European Journal of Operational Research 279 (2019) 732–750

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Straddle carrier routing at seaport container terminals in the presence of short term quay crane buffer areas

Dominik Kress[a,*], Sebastian Meiswinkel[a], Erwin Pesch[a,b]

[a] Management Information Science, University of Siegen, Kohlbettstr. 15, Siegen 57068, Germany
[b] HHL Leipzig, Center for Advanced Studies in Management, Jahnallee 59, Leipzig 04109, Germany

ARTICLE INFO

Article history:
Received 8 August 2018
Accepted 12 June 2019
Available online 18 June 2019

Keywords:
Scheduling
Container logistics
Seaport logistics
Vehicle routing
Vehicle dispatching

ABSTRACT

We address an optimization problem that arises at seaports where containers are transported between stacking areas and small buffer areas of restricted capacity that are located within the reach of quay cranes. The containers are transported by straddle carriers that have to be routed such that given unloading and loading sequences of the containers at the quay cranes are respected. The objective is to minimize the turnaround times of the vessels. We analyze the problem's computational complexity, present an integer program, and propose a heuristic framework that is based on decomposing the problem into its routing component and a component that handles the time variables and buffer capacities. The framework is analyzed in computational tests that are based on real-world data. Based on these tests, we analyze the question of whether or not it pays off to deviate from the approach of permanently assigning a fixed number of straddle carriers to each quay crane, which is the strategy that is currently implemented at the port.

1. Introduction

A container port is a complex system consisting of berths equipped with quay cranes, transport vehicles, stacking areas, stacker cranes, and road or rail connections to the hinterland. Each container port is a unique combination of these components and features an individual vehicle fleet. Large ports handle several million twenty-foot equivalent units (TEU) on an annual basis. Port authorities therefore strive for sophisticated planning approaches based on simulation or optimization techniques in order to stay competitive.

Operations research challenges at seaports are mostly concerned with problem settings that directly or indirectly affect the flow of containers within the ports. Container ports that can serve several vessels simultaneously aim at high berth utilization rates, so that decision makers have to determine appropriate assignments of vessels to berths. Each berth is equipped with one or multiple *quay cranes*. These cranes are needed for the process of loading and unloading containers and have to be scheduled appropriately. Inbound containers, i.e., containers that arrive by vessels, must then be transported to large stacking (or storage) areas, where they are temporarily stored for later processing by train, truck or vessel. Outbound containers, i.e., containers that arrive by train or truck and that have to be loaded onto a vessel, have to be transported in the reverse direction. The corresponding transportation requests within the seaport can be executed by different types of vehicles. *Automated guided vehicles* (AGVs) and manually driven *yard trucks* (YTs) cannot lift or drop containers. They have to be loaded and unloaded at predefined handover positions by quay cranes at the vessels or by yard cranes (or gantry cranes) at the storage areas. *Automated lifting vehicles* (ALVs) are able to perform lifting and dropping operations. However, yard cranes remain necessary for stacking operations. *Straddle carriers* (SCs) and *reach stackers* (RSs) represent the most flexible solutions as these vehicles can perform all necessary container operations, i.e., lifting, dropping, and stacking, without the need for additional cranes. RSs, however, are usually only applied in small and medium sized ports.

Especially in peak times, the major objective of port authorities is the minimization of dwell times of vessels at the berths (see also Jaehn & Kress, 2018; Kovalyov, Pesch, & Ryzhikov, 2018; Kress, Dornseifer, & Jaehn, 2019; Nossack, Briskorn, & Pesch, 2018). A key factor to achieving this objective is an efficient usage of quay cranes (see, e.g., Goodchild & Daganzo, 2007), so that the problem of effectively planning the transportation processes of

* Corresponding author.
E-mail addresses: dominik.kress@uni-siegen.de (D. Kress), sebastian.meiswinkel@uni-siegen.de (S. Meiswinkel), erwin.pesch@uni-siegen.de (E. Pesch).

https://doi.org/10.1016/j.ejor.2019.06.028
0377-2217/

containers between quay cranes and storage areas is of specific importance. Our research (see also Meiswinkel, 2018) is motivated by a real-world setting at a container port in Germany, where SCs are used for transportation and storage (lifting, dropping, restacking) operations. *Buffer areas* of limited capacity (four containers) at the quay cranes allow the intermediate (short term) storage of containers that have been unloaded from a vessel or that have been dropped by a SC and that are waiting to be loaded onto a vessel. The current approach at the considered port is to permanently assign three to five SCs to each quay crane. This, however, restricts the potential of effectively routing the SCs, which is a prerequisite for high quay crane utilization rates and, consequently, short dwell times of the vessels. In this article, we will therefore analyze if relaxing this constraint is beneficial.

1.1. Related literature

General survey articles on the vast amount of literature related to logistic operations at container terminals and to ocean container transport in general are given by Steenken, Voß, and Stahlbock (2004) and Lee and Song (2017), respectively. When it comes to seaport operations, a major focus of the literature lies on the aforementioned berth allocation problem, the scheduling of quay cranes, and on transportation operations within the ports. Comprehensive overviews of the relevant literature regarding these problem settings are given by Bierwirth and Meisel (2010, 2015) and Stahlbock and Voß (2008). Another recent literature overview by Lehnfeld and Knust (2014) focuses on problems of loading, unloading and premarshalling of stacks in storage areas and combined problem settings. Carlo, Vis, and Roodbergen (2014a) survey literature on storage yard operations. Kuzmicz and Pesch (2019) present literature on approaches concerning the repositioning of empty containers between Europe and China.

All of the aforementioned survey articles include at least some articles related to transport operations and vehicle routing problems in container terminals, i.e., the field of study in the article at hand. Detailed and more focused reviews on these topics are provided by Carlo, Vis, and Roodbergen (2014b) and Stahlbock and Voß (2008). Hence, for the sake of brevity, we refer the reader to these articles for a detailed overview of the field and restrict ourselves to presenting only the most relevant literature in the context of our problem setting in the remainder of this section. In doing so, we categorize the publications based on the types of vehicles that are considered by the respective authors, as these types strongly affect the way that buffer areas are implemented at the considered ports.

In case of vehicles that cannot perform lifting and dropping operations themselves, e.g., AGVs or YTs, the nature of buffer areas is profoundly different from our setting as they will always have to be implemented by (loaded or unloaded) vehicles that wait to be served by quay cranes or yard cranes. In contrast to our setting, there is direct interaction between cranes and vehicles, so that specialized solution approaches for these settings are usually not directly applicable to our case. However, there are some insights, analogies, and problem characteristics that are relevant to the problem at hand and related directions for future research. Bish et al. (2005), for example, consider a setting with AGVs and a single quay crane that solely performs loading or unloading operations. They prove that a simple greedy algorithm solves their single crane problem to optimality. This result will most probably also hold for our case. Based on the greedy algorithm, Bish et al. (2005) then present a heuristic approach for the case of multiple cranes. Other relevant papers include Bish (2003), who focusses on the problem of effectively routing AGVs and scheduling the related quay crane operations. The author considers multiple quay cranes that perform both unloading and loading operations. The decision of selecting a storage location out of a set of potential candidate locations for each unloaded container is integrated into the problem. A heuristic method based on formulating the problem as a transshipment problem is presented. Angeloudis and Bell (2010) focus on a setting with various conditions of uncertainty. They develop a dispatching approach with a planning horizon of two container movements per AGV and perform simulation experiments to compare the approach with other heuristics. Briskorn, Drexl, and Hartmann (2006) transform an AGV routing problem with the objective of minimizing the weighted sum of earliness, tardiness and empty travel time into an inventory based formulation and present an exact algorithm. Kim and Bae (2004) present a MIP formulation and a heuristic approach for the dispatching and routing of AGVs with the objective of minimizing the waiting time of cranes as well as the total travel distance of the vehicles. Grunow, Günther, and Lehmann (2004) consider AGVs that are able to carry two containers at a time. They present a priority rule based approach in order to minimize the total lateness. Similarly, Grunow, Günther, and Lehmann (2006) consider AGVs that are able to carry either two small-sized containers or one large-sized container. The authors analyze the performance of this setting in comparison to a setting where only one (either small- or large-sized) container can be transported by each AGV. Lee, Cao, Shi, and Chen (2009) analyze a port that uses YTs and integrate the scheduling of these vehicles with the problem of allocating containers to storage blocks. Each container has a time window with a hard lower bound and a soft upper bound. The authors focus on minimizing the total delay with respect to these upper bounds and the total travel time of the YTs. Ng, Mak, and Zhang (2007) consider YTs for the transportation of containers between yard cranes and quay cranes. A set of genetic algorithms that minimize the makespan is presented and compared.

In comparison to the aforementioned stream of research, the scheduling of vehicles that can lift and drop containers themselves, e.g., SCs or ALVs, is more flexible, as these vehicles will not have to wait until the containers are taken over by the quay cranes. Here, buffer areas allow the short term storage of containers without the interaction of any vehicle or crane, so that the horizontal transport of containers by vehicles is decoupled from the crane operations, which is the perspective taken in this paper. In this context, Nguyen and Kim (2009) consider a setting that takes account of buffer areas at the quay cranes and that is fairly similar to ours. While we consider SCs, they assume that the transportation of containers is performed by ALVs. They do not incorporate the yard crane schedules into their model, but rather assume that the ALVs spend a specific time period at the storage areas to retrieve or store the containers. Hence, they abstract from the explicit incorporation of storage area operations, which we aim to include into our model. Specifically, we include the possibility to split the processing of restacking operations and the retrieving and storage operations of the associated containers (that have to be loaded onto or unloaded from a vessel) among multiple vehicles, rather than assigning just one vehicle to all of these operations. Additionally, different from our model, Nguyen and Kim (2009) do not consider safety time considerations in order to avoid collisions of the vehicles. They develop a heuristic approach that is based on a procedure that heuristically converts buffer constraints into time window constraints. In contrast to our approach, it imposes fairly restrictive assumptions with respect to the sequence of the containers that are processed by the ALVs. Furthermore, it is less flexible with respect to the incorporation of the setting at the beginning of the planning horizon, so that the authors leave the evaluation of their approach in a dynamic environment for future research. Other relevant articles include Böse, Reiners, Steenken, and Voß (2000), who consider a setting with a fixed number of SCs and multiple quay cranes that feature

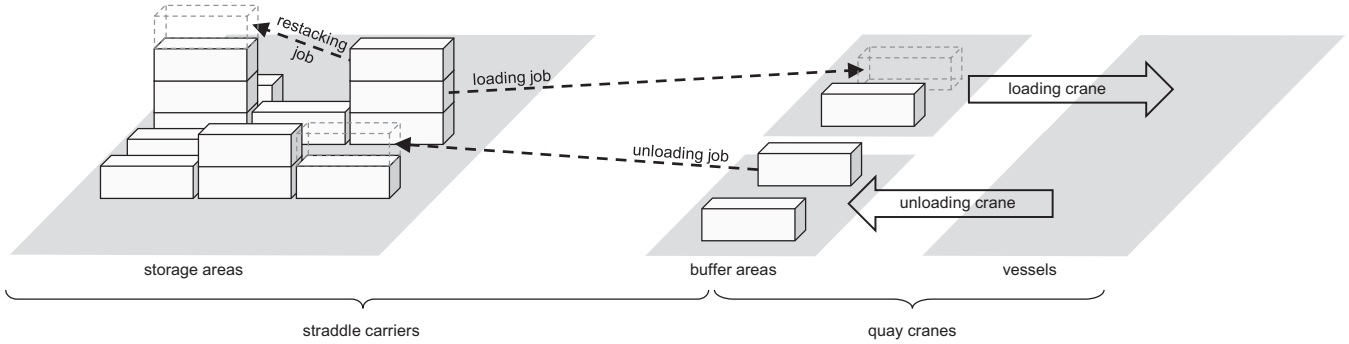

Fig. 1. Problem setting.

buffers of unrestricted capacity. The authors present and evaluate evolutionary algorithms. Kress, Meiswinkel, and Pesch (2015) describe and analyze a combined partitioning and matching problem that can be used to model the transportation of containers by RSs in small to medium sized ports that feature a small temporary storage area in addition to the main stacking areas. They present a heuristic framework that is based on decomposing the problem into its matching and partitioning components. Skinner et al. (2013) develop a genetic algorithm for routing SCs in a fairly specific setting at a container terminal in Australia. In contrast to the other studies, they do not only take account of the transportation of containers between stacking areas and the quay cranes but also consider transportation requests between stacking areas and hinterland connections. Kim and Kim (1999) present a routing problem with a mixed fleet of SCs and YTs in a setting with multiple quay cranes that solely have to perform loading operations. They present and evaluate a beam search algorithm.

