses also arise when planning is too simple. Existing research only investigates the application of a handshake area when coordinating both cranes with simple rules for job sequencing and handshake access prioritization. Our computational results show that these approaches may lead to considerable performance loss.

The main challenge we see for future research is a systematic comparison of all kinds of workload sharing in container blocks. In Sect. 2, we elaborate the four basic kinds of workload sharing, i.e., front evasion, cross-over cranes, any-bay handover, and handshake area. For each of these organizational approaches, different exact and heuristic solution algorithms have been introduced in the literature, and some of these procedures have also been compared with one or two alternatives of workload sharing. A systematic comparison with exact methods (to exclude heuristic gaps) and with state-of-the-art heuristics (whenever computational results have to be obtained quickly) of all kinds of workload sharing on a unique dataset is still lacking. Such a benchmark test would provide great practical decision support for port managers having to set up their container yards for a competitive processing of today's mega vessels.

Another viable direction for future research is to extend our research in order to use a dedicated handshake area in a more flexible manner. Specifically, we see four opportunities to do so:

- First, a switching policy could be applied, where a dedicated handshake area and any-bay handover are employed during distinct time intervals. While crane scheduling procedures exist for both operational regimes in isolation (see Sect. 2), future research should investigate the appropriate time for a switch.
- Under a mixed policy, a handshake area is the default option, but the cranes are exceptionally allowed to cross the handshake area whenever a direct delivery promises a better performance and does not impact the other crane. To account for the mixed policy, our solution frameworks require adaptation when determining the stacking position of a container in $I^{i,l} \cup I^{o,l}$. This additional option is to not store the container intermediately. To do so, we see at least two algorithmic components that require alteration by future research: the lower bound for the makespan and collision avoidance that is no longer restricted to the handshake area.
- Another alternative for more flexible crane operations is the application of multiple alternative handshake areas. This requires the additional choice of the right handshake area for each container move and adds potential crane interference. The adaptations to be made to our solution frameworks are similar to the ones sketched above. We could consider each potential stacking position (in each bay of the handshake area) as an optional stacking position. Additionally, if there is at least one handshake area beyond a container's destination position, it is also possible to deliver it directly. That is, the container can be handled by the seaside crane alone, unless the container passes all bays of the handshake area. The algorithmic components to be adapted are the same as those above.
- Finally, we can allow a crane to pick up a container a second time for further transport (with or without a potential crossing of the handshake bay). Obviously, in the former case, we would lose the fixed assignment of a crane to each transport job. Our frameworks, then, need an additional decision step of whether the same or the other crane picks up a container after it has been intermediately stored. Again, significant changes to our frameworks are inevitable.

Each of these four opportunities promises an improvement in container handling efficiency but adds much complexity to the solution process. Thus, a handshake area remains a challenging field for the port operations community in the foreseeable future.

Funding Open access funding enabled and organized by Projekt DEAL.

A Proofs

Proof of Theorem 2 The proof is by reduction from 3-PARTITION, which is known to be strongly NP-complete; see Garey and Johnson (1979).

3-PARTITION Given $3m+1$ integers $a_1, \ldots, a_{3m}, A$ with $\frac{A}{4} < a_i < \frac{A}{2}$ for each $i = 1, \ldots, 3m$ and $\sum_{i=1}^{3m} a_i = mA$. Does there exist a partition of set $\{1, 2, \ldots, 3m\}$ into m subsets $A_1, \ldots, A_m$ such that $\sum_{i \in A_l} a_i = A$ for each $l = 1, \ldots, m$?

It is not hard to see that 3-PARTITION remains a complex matter if $A > 4m+3$. In this case, we can simply multiply all numbers by $4(m+1)$ without changing the problem structure.

For a given instance of 3-PARTITION, we construct the following instance of TCSP. We have $B = 3A + 1$, $R = 2$, and a handshake area positioned in bay $b^h = A$. We furthermore, have $I^{i,s} = \{1, \ldots, 3m\}$, $I^{i,l} = \{3m+1, \ldots, 4m+1\}$, $I^{o,s} = \emptyset$, and $I^{o,l} = \{4m+2, \ldots, 5m+2\}$ with the following individual properties.

- For each $i \in I^{i,s}$, we have $o_i = (0, 1)$ and $d_i = (a_i, 1)$.
- We have $o_i = (0, 1)$ for each $i \in I^{i,l}$, $d_{3m+1} = (2A + 1, 1)$, and $d_i = (3A + 1, 1)$ for each other $i \in I^{i,l}$.
- We have $d_i = (0, 2)$ for each $i \in I^{o,l}$, $o_{5m+2} = (2A + 1, 2)$, and $o_i = (3A + 1, 2)$ for each other $i \in I^{o,l}$.

Note that all inbound containers travel only through row 1, while all outbound containers travel only through row 2. Initially, cranes are located in $o_1^0 = (0, 1)$ and $o_2^0 = (2A, 2)$.

Clearly, the construction of the instance of TCSP can be done in polynomial time. We claim that the answer to the instance of 3-PARTITION is yes if and only if a makespan of

$$\overline{C} = 4mA + 4m + 2A + 3$$

can be achieved in the instance of TCSP.