1.2. Problem setting and contribution

As outlined above, we consider a setting where SCs are used to transport containers between storage areas and quay cranes that are equipped with buffer areas of limited capacity. This is illustrated in Fig. 1.

Because containers vary significantly in weight and the load of vessels has to be balanced due to safety restrictions, we assume that the sequences of containers that have to be loaded onto or unloaded from the vessels by the quay cranes are determined in a higher level optimization problem and can be considered as given. Moreover, as frequently assumed in the literature, the origin or destination location in the storage areas is assumed to be known in advance for each container. In line with this assumption, we assume that the movement of a given container to, from, or within the storage areas cannot interfere with a lifting or dropping operation of another container within the storage areas, e.g., by permanently “blocking” its corresponding container stack.

A container port is a highly dynamic environment, in which frequent rescheduling of operations is needed. We therefore assume the container sequences to be rather short and consider each quay crane either as a *loading crane* that solely loads containers onto a vessel or as an *unloading crane* that solely unloads containers from a vessel (see Fig. 1). Hence, we do not consider crane double cycling (see, for example, Goodchild & Daganzo, 2007), i.e., the combined processing of loading and unloading operations in single crane cycles (movements from the buffer to the vessel and back or vice versa). Furthermore, at the beginning of the planning horizon, the SCs may be located anywhere in the port and the cranes’ buffer areas may not be empty. SCs and quay cranes may also not be immediately available because they may be in the process of handling a container.

The task of picking up a container at its origin and moving it to its destination by a SC is henceforth referred to as a job. As each job is associated with a distinct container, we will sometimes refer to a job by its corresponding container and vice versa. The jobs can be classified according to the origin and destination of the respective containers (see Fig. 1). *Loading jobs* relate to containers that originate at storage areas and have to be transported to a buffer, where they are loaded onto a vessel by a quay crane. Similarly, *unloading jobs* relate to containers that have to be transported from a buffer to their storage location upon having been unloaded from a vessel. *Restacking jobs* correspond to container movements within the storage areas. They are taken into account because of stacking restrictions in the storage area. It may, for example, be necessary to restack containers in order to be able to access specific storage locations of loading or unloading jobs. We assume that each restacking job is related to a distinct loading or unloading job. Each of these loading or unloading jobs may be related to multiple restacking jobs, all of which relate to the same stack in a storage area and have to be processed in a given sequence in order to be able to access the corresponding container or its designated storage position. These sequences are assumed to be rather short, because many restacking operations are performed in off-peak times in order to be able to quickly access containers in peak times. Nevertheless, the fact that the processing of restacking jobs and their associated loading or unloading jobs may be split among multiple SCs may result in significant time savings when compared to the situation in which only one SC processes all corresponding jobs (see Section 1.1). This is because it allows a container that corresponds to a loading job to potentially be lifted before the processing of the associated restacking containers is completed or to start the processing of a container that has been unloaded from a vessel before the corresponding restacking jobs have been lifted. Note that, in line with our research question, we do not assume the assignment of SCs to cranes to be fixed, so that - in contrast to the cranes - SCs are allowed to process a mix of loading, unloading, and restacking jobs.

As we address questions of rather general nature, the incorporation of the details of the port layout, e.g., driving lanes or the layout of storage blocks, is not expected to have a critical impact on our findings. Hence, for the sake of simplicity, we do not explicitly take these details into account. Instead, we assume that the SCs move on driving lanes that are arranged on a grid that spans over the port and make use of the *Manhattan metric* for the calculation of the distance between any pair of points on this grid. Additionally, in order to extract a somewhat basic problem setting, the SCs are assumed to be homogeneous and we do not take account of their acceleration or of varying speeds for loaded and unloaded movements. They are assumed to always be able to

pass each other. Furthermore, when determining distances, each buffer is assumed to be a single point, i.e., we do not differentiate between storage slots in the buffer areas. In order to avoid collisions, we assume that certain *safety times* must elapse between lifting and dropping operations of containers.

The problem is to determine a routing of the SCs, i.e., an allocation of jobs to SCs and a corresponding sequencing of the jobs of each SC, subject to all capacity restrictions and precedence relations, such that the resulting quay crane schedules allow for short turnaround times of vessels. Hence, the objective relates to avoiding idle times of the quay cranes that arise before all of the cranes' containers have been dropped in the buffer areas or loaded onto the vessels. As mentioned above, we can only consider short container sequences and we will have to reschedule regularly. We therefore consider the objective of minimizing the sum of the time instants at which quay cranes interact with their buffers, i.e., at which the process of lifting a container starts or the process of dropping a container finishes. By doing so, we aim at favoring idle times at the end of the quay crane schedules. This, in turn, allows for flexible rescheduling, which is assumed to be initiated when new container data becomes available, in case of unexpected events, or whenever one of the SCs or quay cranes has completed all of its assigned jobs or operations. Regular rescheduling, of course, requires the ability to determine high quality solutions in relatively short computational times, so that exact approaches do not seem to be a promising research direction for real-world instance sizes in light of our complexity results presented in Section 2.2.

We refer to the problem under consideration as the ***M**anhattan Metric **S**traddle **C**arrier **R**outing Problem with **B**uffer Areas* (MSCRB). To the best of the authors' knowledge, our specific setting is new to the literature. Some basic results for a variant of our setting have been presented and analyzed in the dissertation of Meiswinkel (2018). In the paper at hand, we present results on the computational complexity of MSCRB and develop a heuristic framework that is evaluated in computational tests. These tests are based on real-world data, so that we can elaborate on the question of whether or not our approach is beneficial when compared to the current practice outlined above.

1.3. Overview of this article

The remainder of this article is structured as follows. Section 2 is devoted to defining the notation used throughout the article and analyzing the computational complexity of MSCRB. Furthermore, an integer programming formulation based on an asymmetric traveling salesman problem (TSP) with precedence constraints is presented. A heuristic framework is described in Section 3. It is based on decomposing MSCRB into its routing component and a component that handles the time variables and buffer capacities. The results of our computational tests are subject of Section 4. The paper closes with a conclusion in Section 5.

2. Notation and insights

We assume that the planning horizon is divided into a finite number of intervals of equal length and refer to the length of a time interval as a time unit. All time parameters are assumed to be integral multiples of a time unit and can therefore be specified by natural numbers.

We denote the set of loading cranes by C^l, the set of unloading cranes by C^u, and the complete set of quay cranes by $C = C^l \cup C^u$, $|C| = n^q$. Each crane $c \in C$ is associated with a buffer capacity $b_c \in \mathbb{N}_{>0}$, representing the maximum number of containers that can simultaneously be stored in its buffer, a set of containers $J_c = \{j_{c,1}, \ldots, j_{c,|J_c|}\}$ that it must process, and a set J_c^b of containers that are located in the crane's buffer at the beginning of the planning horizon.

The loading process of a container (performed by some loading crane) corresponds to picking up the container in the buffer, moving it to the vessel, dropping it on the vessel, and then returning to the buffer. Similarly, the unloading process of a container (performed by an unloading crane) corresponds to moving from the buffer to the vessel, picking up the container, moving to the buffer, and dropping the container. We assume that the time needed for an unloading or loading processes is identical for all containers. It is denoted by $t^q \in \mathbb{N}_{>0}$.

Fig. 2 illustrates the container sets that are relevant for loading cranes. Dashed arrows represent container movements that are not subject to the optimization because the respective allocation decisions are fixed at the beginning of the planning horizon (see below for details). For each loading crane $c \in C^l$, the containers of the set J_c^b must still be processed, i.e., $J_c^b \subseteq J_c$. Additionally, there potentially exists a subset $J_c^v \subseteq J_c$ of containers that are handled by SCs at the beginning of the planning horizon and that will be dropped in the crane's buffer. However, J_c does not include a container, the loading process of which has already been started but has not yet been completed by the crane at the beginning of the planning horizon. We define $J_c^l := J_c^b \cup J_c^v$ for each $c \in C^l$.

For each unloading crane $c \in C^u$ (see Fig. 3), we have $J_c \cap J_c^b = \emptyset$, i.e., the containers of the set J_c^b have already been processed by the crane and are waiting to be picked up by a SC. Additionally, there may exist a container, the unloading process of which has already been started but has not yet been completed by c at the beginning of the planning horizon, which we refer to by the set J_c^q that solely includes this container in case of its existence. This container is not included in J_c as well. We define $J_c^u := J_c^b \cup J_c^q = \{j_{c,|J_c|+1}, \ldots, j_{c,|J_c|+|J_c^u|}\}$ for each $c \in C^u$. For the sake of notational convenience, we additionally define $J_c^u := \emptyset$ for all $c \in C^l$.

Each set J_c, $c \in C$, is assumed to be ordered. That is, for each pair of containers $j_{c,i}, j_{c,l} \in J_c$ with $i < l$, crane c must process $j_{c,i}$ before it processes $j_{c,l}$. We take account of the fact that crane $c \in C$ may not have finished a loading or unloading process of a container at the beginning of the planning horizon, by defining a time instant $a_c^q \in \mathbb{N}_{\geq 0}$, at which it is available for starting to lift the next container in the buffer in case of a loading crane or at which it has dropped the container in the buffer in case of an unloading crane. For all cranes $c \in C$ that are immediately available, we set $a_c^q = 0$.

Table 1 summarizes the notation regarding the quay cranes.

Based on the above definitions, the set of loading jobs is $J^l := \bigcup_{c \in C^l} \left(J_c \setminus J_c^l\right)$, while the set of unloading jobs corresponds to $J^u := \bigcup_{c \in C^u} (J_c \cup J_c^u)$ (see Figs. 2 and 3). We define $J := J^l \cup J^u$. In the context of jobs, we will usually not explicitly specify the quay cranes that must process or have processed the corresponding containers in order to ease the notation, i.e., we will write $j \in J$. Each job $j \in J$ corresponds to a container that has not been started to be processed by a SC at the beginning of the planning horizon and that must be transported between buffer areas and storage areas. It is associated with a (potentially empty) sequence of restacking jobs, represented by an ordered set $R_j = \{r_{j,1}, \ldots, r_{j,|R_j|}\}$. $R := \bigcup_{j \in J} R_j$ denotes the complete set of restacking jobs. If $R_j \neq \emptyset$ for some $j \in J$, the container related to job j may only be lifted (in case of a loading job) or dropped (in case of an unloading job) by a SC when the container related to $r_{j,|R_j|}$ has been lifted. Furthermore, for $r_{j,i}$, $r_{j,l} \in R_j$, $j \in J$, $i < l$, the container related to $r_{j,i}$ must be lifted before the container related to $r_{j,l}$ can be lifted. The underlying reasoning is illustrated in Fig. 4 for the case of a loading job which is blocked by two containers that are stored in the same stack. Naturally,

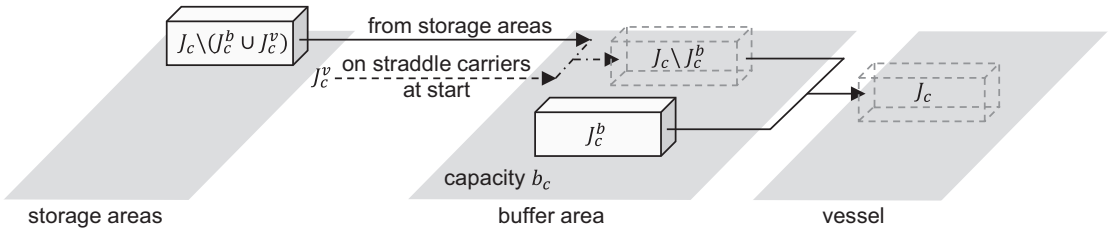

Fig. 2. Container sets associated to a loading crane.

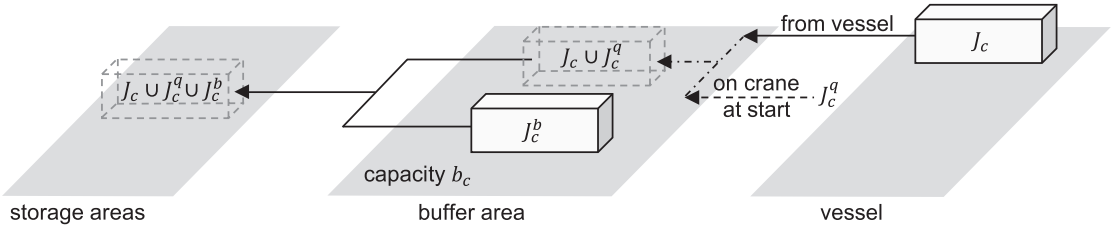

Fig. 3. Container sets associated to an unloading crane.

Table 1
Notation regarding the quay cranes.

C^l	Set of loading cranes	
C^u	Set of unloading cranes	
C	Set of quay cranes	$C = C^l \cup C^u$, $\lvert C\rvert = n^q$
t^q	Time needed to process a container by a quay crane	$t^q \in \mathbb{N}_{>0}$
a_c^q	Availability time of crane $c \in C$	$a_c^q \in \mathbb{N}_{\geq 0}$
b_c	Capacity of the buffer of crane $c \in C$	$b_c \in \mathbb{N}_{>0}$
J_c	Ordered set of containers that must be processed by crane $c \in C$	$J_c = \{j_{c,1}, \ldots, j_{c,\lvert J_c\rvert}\}$
J_c^b	Containers located in the buffer of crane $c \in C$ at the beginning of the planning horizon	
J_c^v	Containers that have to be processed by loading crane $c \in C^l$ and that are handled by a SC at the beginning of the planning horizon	
J_c^l	Subset of the containers related to loading crane $c \in C^l$	$J_c^l := J_c^b \cup J_c^v$, $J_c \cap J_c^l = J_c^l$
J_c^q	Set of at most one container that is in the process of being unloaded by unloading crane $c \in C^u$ at the beginning of the planning horizon	
J_c^u	Subset of the containers related to unloading crane $c \in C^u$	$J_c^u := J_c^b \cup J_c^q$, $J_c \cap J_c^u = \emptyset$ $J_c^u = \{j_{c,\lvert J_c\rvert+1}, \ldots, j_{c,\lvert J_c\rvert+\lvert J_c^u\rvert}\}$

these containers define restacking jobs that have to be processed in the sequence which is defined by the stacking order. Again, for the sake of notational convenience, we will usually write $j \in R$ to refer to a restacking job, i.e., we will omit its corresponding loading or unloading job, whenever possible. Restacking jobs that are in the process of being executed by some SC at the beginning of the planning horizon are assumed to not be included in R.

All relevant locations in the port are identified by points with integer coordinates in a Cartesian coordinate system in the plane. The distance between two points x and y in this coordinate system is calculated by using the Manhattan metric and is denoted by $d(x, y)$. Each job $j \in J \cup R$ is associated with two points, an origin and a destination. We denote these points by $ori(j)$ and $dest(j)$, respectively. For each loading job, the origin is a point that is located in one of the storage areas, where it identifies some stack of containers, while the destination corresponds to a point that represents a buffer area of a quay crane. Similarly, unloading jobs are associated to an origin that represents a buffer and a destination in the storage areas. For restacking jobs, both the origin and destination are points in the storage areas.

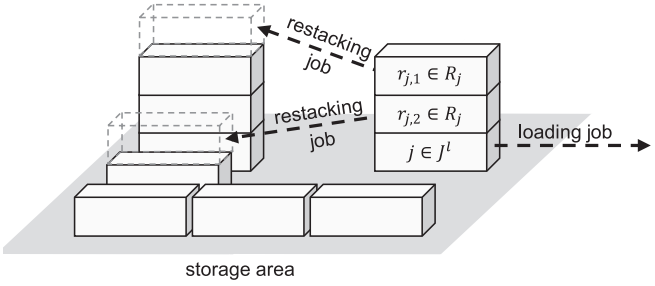

Fig. 4. Restacking jobs associated to a loading job.

The set of SCs is denoted by K, $\lvert K\rvert = n^v$. Each $k \in K$ is associated with a time instant $a_k^v \in \mathbb{N}_{\geq 0}$, at which it is available for processing the next container. In case of a SC that is handling a container at the beginning of the planning horizon, this time instant corresponds to the point in time at which it has finished processing this container. For all remaining SCs, this time instant is set to zero. As SCs may be located anywhere in the port at their availability times, each $k \in K$ is additionally associated with a starting point s_k, which corresponds to its position at time instant a_k^v. We define $S := \{s_1, \ldots, s_{n^v}\}$. Again, we will sometimes write $j \in S$ to refer to a starting position, i.e., we will not explicitly name the corresponding SC. Furthermore, we denote the SC that handles a container $j \in \bigcup_{c \in C^l} J_c^v$ at the beginning of the planning horizon by $SC(j) \in K$. We assume that the SCs move at a constant speed, neglecting acceleration phases, and define $p^v \in \mathbb{N}_{>0}$ to be the time needed by a SC to move one distance unit. Furthermore, we define $t^v \in \mathbb{N}_{>0}$ to be the time needed by a SC in order to lift or drop a container. The total time needed to process job $j \in J \cup R$ by a SC is denoted by t_j. This time includes the lifting and dropping operations, as well as the time needed to move from the origin of job j to the destination of job j. Hence, $t_j = p^v \cdot d(ori(j), dest(j)) + 2t^v$ for all $j \in J \cup R$. Note that these processing times are integer because of using the Manhattan metric on integer coordinates to determine the relevant distances.

Table 2 summarizes the notation regarding the SCs and the jobs.

Table 2
Notation regarding the SCs and the jobs.

K	Set of SCs	$\lvert K\rvert = n^v$
t^v	Time needed to lift or drop a container by a SC	$t^v \in \mathbb{N}_{>0}$
p^v	Time needed by a SC to move one distance unit	$p^v \in \mathbb{N}_{>0}$
a_k^v	Availability time of SC $k \in K$	$a_k^v \in \mathbb{N}_{\geq 0}$
s_k	Starting point of SC $k \in K$	$S := \{s_1, \ldots, s_{n^v}\}$
$SC(j)$	SC that handles a container $j \in \bigcup_{c \in C^l} J_c^v$ at the beginning of the planning horizon	$SC(j) \in K$
J^l	Set of loading jobs	$J^l := \bigcup_{c \in C^l} \left(J_c \setminus J_c^l\right)$
J^u	Set of unloading jobs	$J^u := \bigcup_{c \in C^u} \left(J_c \cup J_c^u\right)$
J	Set of loading and unloading jobs	$J := J^l \cup J^u$
R_j	Sequence of restacking jobs associated to job $j \in J$	$R_j = \{r_{j,1}, \ldots, r_{j,\lvert R_j\rvert}\}$
R	Set of restacking jobs	$R := \bigcup_{j \in J} R_j$
$ori(j)$	Origin of job $j \in J \cup R$	Integer coordinate
$dest(j)$	Destination of job $j \in J \cup R$	Integer coordinate
t_j	Time needed to process job $j \in J \cup R$ by a SC	$t_j = p^v \cdot d(ori(j), dest(j)) + 2t^v$

Table 3
Additional notation.

A	Feasible movements of unloaded SCs	
t_{ij}	Travel time between destination of job $i \in J \cup R$ (or starting position $i \in S$) and the origin of job $j \in J \cup R$	$t_{ij} = p^v \cdot d(dest(i), ori(j))$
t^s	Safety time	$t^s \in \mathbb{N}_{>0}$
t^b	Time that a container must remain unprocessed after it has been dropped in a buffer	$t^b \in \mathbb{N}_{>0}$
a_j^o	Availability time of container $j \in \bigcup_{c \in C} J_c^b$	$a_j^o \in \mathbb{N}_{\geq 0}$
n_c^b	Number of empty slots in the buffer of quay crane $c \in C$ at the beginning of the planning horizon	
A_c^b	Set of indices of empty slots of quay crane $c \in C$ at the beginning of the planning horizon	
a_{ic}^b	Availability time of the ith $(i = 1, \ldots, n_c^b)$ empty slot of the buffer of quay crane $c \in C$	$a_{ic}^b \in \mathbb{N}_{\geq 0}$
a_j^r	Earliest possible completion time of a lifting or dropping operation of a job $j \in J \cup \{r_{j,1} \mid j \in J, R_j \neq \emptyset\}$	$a_j^r \in \mathbb{N}_{\geq 0}$

The movement of unloaded SCs is subject of the optimization and is modelled by an edge-weighted, directed graph $G = (V, A)$, with vertex set V and edge set A. Each $j \in J \cup R \cup S$ defines a distinct vertex of the set V. Feasible movements of unloaded SCs are represented by the edge set $A := \{(i, j) \mid i \in J \cup R \cup S,\ j \in J \cup R,\ i \neq j\}$. This definition of the edge set does not allow one-way travel settings, where SCs can only move to a subset of buffer areas after having dropped a container. However, by appropriately redefining the edge set of the graph (and potentially including dummy vertices), these settings can be incorporated in a straightforward manner. The weight t_{ij} of an edge $(i, j) \in A$ represents the (integer) time needed by a SC to move from the destination of job i (or a starting position i, if $i \in S$) to the origin of job j, i.e., $t_{ij} = p^v \cdot d(dest(i), ori(j))$ if $i \in J \cup R$ or $t_{ij} = p^v \cdot d(i, ori(j))$ if $i \in S$. A solution of the routing component of MSCRB is therefore represented by n^v paths in G, with the path of SC $k \in K$ starting in $s_k \in S$.

Collisions are avoided by making use of safety times. The difference of the time instants at which the lifting process (performed by a quay crane or a SC) of a container in a given slot of a buffer is started and the time instant at which the dropping process of the next container in this slot is finished must be at least $t^s \in \mathbb{N}_{>0}$. The same time period must elapse between the beginning and the end of two succeeding lifting operations of the restacking jobs of a restacking set R_j, $j \in J$, as well as between the beginning of the lifting operation of $r_{j,\lvert R_j\rvert}$, $j \in J$, and the end of the lifting (in case of a loading job) or dropping (in case of an unloading job) operation of the corresponding job j. Similarly, a container that has been dropped in a buffer must remain unprocessed for at least $t^b \in \mathbb{N}_{>0}$ time units before it can be started to be picked up by a quay crane or a SC.