First, we assume that a makespan of $\overline{C}$ is achieved. Recall that $p = 0$, and thus the only time-consuming activities of cranes involve moving and waiting for the other crane to move out of the way. Note that the total loaded travel time of crane 2 amounts to

$$\begin{aligned} L_2 &= \sum_{i \in I^{o,l}} (o_i^b - b^h)) + \sum_{i \in I^{i,l}} (d_i^b - b^h) = (m(2A + 1) \\ &\quad + (A + 1)) + ((A + 1) + m(2A + 1)) \\ &= 4mA + 2m + 2A + 2. \end{aligned}$$

Furthermore, between handling an inbound container and an outbound container, crane 2 needs at least one period for adjusting its trolley, and hence, crane 2 needs at least

$$L_2 + 2m + 1 = \overline{C}$$

time units to accomplish all operations. Thus, for achieving a makespan of $\overline{C}$, crane 2 cannot wait at all or make any empty moves involving the gantry. Note that containers in both $\{3m + 2, \ldots, 4m + 1\}$ and $\{4m + 2, \ldots, 5m + 1\}$ are identical, and therefore, we can assume that crane 2 handles container $5m + 2$ first, container $3m + 1$ last, and containers in $\{3m + 2, \ldots, 5m + 1\}$ in between (alternating between inbound and outbound containers). This fixes the position of the gantry of crane 2 over time throughout the planning horizon. In particular, crane 2 is located in bay b^h in time interval $[A + 1 + k \cdot (4A + 4), A + 2 + k \cdot (4A + 4)]$ for each $k = 0, \ldots, m$, and thereby delivering a container in $I^{o,l}$ at time $A + 1 + k \cdot (4A + 4)$ to $(b^h, 2)$, moving the trolley to $(b^h, 1)$, and picking up a container in $I^{i,l}$ at $A+2+k\cdot(4A+4)$.

Since crane 2 delivers the last container in $I^{o,l}$ to b^h not before $A + 1 + m \cdot (4A + 4)$, crane 1 picks it up not before

$$A + 1 + m \cdot (4A + 4) + 2 = \overline{C} - A$$

and, hence, crane 1 ends up in bay 0 after delivering this job to position $(0, 2)$ if a makespan of $\overline{C}$ is achieved.

The total loaded travel time of crane 1 amounts to

$$L_1 = |I^{o,l} \cup I^{i,l}| \cdot A + \sum_{i \in I^{i,s}} d_i^b = (2m + 2)A + mA.$$

After delivering containers in $I^{i,s}$, a total gantry empty travel distance of $E_1^1 \geq mA$ is due since, after delivering container $i \in I^{i,s}$ to bay a_i, the closest bay where a container can be picked up is bay 0.

Since $L_1 + E_1^1 \geq \overline{C} - (4m + 3)$ and $A > 4m + 3$, crane 1 cannot travel empty from bay b^h to bay 0 if a makespan of $\overline{C}$ is achieved. Thus, after delivering a container in $I^{i,l}$, it needs to pick up a container in $I^{o,l}$ before handling any other container. This results in three extra time units for moving from bay b^h to bay $b^h - 1$, letting crane 2 deliver a container to b^h and pick up the container just delivered by crane 1, and moving back to bay b^h (recall that crane 2 does not make any detours or waits if a makespan of $\overline{C}$ is achieved). Since we have $m+1$, such encounters between cranes 1 and 2, we have at least $D_1 \geq 3(m + 1)$ time units for detours and waiting of crane 1 while exchanging containers with crane 2.

Finally, after delivering a container to bay 0, we have an empty travel time of at least one since crane 1 does not pick up containers in position $(0, 2)$. Since crane 1 delivers $m + 1$ containers to bay 0, this results in an additional empty travel time of at least $E_2^1 \geq m$.

Hence, in order to achieve a makespan of $\overline{C}$, crane 1 needs at least

$$\begin{aligned} L_1 + E_1^1 + D_1 + E_1^2 &= (2m + 2)A + mA \\ &\quad + mA + 3(m + 1) + m = 4mA + 4m + 2A + 3 = \overline{C} \end{aligned}$$

time units to accomplish all operations. Furthermore, it accomplishes its operations in $\overline{C}$ time only if $E_1^1 = mA$, $D_1 = 3(m+1)$, and $E_2^1 = m$. The latter implies that crane 1 is busy delivering containers in $I^{i,l} \cup I^{o,l}$ each time interval $[0+k\cdot(4A+4), 2A+3+k\cdot(4A+4)]$, $k = 0, \ldots, m$. This leaves time interval $[2A+3+k\cdot(4A+4), (k+1)\cdot(4A+4)]$, $k = 0, \ldots, m-1$, for picking up and delivering containers in $I^{i,s}$ plus an additional trolley adjustment before picking up the first container in each interval. Hence, total travel distance (loaded travel plus following empty travel) of containers delivered in each such interval does not exceed $2A$. Since total travel distance equals $2mA$, the total travel distance of containers delivered in each such interval is exactly $2A$ and, thus, constitutes a yes certificate to the instance of 3-PARTITION.

Now, it is easy to see that if the answer to the instance of 3-PARTITION is yes, then we can achieve a makespan of $\overline{C}$ using the structure outlined above. □

Figure 6 illustrates the structure of a solution certifying a yes instance of 3-PARTITION. We see the time horizon on the vertical axis, gantry positions on the horizontal axis. The positions of cranes 1 and 2 over time are depicted as the left line and the right line, respectively.

The position of crane 2 over time is fixed as outlined in the proof. Also, travels of crane 1 starting at bay 0, visiting b^h, and returning to bay 0 are fixed. The encounters between the two cranes are encircled and enlarged to provide greater detail. The choice of containers handled by crane 1 in time intervals of length $2A$ is not fixed and certifies a yes instance of the 3-PARTITION if a makespan of $\overline{C}$ is achieved.