In order to take account of the setting at the beginning of the planning horizon, we make use of some additional notation. The lifting process (by a quay crane or a SC) of a container $j \in \bigcup_{c \in C} J_c^b$ may not start before time instant $a_j^o \in \mathbb{N}_{\geq 0}$. Similarly, there may be a restriction on the earliest possible completion time of a lifting or dropping operation of a job $j \in J \cup \{r_{j,1} \mid j \in J,\ R_j \neq \emptyset\}$ within the storage areas because a corresponding (preceding) restacking operation has been started just before the beginning of the planning horizon, so that it is not included in R while the safety time has not yet elapsed. Therefore, each job of this set is associated with a time instant $a_j^r \in \mathbb{N}_{\geq 0}$ that corresponds to the earliest possible completion time of a lifting or dropping operation of the corresponding container. Finally, an empty slot of a buffer may not be immediately available for dropping a container because of the above safety restrictions. A slot of a buffer is considered to be empty at the beginning of the planning horizon, if there is no container located in the slot and if there is no SC that is processing a container or an unloading crane that has started an unloading process of a container that will be dropped in the slot. Note that a container that is in the process of being lifted at the beginning of the planning horizon is not considered to be located in any slot. The number of empty slots in the buffer of quay crane $c \in C$ at the beginning of the planning horizon is denoted by n_c^b. We define $A_c^b := \{1, \ldots, n_c^b\}$ for $c \in C$. The availability time of the i-th empty slot, $i \in A_c^b$, in the buffer of quay crane $c \in C$ is represented by the parameter $a_{ic}^b \in \mathbb{N}_{\geq 0}$.

The additional notation used throughout this paper is summarized in Table 3.

2.1. An integer programming formulation of MSCRB

We define two non-negative time variables, w_j^{in} and w_j^{out}, for each $j \in \bigcup_{c \in C^l} J_c^l \cup J$, i.e., for all containers that interact with one of the buffer areas within the planning horizon. w_j^{in} represents the time instant at which the dropping process (by a SC or a quay crane) of j in its buffer is finished; w_j^{out} is the time instant at which the lifting process is started. Similarly, for each restacking job $j \in R$, we define a non-negative variable w_j that represents the time instant at which the lifting process of the corresponding container is started. We may restrict ourselves to considering all of these time variables to be integer, because all processing times, travel times, and time parameters are integral multiples of a time unit.

In order to avoid lengthy case differentiations, we define auxiliary variables for the quay cranes,

$$\hat{w}_j^q := \begin{cases} w_j^{in} & \text{if } c \in C^u, \\ w_j^{out} & \text{if } c \in C^l, \end{cases} \quad \forall c \in C,\ j \in J_c, \tag{1}$$

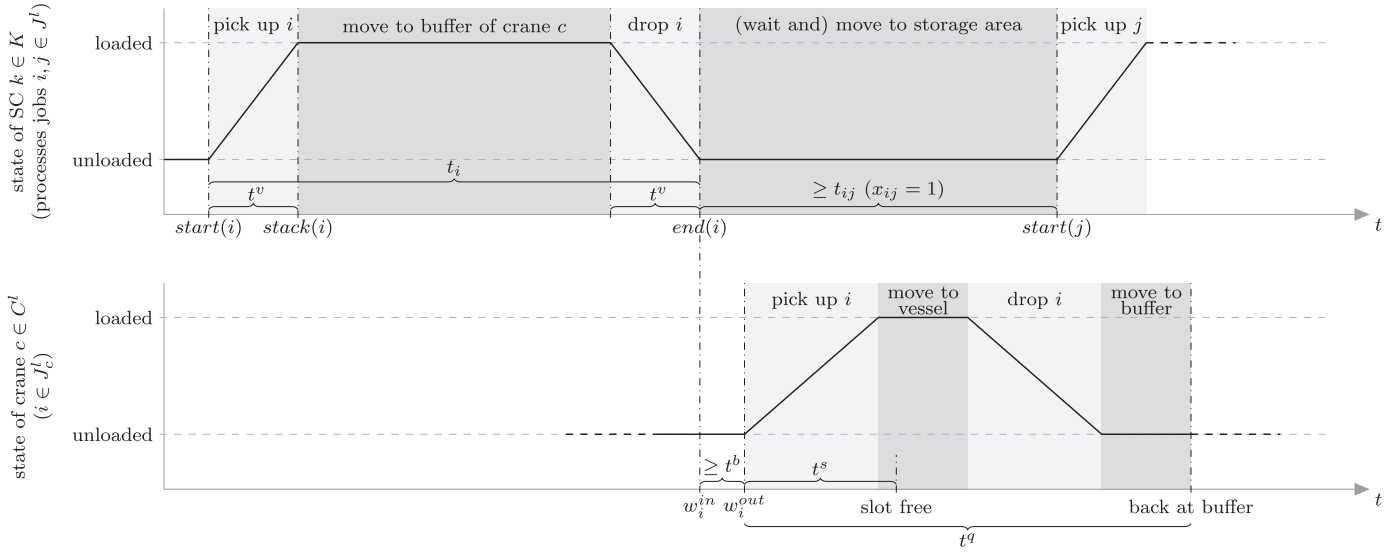

Fig. 5. Illustration of the variables associated to loading jobs.

Table 4
Definition of *start*(*j*), *end*(*j*), and *stack*(*j*).

	start(*j*)	*end*(*j*)	*stack*(*j*)
$j \in J^l$	$w_j^{in} - t_j$	w_j^{in}	$w_j^{in} - t_j + t^v$
$j \in J^u$	w_j^{out}	$w_j^{out} + t_j$	$w_j^{out} + t_j$
$j \in R$	w_j	$w_j + t_j$	$w_j + t^v$
$j \in S$	–	a_k^v	–

and the SCs,

$$\hat{w}_j^v := \begin{cases} w_j^{in} & \text{if } j \in J^l, \\ w_j^{out} & \text{if } j \in J^u, \end{cases} \quad \forall\, j \in J. \tag{2}$$

We define $\hat{w}^q_{j_{c,0}} := -t^q + a_c^q$ for all $c \in C^l$ and $\hat{w}^q_{j_{c,0}} := a_c^q$ for all $c \in C^u$. Moreover, auxiliary variables *start*(*j*) and *end*(*j*), $j \in J \cup R$, represent the time instants at which a SC starts the lifting process or completes the dropping process of a container, respectively. For $j \in S$, j is related to a distinct starting point of a SC $k \in K$, and we set $end(j) := a_k^v$. Analogously, *stack*(*j*), $j \in J \cup R$, represents the time instant at which a SC finishes a lifting or dropping process within one of the stacking areas when executing job j. Table 4 illustrates the values of these auxiliary variables.

The movement of unloaded SCs is modelled by binary variables

$$x_{ij} := \begin{cases} 1 & \text{if the same SC executes } j \text{ immediately} \\ & \text{after executing/starting in } i, \\ 0 & \text{otherwise,} \end{cases} \quad \forall\, (i,j) \in A. \tag{3}$$

Fig. 5 illustrates the variables that have been defined above for the case of loading jobs. It presents the state (loaded, unloaded, picking up, dropping) of a SC $k \in K$ (timeline on the top of the figure) and a loading crane $c \in C^l$ (timeline on the bottom of the figure) over the relevant part of the time horizon. k processes job $j \in J^l$ immediately after $i \in J_c^l$, so that $x_{ij} = 1$. It starts lifting i at time *start*(i) and finishes dropping i at time *end*(i). Note that $end(i) - start(i) = t_i$, i.e., we do generally not allow for a SC to "wait" while being loaded. However, our model allows waiting periods for unloaded movements of a SC. In Fig. 5, this is indicated by the fact that $start(j) - end(i) \geq t_{ij}$. In this context, note that our model does not answer the question of *where* to wait as it is based on the assumption that SCs can always pass each other. Hence, we abstract from congestion situations, especially in front of the buffer areas.

In order to be able to model the buffer capacities, we make use of binary variables $b_{ij}^{in} \in \{0,1\}$ and $b_{ij}^{out} \in \{0,1\}$ for all $c \in C$, $i, j \in J_c \cup J_c^u$, $i \neq j$. The variables b_{ij}^{out} are also defined for all $c \in C$, $i \in A_c^b$, $j \in J_c \cup J_c^u$. We will refer to these variables as the b-variables. In this context, recall that $J_c^u = \emptyset$ for all $c \in C^l$. For a given crane $c \in C$ and its corresponding buffer, b_{ij}^{in} takes the value one if $w_i^{in} < w_j^{in}$, i.e., if container i is dropped in the buffer before container j is dropped in the buffer. If $w_i^{in} = w_j^{in}$, exactly one of the variables b_{ij}^{in} and b_{ji}^{in} is set to one. Similarly, b_{ij}^{out} is set to zero if $w_i^{out} + t^s > w_j^{in}$ (or $a_{ic}^b > w_j^{in}$ if $i \in A_c^b$), i.e., if container j is dropped in the buffer before the slot that container i has been dropped in (or before a slot that is empty at the beginning of the planning horizon) is available.

Based on these definitions and a large positive integer M, an integer programming formulation of MSCRB is as follows:

$$\min \quad \sum_{j \in \bigcup_{c \in C} J_c} \hat{w}_j^q \tag{4}$$

$$s.t. \quad \sum_{(i,j) \in A} x_{ij} = 1 \qquad \forall\, j \in J \cup R, \tag{5}$$

$$\sum_{(j,i) \in A} x_{ji} \leq 1 \qquad \forall\, j \in J \cup R \cup S, \tag{6}$$

$$end(i) + t_{ij} - start(j) \leq (1 - x_{ij})M \qquad \forall\, (i,j) \in A, \tag{7}$$

$$\hat{w}^q_{j_{c,i-1}} + t^q - \hat{w}^q_{j_{c,i}} \leq 0 \qquad \forall\, c \in C,\ j_{c,i} \in J_c, \tag{8}$$

$$w_j^{in} + t^b \leq w_j^{out} \qquad \forall\, j \in J \setminus \bigcup_{c \in C^u} J_c^b \qquad \cup \bigcup_{c \in C^l} J_c^v, \tag{9}$$

$$a_j^o \leq w_j^{out} \qquad \forall\, j \in \bigcup_{c \in C} J_c^b, \tag{10}$$

$$w_j^{in} = 0 \qquad \forall\, j \in \bigcup_{c \in C} J_c^b, \tag{11}$$

$$w_j^{in} = a^v_{SC(j)} \qquad \forall\, j \in \bigcup_{c \in C^l} J_c^v, \tag{12}$$

$$w_j^{in} = a_c^q \qquad \forall c \in C^u, j \in J_c^q, \quad (13)$$

$$start(r_{j,|R_j|}) + t^s \leq stack(j) \qquad \forall j \in J \text{ with } R_j \neq \emptyset, \quad (14)$$

$$start(r_{j,i-1}) + t^s \leq stack(r_{j,i}) \qquad \forall j \in J, r_{j,i} \in R_j, i \geq 2, \quad (15)$$

$$a_j^r \leq stack(j) \qquad \forall j \in J \cup \{r_{j,1} | j \in J, R_j \neq \emptyset\}, \quad (16)$$

$$\sum_{i \in J_c \cup J_c^u, i \neq j} b_{ij}^{in} - \sum_{i \in J_c \cup J_c^u \cup A_c^b} b_{ij}^{out} \leq b_c - 1 \quad \forall c \in C, j \in J_c \cup J_c^u, \quad (17)$$

$$w_{j_{c,l}}^{in} - w_{j_{c,i}}^{in} + 0.5 \leq M\, b_{j_{c,i}j_{c,l}}^{in} \quad \forall c \in C, i = 1, \ldots, |J_c| + |J_c^u| - 1, \; l = i+1, \ldots, |J_c| + |J_c^u|, \quad (18)$$

$$w_{j_{c,l}}^{in} - w_{j_{c,i}}^{in} \leq M\, b_{j_{c,i}j_{c,l}}^{in} \quad \forall c \in C, i = 2, \ldots, |J_c| + |J_c^u|, \; l = 1, \ldots, i-1, \quad (19)$$

$$w_j^{in} - (w_i^{out} + t^s) \geq M(b_{ij}^{out} - 1) \quad \forall c \in C, i, j \in J_c \cup J_c^u, i \neq j, \quad (20)$$

$$w_j^{in} - a_{ic}^b \geq M(b_{ij}^{out} - 1) \qquad \forall c \in C, i \in A_c^b, j \in J_c \cup J_c^u, \quad (21)$$

$$w_j^{in}, w_j^{out} \in \mathbb{N}_{\geq 0} \qquad \forall j \in \bigcup_{c \in C^l} J_c^l \cup J, \quad (22)$$

$$w_j \in \mathbb{N}_{\geq 0} \qquad \forall j \in R, \quad (23)$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in A, \quad (24)$$

$$b_{ij}^{in} \in \{0, 1\} \qquad \forall c \in C, i, j \in J_c \cup J_c^u, i \neq j, \quad (25)$$

$$b_{ij}^{out} \in \{0, 1\} \qquad \forall c \in C, i, j \in J_c \cup J_c^u, i \neq j \text{ or } i \in A_c^b, j \in J_c \cup J_c^u. \quad (26)$$

The objective function (4) minimizes the sum of the time instants at which the quay cranes interact with their buffer areas. Constraints (5) and (6) guarantee that each job is processed by exactly one SC and take account of the fact that the number of SCs and their starting positions are given. Restrictions (7) ensure that at least t_{ij} time units elapse for the unloaded movement when a SC processes job $j \in J \cup R$ immediately after job (or after having started in) $i \in J \cup R \cup S$. They also take account of the availability times of the SCs. Constraints (8) enforce the quay cranes to respect the given loading and unloading sequences of containers and guarantee that t^q time periods elapse between processing two containers. Furthermore, it takes account of the availability times of the cranes. The fact that a container must remain unprocessed for t^b time units after having been dropped in a buffer is modelled by restrictions (9) and (10). Constraints (11)–(13) fix time variables of containers that are located in a buffer or that are being processed by a SC or a quay crane at the beginning of the planning horizon. The precedence relations among restacking jobs and their associated loading or unloading jobs as well as corresponding safety time restrictions are enforced by constraints (14)–(16). The limited capacities of the buffer areas are modelled by constraints (17)–(21).

For the buffer of a crane $c \in C$ and a container $j \in J_c \cup J_c^u$, the first summand on the left hand side of the corresponding constraint (17) counts the number of dropping operations in this buffer that have been completed before the dropping of j has been completed, while the second summand determines the number of lifting operations that have been started and resulted in an available slot before the dropping of j has been completed. The constraint then enforces the difference of these values to be no larger than the buffer capacity reduced by one slot for container j. The summands of the left hand side of constraints (17) make use of the b-variables as defined above. These variables are linked to their corresponding time variables by constraints (18)–(21). Note that, for a given set of containers that have been dropped in a buffer at the same time, constraints (18) make use of the fact that all time variables are integral multiples of a time unit to generate an artificial sequence that is needed for constraints (17). Furthermore, note that constraints (20) and (21) take account of safety time considerations. As we allow the SCs to process the containers associated to a given buffer area in an arbitrary order as long as the buffer capacity is not exceeded and the container sequence of the crane is met, we cannot convert restrictions (17)–(21) into time window constraints in a straightforward manner as, for example, done in Nguyen and Kim (2009) and Vis, de Koster, and Savelsbergh (2005). Finally, the domains of the variables are defined by constraints (22)–(26).

2.2. Computational complexity

In this section, we will prove that the decision version of MSCRB, referred to as D-MSCRB, is NP-complete in the strong sense by reduction of 3-Partition. D-MSCRB is defined in line with its optimization version and asks whether there exists a feasible solution with objective function value of no more than a given L.

An instance of 3-Partition, which is well known to be strongly NP-complete (Garey & Johnson, 1979), is defined by $3m + 1$ integers $u_1, \ldots, u_{3m}, B$ with $\sum_{j=1}^{3m} u_j = mB$ and $\frac{B}{4} < u_j < \frac{B}{2}$ for all $j \in \{1, \ldots, 3m\}$. It asks if there exists a partition of the set $\{1, \ldots, 3m\}$ into m subsets $U_1, \ldots, U_m$, such that $\sum_{j \in U_i} u_j = B$ for all $i \in \{1, \ldots, m\}$. Note that for every yes-instance of 3-Partition, we have $|U_i| = 3$ for all $i \in \{1, \ldots, m\}$. Therefore, 3-Partition is still strongly NP-hard if all integers are assumed to be multiples of 4 (if this is not the case, they can simply be multiplied by 4).

Theorem 1. *D-MSCRB is NP-complete in the strong sense.*

Proof. It can easily be seen that D-MSCRB is in NP.

Now, assume that we are given an instance I_P of 3-Partition with all integers $u_1, \ldots, u_{3m}, B$ being multiples of 4. By definition of 3-Partition, we have

$$\frac{B}{4} + 1 \leq u_j \leq \frac{B}{2} - 2 \qquad \forall j \in \{1, \ldots, 3m\}. \quad (27)$$

Based on I_P, we construct an instance I_S of D-MSCRB in polynomial time as described in the following and as illustrated in Fig. 6 for even m. We define $u'_j := u_j \cdot 11m$ for all $j \in \{1, \ldots, 3m\}$, $B' := B \cdot 11m$, $u'_{min} := min\{u'_1, \ldots, u'_{3m}\}$, and set $K = \{1, 2, \ldots, m\}$, so that there are $n^v = m$ homogenous SCs. All SCs are immediately available and located in the same starting point $\alpha := (0, \frac{u'_{min}}{2} - 3m - 2)$ at the beginning of the planning horizon, i.e., $s_k = \alpha$ for $k = 1, \ldots, n^v$. Furthermore, we set $t^q = B' - \frac{u'_{min}}{2} + 3m - 1$ and $p^v = t^b = t^s = t^v = 1$.

There are $3m$ loading cranes $C^l = \{c_1^l, \ldots, c_{3m}^l\}$ and $3m$ unloading cranes $C^u = \{c_1^u, \ldots, c_{3m}^u\}$ that are equally distributed on the line $(-3m + 1, 0) - (3m, 0)$ and that are placed as described in the following. For the sake of notational convenience, we refer to the points that correspond to the locations of the cranes by their cranes' identifiers. Crane c_1^l defines the rightmost crane at $(3m, 0)$.

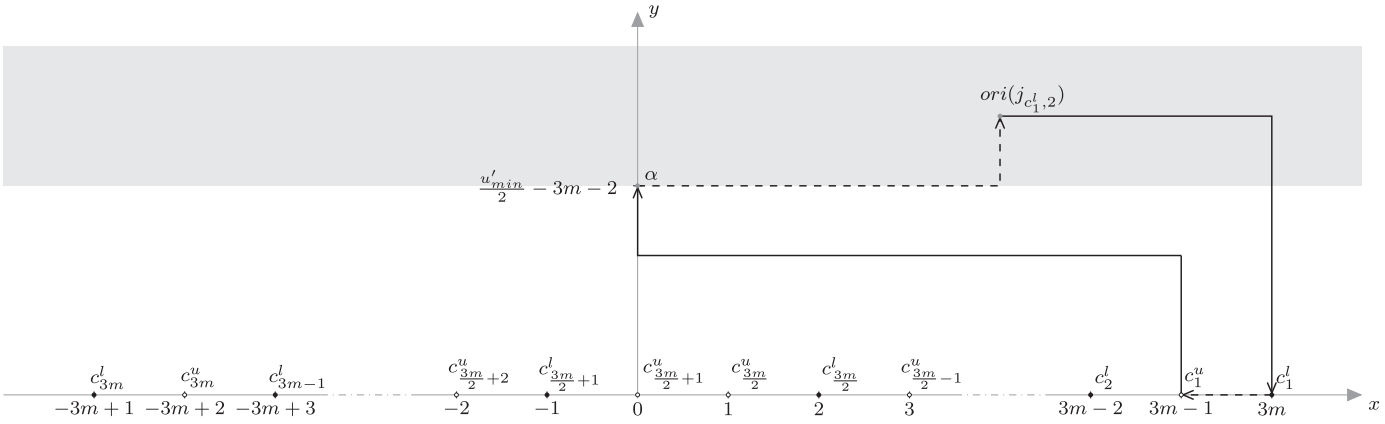

Fig. 6. Illustration of I_S for even m.

The remaining cranes are placed to the left of this crane in an alternating manner such that $d(c_i^l, c_i^u) = 1$ for all $i \in \{1, \dots, 3m\}$. Finally, the cranes c_i^l and c_i^u are interchanged if $d(c_i^l, \alpha) < d(c_i^u, \alpha)$ for all $i \in \{1, \dots, 3m\}$. Hence, the resulting placement of the cranes is such that $d(c_i^l, \alpha) > d(c_i^u, \alpha)$ for all $i \in \{1, \dots, 3m\}$ and $d(c_i^j, \alpha) \leq \frac{u'_{min}}{2} - 2$ for all $j \in \{u, l\}$ and $i \in \{1, \dots, 3m\}$.

Each loading crane $c_i^l \in C^l$ is associated to exactly two containers, $j_{c_i^l,1}$ and $j_{c_i^l,2}$, the first one of which is located in the crane's buffer at the beginning of the planning horizon, so that it does not define a job. Each unloading crane $c_i^u \in C^u$ is associated to exactly two containers, $j_{c_i^u,1}$ and $j_{c_i^u,2}$, both of which have to be processed by SCs. Note, however, that the processing of $j_{c_i^u,2}$ by some SC does not have an effect on the answer of I_S for all $c_i^u \in C^u$. We will therefore restrict ourselves to solely considering the jobs related to $j_{c_i^u,1}$ for all $c_i^u \in C^u$ in the remainder of this proof. Hence, each crane is associated to exactly one job that is relevant for the answer of I_S. We set $dest(j_{c_i^u,1}) = \alpha$ for all $c_i^u \in C^u$. Furthermore, for all $i \in \{1, \dots, 3m\}$, we set $ori(j_{c_i^l,2})$ to arbitrary values, such that

$$d(\alpha, ori(j_{c_i^l,2})) + d(ori(j_{c_i^l,2}), c_i^l) + 2t^v + d(c_i^l, c_i^u) + d(c_i^u, \alpha) + 2t^v = d(\alpha, ori(j_{c_i^l,2})) + t_{j_{c_i^l,2}} + d(c_i^l, c_i^u) + t_{j_{c_i^u,1}} = u'_i \quad (28)$$

(see Fig. 6) and such that the y-coordinate of $ori(j_{c_i^l,2})$ is greater or equal to $\frac{u'_{min}}{2} - 3m - 2$ (shaded area in Fig. 6). Hence, for all $i \in \{1, \dots, 3m\}$, there is a one to one correspondence of the integer u_i of I_P and the pair $\{c_i^l, c_i^u\}$ of cranes of I_S.

All remaining parameters of I_S are set to arbitrary values, such that

- each crane is necessarily idle for at least one time unit before the processing of its last associated container if a SC starts the lifting process or finishes the dropping process of the associated job after time instant $B' - \frac{u'_{min}}{2} + 3m$, and
- each crane can process all of its associated containers without idle time if a SC starts the lifting process or finishes the dropping process of the associated job no later than by time instant $B' - \frac{u'_{min}}{2} + 3m$, and
- $j_{c_i^u,1}$ can immediately started to be lifted (t^b has elapsed) upon the earliest possible arrival of any SC at the corresponding buffer for all $c_i^u \in C^u$, and
- there exists at least one slot in the buffer of c_i^l, where a container can be dropped immediately upon the earliest possible arrival of any SC for all $c_i^l \in C^l$.

Finally, we set L to the objective function value of the optimization version of D-MSCRB that is defined by all quay cranes processing their associated containers without any idle times before processing their last container.