Proof of Lemma 1 For the proof, we focus on a single stacking position and the operations 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 job $j^2(a)$ precedes $j^2(b)$ in the other crane's job sequence as well, then a must be retrieved before b can be stored, since otherwise b needs to be retrieved before a can be retrieved. Second, if job $j^2(a)$ succeeds $j^2(b)$ in the other crane's job sequence, then b is stored before a is retrieved. In both cases, we have a distinct order in which the operations in the handshake area related to a and b are conducted. □

Proof of Lemma 2 We consider two arbitrary containers a and b that are 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 operation of $j^2(a)$ precedes the storage operation of $j^1(b)$ in the stacking position due to Lemma 1, and the order in which operations related to a or b are carried out coincides with the order in which the respective jobs are 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 operations are carried out. □

Proof of Lemma 3 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 operations related to j and j' are in the same stacking position. However, due to Lemma 2, the precedence relations between operations in the same stack are in line with the order in which corresponding jobs are appended. □

Proof of Theorem 3 The proof is twofold. 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_1, n_2) constructed by the proposed branching scheme. Assume that n_1 and n_2 are constructed up to a certain point, and let k_c, with $c \in \{1, 2\}$, 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 $\{k_1, k_2\}$ is the next job in σ.

We first show that, following Rules 1 to 4, only one job in $\{k_1, k_2\}$ can be the next job in σ. We do so by distinguishing the following cases.

- If $k_1 \in N_1$, then k_1 is the next job in σ, since k_1 cannot follow a job from J_2 in σ due to Rule 4.
- If $k_1 \notin N_1$ and k_1 and k_2 imply operations in the same stacking position, then (n_1, n_2) together with Rules 1 and 2 implies the next job in σ due to Lemma 2.

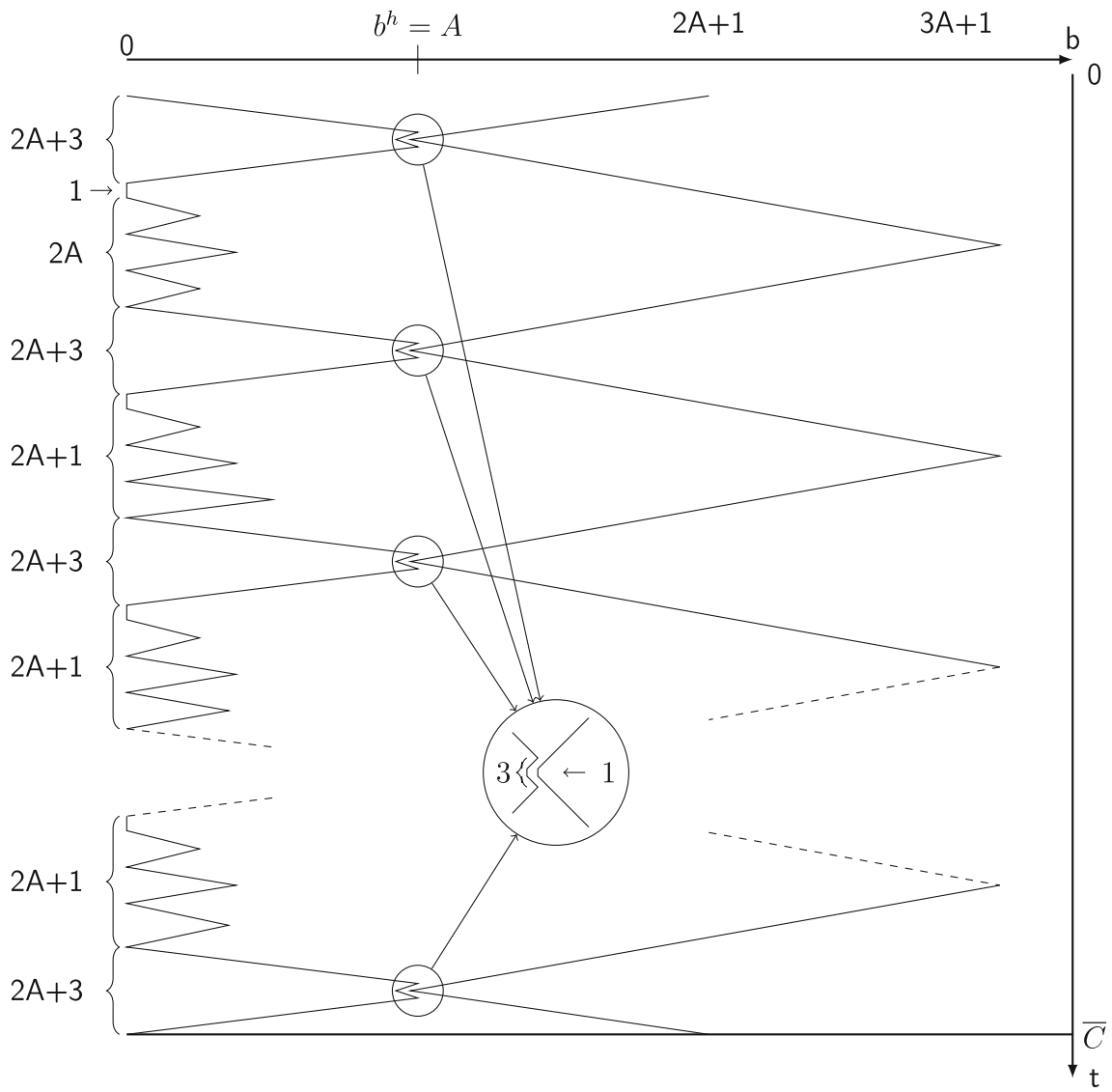

Fig. 6 Solution structure in the proof of Theorem 2

- If $k_1 \notin N_1$ and k_1 and k_2 imply operations in different stacking positions, jobs are appended as follows.
  - If any $k_c, c \in \{1, 2\}$ has a predecessor in n_{3-c} not yet in σ (all predecessors are implied by (n_1, n_2) due to Lemma 1), this job cannot be the next job in σ due to Lemma 2. Hence, only k_{c-1} can be the next job in σ.
  - If both k_1 and k_2 have all their predecessors in σ, then Rule 4 implies that k_1 is the next job in σ. Assume that k_2 is the next job in σ. Then, Rule 4 requires that k_1 follows immediately after a job in J_2, implying an operation in the same stacking position. However, such a job would be a predecessor of k_1, which we assumed does not exist.

Second, we show that for any feasible pair of job sequences (n_1, n_2) there is a branching sequence σ leading to (n_1, n_2). We do so by giving a procedure constructing σ from (n_1, n_2) and obeying Rules 1 to 4

1. Let σ be empty.
2. Append jobs from n_1 not yet in σ in the same order until the next job k_1 in n_1 has a predecessor not yet in σ or all jobs in n_1 are appended. Go to 2.
3. Append jobs from n_2 not yet in σ in the same order until all jobs in n_2 are appended or the predecessor of k_1 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_1 and n_2. 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_1. Since jobs in n_2 are appended to σ only if necessary to proceed with n_1, we do not have to ensure this in step 3 due to acyclic precedence relations in a feasible pair of job sequences (n_1, n_2).
- Rule 3 is followed, since feasible pair of job sequences (n_1, n_2) 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_1 to σ.

Concluding, the branching scheme ensures that each feasible pair of job sequences (n_1, n_2) can be constructed. □

B Competitors of our B&B procedures

This appendix details the two competitors challenging our B&B procedures. First, we address simple priority-rule-based approaches for workload sharing via a handshake area taken from the literature. We briefly explain these approaches and adapt them to our more involved problem setting where also the stacking positions in the handshake area are part of the decision. Furthermore, we describe and adapt a solution procedure from the literature for any-bay handover (see Sect. 2) where preemptive container handling is not restricted to a dedicated handshake area, but can be split in any available stacking position in the yard.

B.1 Heuristics for workload sharing via handshake area

In this section, we present two approaches closely following the work in Gharehgozli et al. (2017). In particular, we take the first three parts of the decision taking up ideas of Gharehgozli et al. (2017) and deviate only where it becomes necessary due to our problem setting differing in details.

First, we specify stacking positions of containers. While Gharehgozli et al. (2017) 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 exists 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. We do exactly the same, except we choose a position within the only handshake area this way. In the case of a tie between positions, we choose the one closest to the middle row $(R/2)$. By storing the containers closely together, we aim to reduce the time necessary to adjust the trolley between consecutive operations of the same crane in b^h.

For the first part of the decision taken, we employ the two heuristic approaches introduced by Gharehgozli et al. (2017), namely a nearest-neighbor and a local search heuristic, in order to construct a feasible pair of job sequences.

- For the nearest-neighbor heuristic (dubbed HANDSHAKE-HEU-NN within our computational study), we build a sequence of jobs for crane c by consecutively appending jobs to n_c following the nearest-neighbor approach, i.e., we append the job adding the least empty travel. Afterwards, we determine a sequence for crane $3 - c$, again, following the nearest-neighbor approach. While Gharehgozli et al. (2017) do not consider stacking sequences, we modify their 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 once with $c = 1$ and once with $c = 2$ for both strategies determining stacking positions and choose the best among the resulting schedules.
- The local search heuristic (dubbed HANDSHAKE-HEU-LS within our computational study) applies the solution obtained with HANDSHAKE-HEU-NN as its initial solution. Then, for 10,000 iterations, we choose two jobs randomly in each sequence and swap them. Note that in Gharehgozli et al. (2017), the authors merely execute 1000 iterations. In our study, however, increasing the number of iterations improved the performance considerably, while the effect on runtimes is minor. The swap is accepted whenever the pair of resulting sequences is feasible and the makespan (evaluated by using the routing approach in Sect. 4.1.2) decreases.

Deviating from Gharehgozli et al. (2017), the remaining fourth part of the decision is then determined optimally by the routing approach from Sect. 4.1.2.

B.2 A heuristic for any-bay handover

In this section, we present a heuristic optimization procedure for any-bay handover in which the cranes are allowed to intermediately store a container in every free stacking position within the yard. It is based on the approach presented by Briskorn et al. (2016). The authors tackle a setting where containers exclusively arrive at the seaside access point and need to be stored within the yard. Hence, the containers considered are only those in I^i in our setting. 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 of containers within a bay are considered. Initially, containers are picked up by the seaside crane 1, which

is allowed to preempt a job and hand it over to the landside crane 2. The authors aim at minimizing the makespan and develop—among others—a bucket-brigade scheduling approach (see e.g., Bartholdi & Eisenstein, 1996) that obtains near-optimal results. According to the bucket-brigade protocol, crane 1 hands over a container to crane 2 whenever the cranes meet and crane 2 is not carrying a container. Afterwards, crane 2 transports the container to its final destination and then heads back to toward crane 1 in order to receive the next container.