In the remainder of this proof, we will show that I_P is a yes-instance if and only if I_S is a yes-instance, i.e., that we have constructed a pseudo-polynomial transformation (Garey & Johnson, 1979) from 3-Partition to D-MSCRB, which proves the latter problem to be strongly NP-complete.

First, assume that I_P is a yes-instance and let $U_i = \{u_{i_1}, u_{i_2}, u_{i_3}\}$ denote the i-th subset of a corresponding partition for $i = 1, \dots, m$. Now, for all $i \in \{1, \dots, m\}$, assign the following sequence of jobs to SC i: $j_{c_{i_1}^l,2}, j_{c_{i_1}^u,1}, j_{c_{i_2}^l,2}, j_{c_{i_2}^u,1}, j_{c_{i_3}^l,2}, j_{c_{i_3}^u,1}$. Let each SC process the jobs of its sequence as fast as possible. As $d(\alpha, ori(j_{c_i^l,2})) + t_{j_{c_i^l,2}} + d(c_i^l, c_i^u) + t_{j_{c_i^u,1}} = u'_i$ for all $i \in \{1, \dots, 3m\}$, each SC finishes processing its sequence at time instant B'. Hence, SC i begins lifting $j_{c_{i_3}^u,1}$ at time instant $B' - 2 - (\frac{u'_{min}}{2} - 3m - 2) - d(c_{i_3}^u, (0,0)) \leq B' - \frac{u'_{min}}{2} + 3m$ for all $i \in \{1, \dots, 3m\}$, so that there exists a solution to I_S where none of the quay cranes are idle before processing their last associated containers.

Next, assume that I_S is a yes-instance and let Θ be a corresponding feasible solution, i.e., a solution where none of the quay cranes are idle before processing their last associated containers. We will make use of three auxiliary properties.

Property 1. *Each SC $k \in K$ processes exactly three unloading jobs in Θ.*

Proof of Property 1. Assume that there is a SC $k \in K$ that processes at least four unloading jobs in Θ. Then this SC has to move from point α, which corresponds to its starting point at the beginning of the planning horizon as well as the destination of all unloading jobs, or an origin of some loading job (with a y-coordinate which is not smaller than the one of point α) to the quay, i.e., the x-axis in Fig. 6, and back for at least four times. As $p^v = 1$, this takes at least $8 \cdot (\frac{u'_{min}}{2} - 3m - 2) = 4u'_{min} - 24m - 16$ time units. Because of (27), we have $4u'_{min} - 24m - 16 \geq 4(\frac{B'}{4} + 11m) - 24m - 16 = B' + 20m - 16$. Therefore, when additionally taking account of six time units needed to lift and drop three containers, the SC starts lifting the container of the fourth unloading job no earlier than at time instant $B' + 20m - 10 - (\frac{u'_{min}}{2} - 3m - 2) = B' - \frac{u'_{min}}{2} + 23m - 8 > B' - \frac{u'_{min}}{2} + 3m$, so that the corresponding unloading crane is idle for at least one time unit, which contradicts the

feasibility of Θ. We can therefore conclude that no SC processes more than three unloading jobs. Since a total of $3m$ unloading jobs must be processed and there are m SCs, each SC must processes exactly three unloading jobs. □

Property 2. *Each SC $k \in K$ processes exactly three loading jobs in Θ.*

Proof of Property 2. As the y-coordinate of the origin of each loading job is not smaller than the y-coordinate of point α, the argumentation is analogous to the proof of Property 1. □

Property 3. *Θ is such that each SC $k \in K$ processes its loading and unloading jobs (Properties 1 and 2) in an alternating manner, starting with a loading job.*

Proof of Property 3. If SC $k \in K$ were to process an unloading job first or if it were to process its jobs (Properties 1 and 2) in a non-alternating manner, it would necessarily have to make at least seven trips between some point with a y-coordinate not smaller than the y-coordinate of point α and the quay before starting to lift (in case of an unloading job) or drop (in case of a loading job) its last container in a buffer at the quay. When taking account of the resulting travel times and the time needed to lift and drop containers, the remaining argumentation is analogous to the proof of Property 1. □

Based on Properties 1–3 and the fact that I_S is a yes-instance we make two observations. First, Θ is such that each SC $k \in K$ starts its last lifting operation of an unloading job no later than $B' - \frac{u'_{min}}{2} + 3m$, so that it finishes processing its last job no later than $B' + 3m$. Second, Θ is such that the sequence of jobs to be processed by each SC $k \in K$ is composed of three subsequences, each of which is composed of a loading and an unloading job and relates to a trip where the SC starts in α, processes a loading job, and returns to α while processing an unloading job. Denote the unique trip that contains loading job $j_{c_i^l,2}$, $i \in \{1, \dots, 3m\}$, by τ_i and the sum of all travel and processing times of this trip by $t(\tau_i)$. Because of (28) and the fact that $d(c_i^l, \alpha) > d(c_i^u, \alpha)$ for all $i \in \{1, \dots, 3m\}$, we have $t(\tau_i) \geq u'_i$ for all $i \in \{1, \dots, 3m\}$. Furthermore, denote the loading crane indices that correspond to the three loading jobs processed by SC $k \in K$ in Θ by $k_1, k_2, k_3 \in \{1, \dots, 3m\}$. Then, based on the above deliberations, we have $\sum_{i=1}^{3} u'_{k_i} \leq \sum_{i=1}^{3} t(\tau_{k_i}) \leq B' + 3m$ for all $k \in K$. When dividing this expression by 11 m, we get $\sum_{i=1}^{3} u_{k_i} \leq B + \frac{3}{11}$. Moreover, as all u_i, $i \in \{1, \dots, 3m\}$, are integer, we have $\sum_{i=1}^{3} u_{k_i} \leq B$. By definition of 3-Partition, we additionally know that $\sum_{i=1}^{3m} u_i = mB$, so that $\sum_{i=1}^{3} u_{k_i} = B$ for all $k \in K = \{1, \dots, m\}$ in Θ. Hence, we have constructed a solution to I_P, which concludes the proof. □

3. Heuristic framework

We now present a heuristic framework for MSCRB. It is illustrated in Fig. 7. Given some instance of MSCRB, it initiates with a *constructive procedure* that generates a first feasible solution. This procedure iteratively assigns jobs to SCs (in a greedy or random manner) such that idle times of quay cranes are avoided as far as possible. Details are presented in Section 3.1. The solution is then passed to an *improvement procedure* that is based on decomposing MSCRB into a graph-based *routing component* and a remaining integer part for handling the *time variables and buffer capacities*. Details are presented in Section 3.2. Due to the close relationship of the routing component of MSCRB and the asymmetric TSP, we propose to apply local search approaches for the routing component that have been designed and proven to perform well for the TSP, namely an ejection chain heuristic (Section 3.2.1) and a 3-Opt approach (Section 3.2.2). Each routing solution examined within the local search is evaluated by constructing a corresponding feasible solution of MSCRB (taking account of all time variables and buffer capacities) and computing its objective function value as described in Section 3.2.3. Note that we assume the input instance to have at least one feasible solution during the remainder of this section.

3.1. Constructive procedure for MSCRB

Our constructive procedure is illustrated in Fig. 8. As mentioned above, its main idea is to iteratively assign jobs to SCs such that idle times of quay cranes are avoided as far as possible.

During runtime of the algorithm, we keep track of the status of each SC, i.e., the latest job that has been assigned to the SC and the time at which the dropping operation of the corresponding container is completed by the SC. In the initialization step, each SC $k \in K$ is associated to its starting position s_k and availability time a_k^v. Similarly, each crane $c \in C$ is associated with a time stamp representing the time instant at which it will next be available for processing a container. It is initialized with a_c^q. Furthermore, we initialize the time variables w_j^{in}, with $j \in \bigcup_{c \in C} J_c^b$, $j \in \bigcup_{c \in C^l} J_c^v$, or $c \in C^u$ and $j \in J_c^q$, as in constraints (11)–(13). All remaining time variables are marked as unset. The variables x_{ij} are set to zero for all $(i, j) \in A$.

The algorithm then iterates over the cranes $c \in C$ and their corresponding sets J_c to potentially fix variables (1) to their earliest possible time instants and update the status of the cranes. Let j be the container that is currently considered while iterating over J_c. If c is a loading crane and w_j^{in} is set to some value, the algorithm fixes w_j^{out} in accordance with all additional time restrictions, e.g., safety time considerations or the time needed by a crane to process a container. The iteration over J_c stops, when the first container with w_j^{in} unset is reached. If, on the other hand, c is an unloading crane, the algorithm successively fixes variables w_j^{in} to their earliest possible time instants with respect to all additional time restrictions until all empty slots of the corresponding buffer have been taken into account.

The algorithm then enters its main loop, where it first generates (or later modifies) a list L of most urgent containers. For each

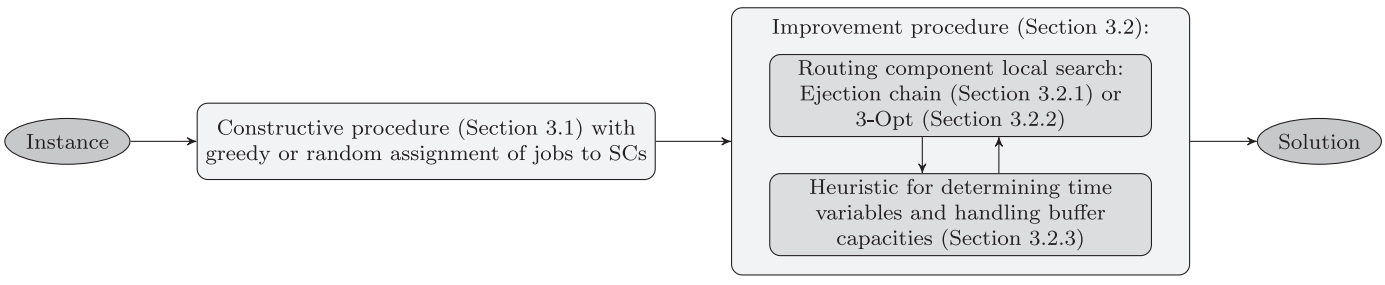

Fig. 7. Heuristic framework.

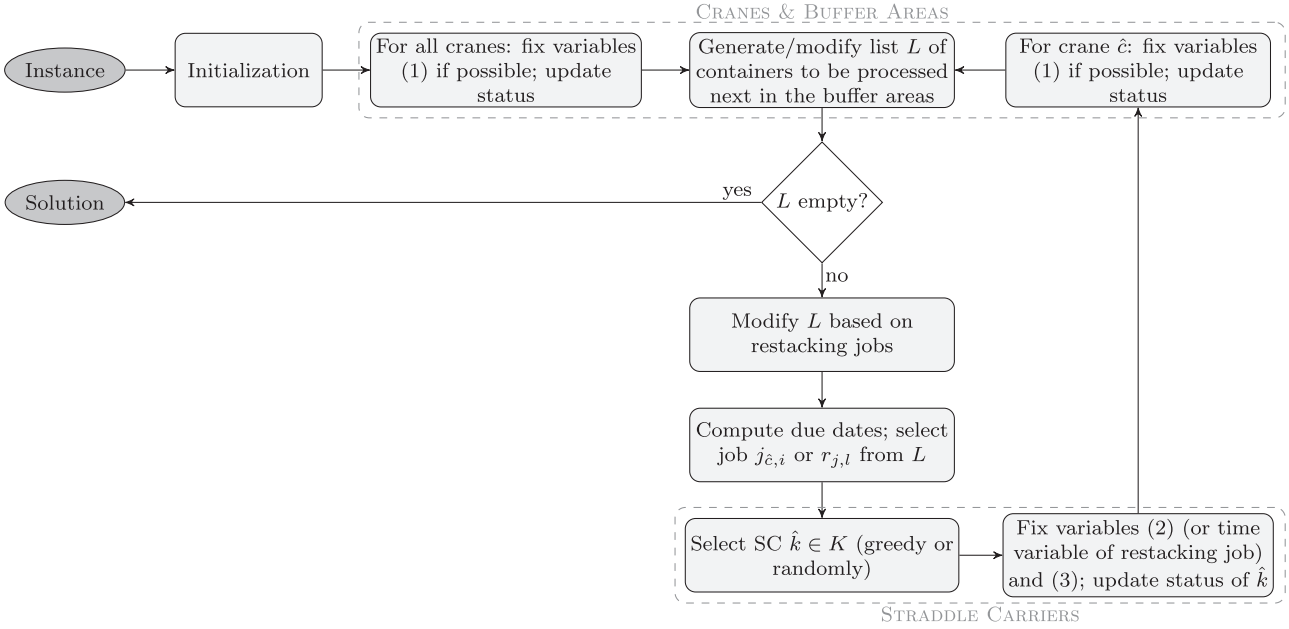

Fig. 8. Constructive Procedure.

buffer area, L includes at most one container. If the buffer area is associated to a loading crane $c \in C^l$, L includes the first container $j \in J_c$ (in case of this container's existence) with w_j^{in} unset that is detected when iterating over J_c as above. For an unloading crane $c \in C^u$, it includes a job $j \in J_c$ (if existing) with w_j^{in} minimal and w_j^{out} unset.