For the approach proposed in this section, we neglect capacities and precedence relations among jobs in the same position and allow containers to be intermediately stored in any position. Obviously, it does not yield feasible solutions for the TCSP and serves as a benchmark only. Hence, we explain the approach only briefly. Clearly, the optimal schedule to this setting constitutes a lower bound to the TCSP. As opposed to Briskorn et al. (2016), we consider two types of containers, namely in I^i and I^o, as well as 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 determining all parts of the decision in advance and only decide about the sequence $\overline{n}_1$ in which crane 1 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 2. Such a decision is made at the beginning of the planning horizon and every time crane 2 sets down a container.

When deciding about the next container to be handled by crane 2, we consider relocating containers in I^o and receiving a container in I^i from crane 1 as options. For receiving a container from crane 1 following the original approach from Briskorn et al. (2016), we assume that crane 2 approaches crane 1 as fast as possible and receives the first container that crane 1 can hand over. For relocating containers in I^o, we consider all boxes that crane 1 has not yet transported.

We evaluate each of these options according to the benefit of crane 1, that is, the reduction in workload of crane 1 by crane 2 carrying a container, as compared to the cost of crane 2, that is, the empty travel time for crane 2 to approach the container. Each option is evaluated by benefit minus costs, 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 box has been relocated by crane 2. 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 1 can actually gain a benefit. Hence, for two consecutive containers i and i' in $\overline{n}_1$, focusing only on bay positions, we say that

- crane 1 is allowed to hand over $i \in I^i$ only if $\overline{o}_{i'}^b < \overline{d}_i^b$ holds, and
- crane 2 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 1 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 1 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 down when the cranes are positioned at most $p + 1$ bays apart. While carrying a container, crane 2 is prioritized in the case of a conflict, while crane 1 is prioritized otherwise. The cranes position their trolleys such that they do not extend the time necessary for the other crane to transport a container to its final destination. The makespan of a hereby determined schedule equals then the point of time when all containers in $\overline{n}_1$ are dropped off in their destination positions.

Let us now conclusively detail how we determine $\overline{n}_1$. We do so by employing a simulated annealing approach resembling the one in Briskorn et al. (2016). We set the initial temperature T to three times the workload assuming that only crane 1 handles containers. In each iteration, we generate $|I^i| + |I^o|$ neighboring solutions for the current sequence $\overline{n}_1$ 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 ANY-BAY within our computational study.

C MIP model

A MIP model representing TCSP, has to account for collision avoidance, the trolley position of cranes over time, temporal interdependencies between drop-off and pickup operations in the handshake area, and capacities in the handshake area. While collision avoidance has been handled in multiple other models, e.g., in Briskorn and Zey (2020), the decision about the intermediate stacking position and integrated tracking of trolley positions, loads of stack capacities, and sequences of operations in the handshake area have (to the best of the authors' knowledge) not yet been covered by MIP models in the literature. This results in multiple new types of binary variables representing, e.g., the stacking position of a container and the time at which a container is removed from the handshake area. We summarize the notation used in the MIP model in Table 6.

Table 6 Notation

Symbol	Notation
C_{max}	Makespan
I_c	Set of containers that need to be handled by crane c with $I_1 = I$ and $I_2 = I \setminus I^{i,s} \cup I^{o,s}$
$p^b_{c,t}$	Continuous variable, bay position of crane c at time t
$p^r_{c,t}$	Continuous variable, row position of crane c at time t
$x^o_{t,i,c}$	Binary variable, equals 1 if and only if crane c completes lifting container $i \in I_c$ in t
$x^d_{t,i,c}$	Binary variable, equals 1 if and only if crane c completes releasing container $i \in I_c$ in t
$a_{i,r}$	Binary variable, equals 1 if and only if container $i \in I_2$ is stored in row r
$y_{i,i',c}$	Binary variable, equals 1 if and only if container $i \in I_2$ is transported before container $i' \in I_c$ by crane c
$c^+_{t,r}$	Continuous variable, equals 1 if storing a container in (b^h, r) is completed at time t
$c^-_{t,i,r}$	Binary variable, equals 0 if retrieval of container i from (b^h, r) is not completed at time t
$z_{i,i'}$	Continuous variable, equals 1 if container $i' \in I_2$ is intermediately stored in the same stack as $i \in I_2$ but after i has been retrieved
$u_{r,t}$	Continuous variable, number of containers stored in (b^h, r) at time t
$b^o_{i,c}, b^d_{i,c}$	Bay position where c picks up/releases $i \in I_c$
$r^o_{i,c}, r^d_{i,c}$	Row position where c picks up/releases $i \in I_c$ outside the handshake area
$\hat{I}^h$	Set of triples implying operations in the handshake area
$\hat{I}$	Set of triples implying operations outside of the handshake area

Sets

$$\hat{I}^h := \{(i,d,1),(i,o,2) \mid i \in I^{i,l}\} \cup \{(i,o,1),(i,d,2) \mid i \in I^{o,l}\}$$

and

$$\hat{I} := \{(i,o,2),(i,d,1) \mid i \in I^{o,l}\} \cup \{(i,d,2),(i,o,1) \mid i \in I^{i,l}\} \cup \{(i,q,1) \mid i \in I^{i,s} \cup I^{o,s}, q \in \{o,d\}\}$$

contain triples with the first entry referring to a container, the second entry referring to the type of operation with $q = o$ for a pickup and $q = d$ for a drop-off, and the third entry referring to the crane involved. The MIP model can then be formulated as objective function (2) and constraints (3) to (45).