The algorithm terminates with a feasible solution once L is empty. If this is not the case, L is updated with respect to the restacking jobs. That is, if an element $j \in L$ has an associated non-empty set R_j, the first job l of this set with w_l unset (if existent) replaces j in L.

Next, the algorithm generates (or later updates) a due date d_j for each job $j \in L$. This idea is similar to the time window approach by Nguyen and Kim (2009) and Vis et al. (2005). A due date represents the latest time instant at which the processing of job j must start in order to prevent the associated quay crane from staying idle. Note that this due date may not be achievable or even be negative. In case of a restacking job, the due date takes account of all lifting operations needed to process the succeeding restacking jobs of the corresponding loading or unloading job. Again, the computation of due dates must take account of all additional time restrictions. The algorithm then selects a job with smallest due date from L. We refer to this job by $j_{\hat{c},i}$ in case of a loading or unloading job or by $r_{j,l}$ in case of a restacking job.

An example for the computation of due dates is as follows. Assume that a subset of the time variables associated to a loading crane $c \in C^l$ has been fixed, so that the crane's current time stamp is time instant t. Moreover, assume that $j_{c,i}$ is the first container of this crane's sequence J_c with an unset arrival time in the buffer, i.e., assume that $w_{j_{c,i}}^{in}$ is unset. Let $R_{j_{c,i}} = \{r_{j_{c,i},1}\}$ and assume that $w_{r_{j_{c,i},1}}$ is unset, so that $r_{j_{c,i},1}$ is included in L. That is, assume that $j_{c,i}$ has an associated restacking job, the processing of which has not yet been decided on. Obviously, c will not run idle if $j_{c,i}$ is dropped in its buffer at time instant $t - t^b$. To do so, a SC will first have to lift $r_{j_{c,i},1}$, before $j_{c,i}$ can finally be processed. Additionally, the safety time t^s must elapse between the beginning of the lifting operation of $r_{j_{c,i},1}$ and the end of the lifting operation of $j_{c,i}$. Hence, as $t_{j_{c,i}}$ includes the lifting and dropping operations needed to process $j_{c,i}$, we have $d_{j_{c,i}} = t - t^b - (t_{j_{c,i}} - t^v) - t^s$.

In the next step of our constructive procedure, a SC $\hat{k} \in K$ for processing job $j_{\hat{c},i}$ or $r_{j,l}$ is selected. There are two strategies of choosing $\hat{k}$ based on the status of all SCs. The *greedy* approach selects a SC that will be the first to arrive at the origin of the job. In the alternative strategy, a SC is *randomly* selected. Based on the selection of $\hat{k}$, it is then possible to fix variables (2) (or a time variable of a restacking job) and (3). The former variable is fixed to its earliest possible value with respect to all relevant time restrictions.

Based on the preceding computations for the selected SC $\hat{k}$, the algorithm then fixes variables (1) of crane $\hat{c}$ (if the selected job is a loading or an unloading job) with respect to all additional time restrictions as described above. If the selected job is a restacking job, no variables are fixed. Hereafter, L is modified in accordance with these computations and the algorithm proceeds as described above.

Note that, in the case of unloading cranes $c \in C^u$, there may remain jobs $j \in J_c$ with unset variables w_j^{out} and without an assigned SC upon termination of the algorithm. These variables, however, are not relevant for the objective function value and the corresponding jobs can, for example, be assigned to and sequentially processed by an arbitrary SC.

3.2. Improvement procedure

As mentioned above, our improvement procedure essentially corresponds to calling a local search procedure that is well established for the TSP on the routing component of a MSCRB solution. The evaluation of the routing solutions examined within the local search is performed by constructing a corresponding feasible solution of MSCRB that takes account of all time variables and capacity restrictions (Section 3.2.3).

We represent a solution of a TSP on some directed graph $\tilde{G} = (\tilde{V}, \tilde{A})$ by a directed graph with vertex set $\tilde{V}$, that includes an edge (i, j) iff vertex $j \in \tilde{V}$ follows vertex $i \in \tilde{V}$ in the considered solution. This latter graph is referred to as the *supporting graph* of the solution. As pointed out in Section 2, a feasible solution of the routing component of MSCRB is represented by n^v paths on $G = (V, A)$. We therefore have to augment the edge set A in order to be able to convert this solution into a tour, i.e., a feasible solution of a TSP.

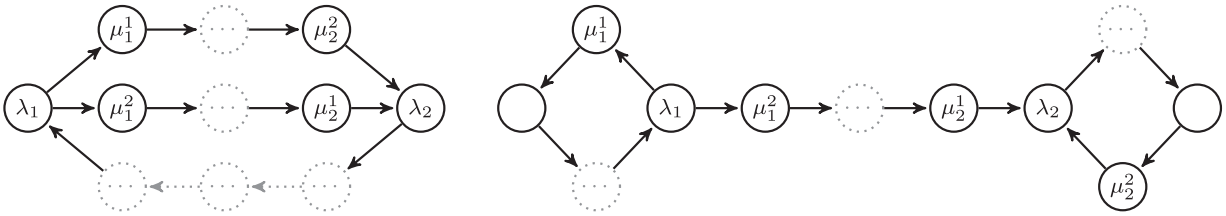

Fig. 9. Doubly rooted reference structure.

We thus define $A' := A \cup \{(i, j) \mid i \in V, j \in S \backslash \{i\}\}$ and consider the directed graph $G' := (V, A')$ in the remainder of this section. A given feasible solution of MSCRB is then transformed into the supporting graph of a tour by connecting the last vertex of the path of the k-th SC with the first vertex of the path of SC $k+1$ (with $n^v + 1 \equiv 1$) for all $k \in K$. The reverse transformation, i.e., the transformation of a tour into a feasible solution of MSCRB, is performed analogously.

We will make use of two local search approaches. The first algorithm is an *ejection chain approach* (Section 3.2.1). A detailed overview of these approaches is given by Rego and Glover (2010). Ejection chain methods have been proven to be a promising approach for a variety of combinatorial optimization problems, e.g., assignment problems (Rego, James, & Glover, 2010; Yagiura, Ibaraki, & Glover, 2004), the crew scheduling problem (Cavique, Rego, & Themido, 1999), or partitioning problems (e.g., Dorndorf, Jaehn, & Pesch, 2008; Kress, Boysen, & Pesch, 2017). Most important, they have very successfully been implemented for the TSP (examples include Glover, 1996; Pesch & Glover, 1997) and have been identified to "lead the state-of-the-art in local search heuristics for the traveling salesman problem" (Rego & Glover, 2010). As an alternative approach, we make use of a *3-Opt heuristic* (Section 3.2.2), which is another well known and successful technique for routing problems (see Korte & Vygen, 2012; Laporte, 1992).

3.2.1. Routing component: ejection chain approach

Ejection chain methods are local search approaches that "provide the ability to strategically extend simpler neighborhoods, such as those consisting of exchange (swap) moves or insert (shift) moves" (Rego et al., 2010), for example by making use of a *reference structure* that guides the generation of moves (Glover, 1996) by allowing a controlled portion of infeasibility in temporal solutions. A reference structure can be transformed into another reference structure or into feasible solutions by making use of well-defined rules.

Our ejection chain approach is embedded into a tabu search framework and makes use of the *doubly rooted reference structure*, which has been introduced by Glover (1996) in the context of the TSP and allows being used on a directed graph. This reference structure comes in two variants (see Fig. 9, where dotted vertices represent vertex chains of arbitrary length), which we represent by supporting graphs in analogy to the representation of solutions of the TSP, a *tricycle* (Fig. 9a) and a *bicycle* (Fig. 9b). Both variants represent infeasible solutions of a TSP, with at most two vertices of the supporting graph violating constraints of the TSP because of being incident to more than two edges. These vertices are referred to as *root vertices* and we will denote them by λ_1 (too many outgoing edges) and λ_2 (too many incoming edges). The root vertices may coincide. Each root vertex λ_i, $i \in \{1, 2\}$, is associated to two *subroots*, denoted by μ_i^1 and μ_i^2, being defined as the vertices that are incident to the two outgoing or incoming edges of the root vertex other than the root vertex itself, respectively. Subroots of a root vertex may coincide with the other root vertex. Furthermore, subroots of two root vertices may coincide. A tricycle connects the root vertices by three edge-disjoint paths, two of which include two subroots each. The remaining path does not include a subroot. Similarly, a bicycle is composed of two edge-disjoint cycles, each of which includes one root vertex and one of its subroots. If the root vertices are distinct, they are connected by a path that includes the remaining subroots.

Each iteration of our ejection chain method has two stages, as briefly illustrated in Algorithm 1. In the *probing stage*, the reference structures in the neighborhood of the incumbent reference structure are examined. They are constructed by iterating over all potential transformation steps on the incumbent reference structure as described below. Each neighbor is evaluated based on transforming its reference structure into feasible solutions of MSCRB (via tours) and computing their objective function values as described in Section 3.2.3. After having explored the complete neighborhood, the incumbent reference structure is updated to the one of a most promising neighbor (referred to as *bestNeighbor* in Algorithm 1) in the *transition stage*. Moreover, the overall best solution detected by the ejection chain method is potentially updated.

Algorithm 1: Ejection chain approach.

Input: Feasible solution *sol* of MSCRB instance, tabu length τ, parameter *stop*
Output: Feasible solution *sol*

1. *Initialization*: Initialize tabu list and generate an instance *refStruc* of the reference structure based on *sol*.
2. *Probing stage*: Generate all neighbors of *refStruc* by iterating over all potential transformation steps in accordance with the tabu list. Evaluate each neighbor by transformation into its corresponding MSCRB solution(s) and computation of the objective function value(s) as described in Section 3.2.3. Denote the most promising neighbor by *bestNeighbor*.
3. *Transition stage*: Set *refStruc* := *bestNeighbor*, update tabu list, and potentially update the best known solution *sol*.
4. *Stopping criterion*: If *sol* has not improved for *stop* succeeding calls of step 3, terminate the algorithm. Otherwise, go to step 2.