$$\text{Min } C_{max} \tag{2}$$

$$C_{max} \geq \sum_{t=1}^{T} x^d_{t,i,c} \cdot t \quad \forall c \in \{1,2\}, i \in I_c \tag{3}$$

$$\sum_{t=p}^{T} x^q_{t,i,c} = 1 \quad \forall c \in \{1,2\}, q \in \{o,d\}, i \in I_c \tag{4}$$

$$\sum_{t=p}^{T} t \cdot x^o_{t,i,c} \leq \sum_{t=2p}^{T} (t-p) \cdot x^d_{t,i,c} \quad \forall c \in \{1,2\}, i \in I_c \tag{5}$$

$$\sum_{t'=t}^{\min\{t+p-1,T\}} \sum_{i \in I_c} \sum_{q \in \{o,d\}} x^q_{t',i,c} \leq 1 \quad \forall t = 1,\ldots,T, c \in \{1,2\} \tag{6}$$

$$\sum_{t'=1}^{t} \sum_{i \in I_c} (x^o_{t',i,c} - x^d_{t',i,c}) \leq 1 \quad \forall t = 1,...,T, c \in \{1,2\} \tag{7}$$

$$p^b_{1,t} \leq p^b_{2,t} - 1 \quad \forall t = 1,...,T \tag{8}$$

$$p^b_{c,0} = b^0_c \quad \forall c \in \{1,2\} \tag{9}$$

$$p^r_{c,0} = r^0_c \quad \forall c \in \{1,2\} \tag{10}$$

$$p^k_{c,t} - p^k_{c,t-1} \leq 1 \quad \forall c \in \{1,2\}, t = 1,...,T, k \in \{b,r\} \tag{11}$$

$$p^k_{c,t-1} - p^k_{c,t} \leq 1 \quad \forall c \in \{1,2\}, t = 1,...,T, k \in \{b,r\} \tag{12}$$

$$B + (b^q_{c,i} - B) \left(\sum_{t'=t}^{\min\{t+p,T\}} x^q_{t',i,c} \right) \geq p^b_{c,t} \quad \forall t = 1,\ldots,T, (i,q,c) \in \hat{I}^h \cup \hat{I} \tag{13}$$

$$b^q_{c,i} \cdot \left(\sum_{t'=t}^{\min\{t+p,T\}} x^q_{t',i,c} \right) \leq p^b_{c,t} \quad \forall t = 1,...,T, (i,q,c) \in \hat{I}^h \cup \hat{I} \tag{14}$$

$$R + (r^q_{c,i} - R) \cdot \left(\sum_{t'=t}^{\min\{t+p,T\}} x^q_{t',i,c} \right) \geq p^r_{c,t} \quad \forall t = 1,...,T, (i,q,c) \in \hat{I} \tag{15}$$

$$1 + (r^q_{c,i} - 1) \cdot \left(\sum_{t'=t}^{\min\{t+p,T\}} x^q_{t',i,c} \right) \leq p^r_{c,t} \quad \forall t = 1,...,T, (i,q,c) \in \hat{I} \tag{16}$$

$$R \cdot \left(1 - \sum_{t'=t}^{\min\{t+p,T\}} x^q_{t',i,c} \right) + \sum_{r=1}^{R} r \cdot a_{i,r} \geq p^r_{c,t} \quad \forall t = 1,...,T, (i,q,c) \in \hat{I}^h \tag{17}$$

$$R \cdot \left(\sum_{t'=t}^{\min\{t+p,T\}} x^q_{t',i,c} - 1 \right) + \sum_{r=1}^{R} r \cdot a_{i,r} \leq p^r_{c,t} \quad \forall t = 1,...,T, (i,q,c) \in \hat{I}^h \tag{18}$$

$$\sum_{(b^h,r) \in H_1} a_{i,r} = 1 \quad \forall i \in I^{i,l} \tag{19}$$

$$\sum_{(b^h,r) \in H_2} a_{i,r} = 0 \quad \forall i \in I^{i,l} \tag{20}$$

$$\sum_{(b^h,r) \in H_2} a_{i,r} = 1 \quad \forall i \in I^{o,l} \tag{21}$$

$$\sum_{(b^h,r) \in H_1} a_{i,r} = 0 \quad \forall i \in I^{o,l} \tag{22}$$

$$\sum_{t=p}^{T} t \cdot x^d_{t,i,1} \leq \sum_{t=p}^{T} t \cdot x^o_{t,i,2} \quad i \in I^{i,l} \tag{23}$$

$$\sum_{t=p}^{T} t \cdot x^d_{t,i,2} \leq \sum_{t=p}^{T} t \cdot x^o_{t,i,1} \quad i \in I^{o,l} \tag{24}$$

$$\sum_{t=p}^{T} t \cdot x^{o}_{t,i',c} - \sum_{t=p}^{T} t \cdot x^{o}_{t,i,c} \leq y_{i,i',c} \cdot T$$
$$\forall c \in \{1,2\}, i, i' \in I_2, i \neq i' \tag{25}$$

$$\sum_{t=p}^{T} t \cdot x^{o}_{t,i,c} - \sum_{t=p}^{T} t \cdot x^{o}_{t,i',c} \leq (1 - y_{i,i',c}) \cdot T$$
$$\forall c \in \{1,2\}, i, i' \in I_2, i \neq i' \tag{26}$$

$$y_{i,i',1} + y_{i,i',2} + a_{i,r} + a_{i',r} \leq 3 + z_{i,i'}$$
$$\forall (b^h, r) \in H_1, i, i' \in I^{i,l}, i \neq i' \tag{27}$$

$$y_{i,i',1} + y_{i,i',2} + a_{i,r} + a_{i',r} \leq 3 + z_{i,i'}$$
$$\forall (b^h, r) \in H_2, i, i' \in I^{o,l}, i \neq i' \tag{28}$$

$$\sum_{t=p}^{T} t \cdot x^{o}_{t,i,2} \leq (1 - z_{i,i'}) \cdot T + \sum_{t=p}^{T} t \cdot x^{d}_{t,i',1}$$
$$\forall i, i' \in I^{i,l}, i \neq i' \tag{29}$$

$$\sum_{t=p}^{T} t \cdot x^{o}_{t,i,1} \leq (1 - z_{i,i'}) \cdot T + \sum_{t=p}^{T} t \cdot x^{d}_{t,i',2}$$
$$\forall i, i' \in I^{o,l}, i \neq i' \tag{30}$$

$$x^{d}_{t,i,1} + a_{i,r} \leq 1 + c^{+}_{t,r}$$
$$\forall t = 1, \ldots, T, (b^h, r) \in H_1, i \in I^{i,l} \tag{31}$$

$$x^{d}_{t,i,2} + a_{i,r} \leq 1 + c^{+}_{t,r}$$
$$\forall t = 1, \ldots, T, (b^h, r) \in H_2, i \in I^{o,l} \tag{32}$$

$$c^{-}_{t,i,r} \leq 0.5 \cdot (x^{o}_{t,i,2} + a_{i,r})$$
$$\forall t = 1, \ldots, T, (b^h, r) \in H_1, i \in I^{i,l} \tag{33}$$

$$c^{-}_{t,i,r} \leq 0.5 \cdot (x^{o}_{t,i,1} + a_{i,r})$$
$$\forall t = 1, \ldots, T, (b^h, r) \in H_2, i \in I^{o,l} \tag{34}$$

$$u_{r,t} = u_{r,t-1} + c^{+}_{t,r} - \sum_{i \in I_2} c^{-}_{t,i,r}$$
$$\forall r = 1, \ldots, R, t = 1, \ldots, T \tag{35}$$

$$u_{r,0} = 0 \forall r = 1, \ldots, R \tag{36}$$

$$u_{r,t} \leq C^{h}_{r} \forall r = 1, \ldots, R, t = 1, .., T \tag{37}$$

$$x^{q}_{t,i,c} \in \{0,1\}$$
$$\forall t = 0, ..., T, i \in I_c, q \in \{o,d\}, c \in \{1,2\} \tag{38}$$

$$0 \leq c^{+}_{t,r} \leq 1 \forall t = 1, \ldots, T, r = 1, \ldots, R \tag{39}$$

$$c^{-}_{t,i,r} \in \{0,1\} \forall t = 1, \ldots, T, r = 1, \ldots, R, i \in I_2 \tag{40}$$

$$u_{r,t} \geq 0 \forall t = 1, \ldots, T, r = 1, \ldots, R \tag{41}$$

$$y_{i,i',c} \in \{0,1\} \forall c \in \{1,2\}, i, i' \in I_2, i \neq i' \tag{42}$$

$$0 \leq z_{i,i'} \leq 1 \forall i, i' \in I^{i,l}, i \neq i' \tag{43}$$

$$0 \leq z_{i,i'} \leq 1 \forall i, i' \in I^{o,l}, i \neq i' \tag{44}$$

$$a_{i,r} \in \{0,1\} \forall i \in I_2, r = 1, \ldots, R \tag{45}$$

Equation (3) bounds the makespan from below, while (4) ensures that each operation is carried out exactly once. In (5), we enforce each container to be picked up before it is dropped off, while (6) ensures that a crane can complete at most one operation in each time interval of length p. Constraint (7) prohibits each crane from having more than one container picked up at a time.

Constraint (8) implements the non-interference constraints, while (9) to (12) account for initial positions and travel speed restricting the position of both cranes' gantries and trolleys over time.

Constraints (13) to (18) ensure that each crane stays in the correct position while picking up or dropping off a container. Here, (13) and (14) are concerned with the bay the gantry is positioned in, while for operations outside the handshake area, (15) and (16) are concerned with the row. Recall that positions of operations outside the handshake area are given as the origin position or the destination position of a container. The row position for picking up or dropping off container i in the handshake area depends on the stack it is intermediately stored in and, thus, (17) and (18), concerned with the row for those operations, take the corresponding decision variables $a_{i,1}, \ldots, a_{i,R}$ into account. These variables reflect the stack assignment to containers in I_2, which is ensured by (19) to (22).

Constraints (23) to (30) represent the precedence constraints for operations in the handshake area. First, (23) and (24) ensure that each container is retrieved from the handshake area after being dropped off there. Second, variable $y_{i,i',c}$ equals 1 if and only if container i is handled before container i' by crane c due to (25) and (26). Constraints (27) and (28) force variable $z_{i,i'}$ to take a value of 1 if both i and i' are stored in the same stack and if i is dropped off and retrieved first. Due to the proof of Lemma 1, i and i' are not in the stack at the same time, and thus i is retrieved before i' is dropped off. The latter is implemented by (29) and (30).

Constraints (31) to (37) account for the stack capacities. Variable $c^{+}_{t,r}$ is forced to take a value of 1 if a container in $I^{i,l} \cup I^{o,l}$ is stored in position (b^h, r) at time t by (31) and (32). Furthermore, $c^{-}_{t,i,r}$ can take a value of 1 only if container i is retrieved from position (b^h, r) at time t. Then, $u_{r,t}$ constitutes an upper bound of the load of position (b^h, r) over time due to (35) and (36). Finally, (37) ensures it does not exceed the capacity.

The domains of the decision variables are defined in (38) to (45). Note that variables $z_{i,i'}$ and $c^{+}_{t,r}$ essentially function as binary variables, but we do not need to define them as such.

References

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

Bierwirth, C., & Meisel, F. (2015). A follow-up survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research, 244*(3), 675–689.

Boysen, N., Briskorn, D., & Emde, S. (2015). A decomposition heuristic for the twin robots scheduling problem. *Naval Research Logistics, 62*(1), 16–22.

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

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

Boysen, N., Fliedner, M., Jaehn, F., & Pesch, E. (2013). A survey on container processing in railway yards. *Transportation Science, 47*(3), 312–329.

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

Briskorn, D. (2021). Routing of two stacking cranes with predetermined container sequences. *Journal for Scheduling, 24*(3), 367–380.

Briskorn, D., & Angeloudis, P. (2016). Scheduling co-operating stacking cranes with predetermined container sequences. *Discrete Applied Mathematics, 201,* 70–85.

Briskorn, D., Emde, S., & Boysen, N. (2016). Cooperative twin-crane scheduling. *Discrete Applied Mathematics, 211,* 40–57.

Briskorn, D., Jaehn, F., & Wiehl, A. (2019). A generator for test instances of scheduling problems concerning cranes in transshipment terminals. *OR Spectrum, 41,* 45–69.

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

Briskorn, D., & Zey, L. (2020). Interference aware scheduling of triple-crossover-cranes. *Journal of Scheduling, 23*(4), 465–485.

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

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

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

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

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

Carlo, H. J., & Vis, I. F. A. (2010). New initiatives in stacking crane configurations. *Port Technology International, 44,* 32–37.

Dell, R. F., Royset, J. O., & Zyngiridis, I. (2009). Optimizing container movements using one and two automated stacking cranes. *Journal of Industrial and Management Optimization, 5,* 285–302.

Dorndorf, U., & Schneider, F. (2010). Scheduling automated triple cross-over stacking cranes in a container yard. *OR Spectrum, 32,* 617–632.

Ehleiter, A., & Jaehn, F. (2018). Scheduling crossover cranes at container terminals during seaside peak times. *Journal of Heuristics, 24,* 1–34.

Emde, S., & Boysen, N. (2014). One-dimensional vehicle scheduling with a front-end depot and non-crossing constraints. *OR Spectrum, 36*(2), 381–400.

Galle, V., Barnhart, C., & Jaillet, P. (2018). Yard crane scheduling for container storage, retrieval, and relocation. *European Journal of Operational Research, 271*(1), 288–316.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability—A guide to the theory of NP-completeness*. W.H. Freemand and Company.

Gharehgozli, A. H., Laporte, G., Yu, Y., & de Koster, R. (2014a). Scheduling twin yard cranes in a container block. *Transportation Science, 49*(3), 686–705.

Gharehgozli, A. H., Vernooij, F. G., & Zaerpour, N. (2017). 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.

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

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

He, J., Huang, Y., & Yan, W. (2015). Yard crane scheduling in a container terminal for the trade-off between efficiency and energy consumption. *Advanced Engineering Informatics, 29*(1), 59–75.

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

Jaehn, F., & Kress, D. (2018). Scheduling cooperative gantry cranes with seaside and landside jobs. *Discrete Applied Mathematics, 242,* 53–68.

Kellner, M., & Boysen, N. (2015). RMG vs. DRMG: An evaluation of different crane configurations in intermodal transshipment yards. *EURO Journal on Transportation and Logistics, 4*(3), 355–377.

Kemme, N. (2012). Effects of storage block layout and automated yard crane systems on the performance of seaport container terminals. *OR Spectrum, 34*(3), 563–591.

Kovalyov, M. Y., Pesch, E., & Ryzhikov, A. (2018). A note on scheduling container storage operations of two non-passing stacking cranes. *Networks, 71*(3), 271–280.

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

Lehnfeld, J., & Knust, S. (2014). Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. *European Journal of Operational Research, 239*(2), 297–312.

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

Schuler, M. (2019). World's largest containership completes maiden voyage from China to Europe. https://gcaptain.com/worlds-largest-containership-completes-maiden-voyage-from-china-to-europe/. Accessed February 2020.

Speer, U., & Fischer, K. (2017). Scheduling of different automated yard crane systems at container terminals. *Transportation Science, 51*(1), 305–324.

Stahlbock, R., & Voß, S. (2008). Operations research at container terminals: A literature update. *OR Spectrum, 30,* 1–52.

Steenken, D., Voß, S., & Stahlbock, R. (2004). Container terminal operations and operations research—A classification and literature review. *OR Spectrum, 26,* 3–49.

Vis, I. F. A., & de Koster, R. (2003). Transshipment of containers at a container terminal: An overview. *European Journal of Operational Research, 147,* 1–16.

XiaoLong, H., Qianqian, W., & JiWei, H. (2019). Scheduling cooperative twin automated stacking cranes in automated container terminals. *Computers & Industrial Engineering, 128,* 553-558.