When generating the neighborhood of an incumbent reference structure in the probing stage, we have to perform transitions into other reference structures. Each of these transitions is based on selecting any subroot $\mu_i^j \in \{\mu_1^1, \mu_1^2, \mu_2^1, \mu_2^2\}$ and deleting the edge $e = (\lambda_i, \mu_i^j)$ if $i = 1$ or (μ_i^j, λ_i) if $i = 2$. Now, we add any edge $e' = (v, \mu_i^j) \in A'$ or $e' = (\mu_i^j, v) \in A'$, $e \neq e'$, that is not yet part of the supporting graph, such that the resulting supporting graph remains connected and such that each vertex has at least one incoming and one outgoing edge. This results in a reference structure, in which v is a root vertex.

Let us now turn our attention towards the evaluation of reference structures in the probing stage of our ejection chain method. There are two cases to consider. First, assume that the reference structure is a tricycle. In this case, we construct two tours. For each

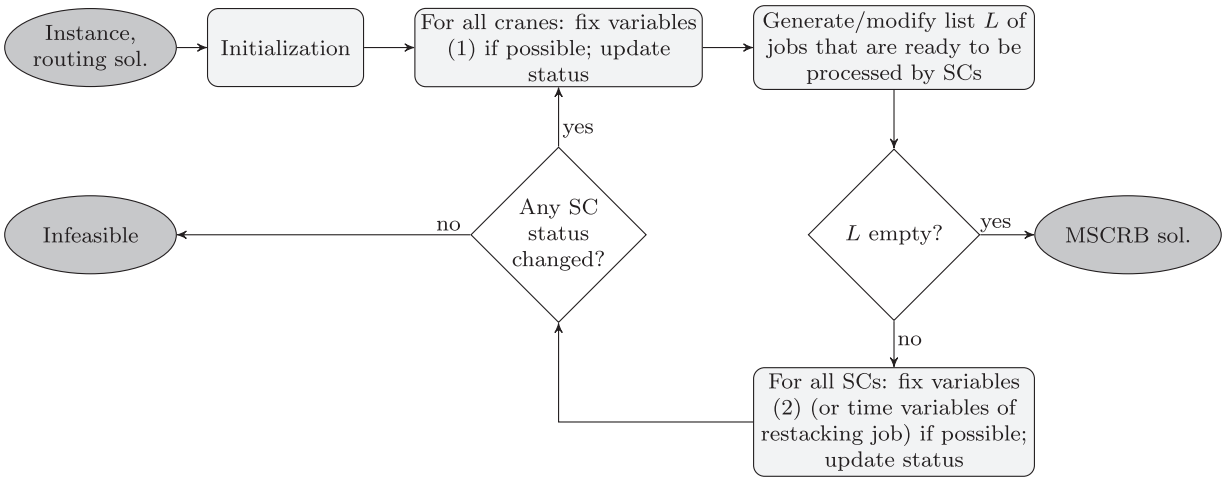

Fig. 10. Computing time variables.

Table 5
Example for the jobs associated to a crane $c \in C^l$ with $b_c = 3$ that are included in L.

l (container index)	1	$2 = i$	3	4	$5 = \lvert J_c \rvert$
$w^{in}_{j_{c,l}}$	Fixed		Fixed		
$w^{out}_{j_{c,l}}$	Fixed				
Included in L?	no ($1 < i$)	yes	no ($w_{j^{in}_{c,3}}$ fixed)	yes	no ($5 \geq i + b_c = 5$)

pair of subroots, $\mu^i_1 \in \{\mu^1_1, \mu^2_1\}$ and $\mu^j_2 \in \{\mu^1_2, \mu^2_2\}$, that are not part of the same path between the root vertices, we remove the edges (λ_1, μ^i_1) and (μ^j_2, λ_2) and add the edge (μ^j_2, μ^i_1) in the supporting graph. Second, in case of a bicycle, one tour is constructed if the root vertices are distinct. This is done by removing the edges (λ_1, μ^i_1) and (μ^j_2, λ_2) and adding the edge (μ^j_2, μ^i_1), where μ^i_1 and μ^j_2 are subroots that are part of the cycles. If the root vertices coincide, analogous transformations result in two tours.

The ejection chain method is initialized by generating an instance of the reference structure based on an initial feasible solution of MSCRB (see Algorithm 1). To do so, we first transform this solution into a tour as described above. This tour defines a supporting graph. We then randomly remove an edge (i, j) of this supporting graph and select two root vertices λ_1 and λ_2 that are not incident to (i, j). Finally, the edges (i, λ_1) and (λ_2, j) are added to the supporting graph. A tabu list is used to prevent the algorithm from cycling. Its length is limited to a predefined threshold value, referred to as the tabu length (referred to as τ in Algorithm 1). At the end of the transition stage, the edge e' which has been added to the supporting graph during the transformation of the reference structure (from *refStruc* to *bestNeighbor* in Algorithm 1) is inserted at the end of this list. Additionally, the first element of the list is removed, if the tabu length is exceeded. Elements of the tabu list may not be chosen as edges e to be deleted when generating neighbors of an incumbent reference structure. The algorithm terminates when the overall best solution has not improved for a given number of iterations (parameter *stop* in Algorithm 1). In order to intensify the search process, we restart the ejection chain algorithm for a given number of times. These restarts initiate with the MSCRB solution that was returned by the previous call of the algorithm and an empty tabu list.

3.2.2. Routing component: 3-Opt approach

The 3-Opt approach is well established in the literature. For the sake of brevity, we refrain from presenting it in detail and refer the reader to Korte and Vygen (2012) for details. In our context, it is a local search algorithm that starts with an initial feasible solution of MSCRB that has been transformed into a tour as described above. Roughly speaking, the neighborhood of a given tour is defined by all tours that can be constructed by deleting at most three edges in the supporting graph and afterwards replacing these edges with an identical amount of new edges. As this neighborhood can get very large for real-world instances of MSCRB, our implementation of 3-Opt skips an edge triple with a given probability when iterating over the triples. As before, each tour is evaluated by computing the objective function value of the corresponding solution of MSCRB as described in Section 3.2.3. The local search proceeds in a first-fit manner, until no additional improvement that exceeds a given threshold is achievable.

3.2.3. A fast heuristic for determining time variables and handling buffer capacities

Given a solution of the routing component of MSCRB, we have to compute feasible values of the time variables subject to the capacity constraints of the buffer areas in order to be able to determine the objective function value of an associated solution of MSCRB. To do so, we propose to make use of a heuristic approach that corresponds to a variant of the constructive procedure introduced in Section 3.1. It is illustrated in Fig. 10.

The initialization step and the process of fixing variables (1) is identical to the constructive procedure. The list L, however, is constructed in a different manner. It includes all loading and unloading jobs that are ready to be processed by SCs based on the fixed time variables that are associated to the buffer areas and the precedence constraints defined by their associated restacking sets. Let $j_{c,i}$, $c \in C$, be the first container of the set J_c with both $w^{in}_{j_{c,i}}$ and $w^{out}_{j_{c,i}}$ marked as unset (if this job exists). If $c \in C^l$, the jobs that are associated with this crane's buffer area and that are included in L are the ones of the set $\{j_{c,l} \in J_c | i \leq l < i + b_c \wedge w^{in}_{j_{c,l}}$ unset $\wedge$ $r_{j_{c,l},|R_{j_{c,l}}|}$ fixed if it exists$\}$. An example with $c \in C^l$, $b_c = 3$, and without the existence of restacking jobs is presented in Table 5.

It assumes that only few time variables have already been fixed (as indicated in the table), so that $i = 2$ and the jobs $j_{c,2}$ and $j_{c,4}$ are included in L. Similarly, for each $c \in C^u$, L includes all jobs of the set $\{j_{c,l} \in J_c | l < i \wedge w^{out}_{j_{c,l}}$ unset $\wedge\ r_{j_{c,l},|R_{j_{c,l}}|}$ fixed if it exists$\}$. Moreover, L includes all restacking jobs that can be processed in accordance with the relevant precedence constraints of their restacking sets. Let $r_{j,i}$, $j \in J$, be the first job of the set R_j with $w_{r_{j,i}}$ unset (if this job exists). Then this job is included in L.

The algorithm terminates with a feasible solution of MSCRB if L is empty, i.e., if all time variables have been fixed. If this is not the case, the algorithm proceeds in line with the constructive procedure. Based on the paths of the SCs that are given by the input solution of the routing component, the algorithm iterates over all SCs and fixes as many variables (2) and w_j, $j \in J$, as possible to their earliest possible time instants. The status of each SC is then updated. If no status has changed, the algorithm terminates because a feasible solution of MSCRB cannot be constructed based on the given routing solution. Otherwise, the algorithm continues as described above.

4. Computational study

We performed an extensive computational study based on real-world data of a port in Germany. The generation of the corresponding test instances is described in detail in Section 4.1. Our analysis was driven by the following main research questions:

- Q1: Does it pay off to deviate from the approach of permanently assigning a fixed number of SCs to each quay crane, which is the strategy that is currently implemented at the port?
- Q2: How does an increasing number of quay cranes (and, thus, jobs), an increasing number of SCs or larger buffer capacities influence the above findings?

We will elaborate on these questions in Section 4.4 after having answered some auxiliary questions dealing with the appropriateness of the proposed heuristic framework in Section 4.3:

- A1: What is the quality of the solutions determined by the heuristic framework when being compared with exact solutions, determined by calling CPLEX on model (4)–(26), for small instances? Is this quality sufficient for real-world usage of the heuristic framework?
- A2: Which setup of the heuristic framework (ejection chain vs. 3-Opt, random vs. greedy selection of SCs in the constructive procedure) performs best in terms of solution quality for medium and large instances that are inspired by real-world szenarios?
- A3: Are the runtimes of the heuristic framework (as analyzed in A2) in ranges that allow its usage in real-world scenarios?

All computational tests were executed on a PC with an Intel®Core™ i7-4770 CPU running at 3.4 GHz and 16 GB of RAM under a 64-bit version of Windows 8. We used a 64bit version of IBM ILOG CPLEX 12.7.

4.1. Instance generation

Our test instances were generated randomly, based on real-world data of a port in Germany. We received data for about 5000 containers, including their origin and destination locations, the SCs that were assigned to the containers, the distances travelled by the loaded SCs when transporting the containers, and the average speed of the SCs. Moreover, we received information on all relevant parameters of MSCRB, e.g., the time needed to lift or drop a container by a SC or to process a container by a quay crane.

All relevant locations of a problem instance were generated randomly in a Cartesian coordinate system in the plane, with the dimensions having been set according to the real-world data. The quay cranes are located at the waterfront, so that we arbitrarily set their y-coordinates to 0. The x-coordinates of the quay cranes were drawn from uniform distributions on the interval [0,600]. Similarly, all relevant origin or destination locations of jobs were drawn from uniform distributions over the intervals [0,600] (x-coordinates) and [50,350] (y-coordinates). Finally, the starting positions of the SCs were drawn from the intervals [0,600] (x-coordinates) and [0,350] (y-coordinates). All generated coordinates are integer valued. In order to align the generated locations with the real-world data, we set the length of a time unit as well as the value of p^v to one second. Moreover, in line with the real-world data, we set t^v to 20 seconds. Furthermore, we set $t^q = 4 \cdot t^v$, $t^s = t^b = 0.25 \cdot t^v$ (see also Soriguera & Espinet, 2006; Steenken et al., 2004).

In all of our test instances, a crane is a loading crane with probability 0.5. Otherwise, it is an unloading crane. The buffer capacity b_c is assumed to be equal for all cranes $c \in C$ of each test instance. Furthermore, each test instance features containers that are located in the buffer areas at the beginning of the planning horizon. In case of unloading cranes, we assume that the restacking sets of the corresponding jobs are empty. In case of loading cranes, we assume that these containers are the first ones to be processed by the cranes. For the sake of simplicity, all cranes, SCs, empty slots and containers are assumed to be immediately available, and the length of each nonempty restacking set is set to one container. Table 6 summarizes these basic assumptions for all test instances.

The parameters of our test instances, which we will apply for analyzing Q1–Q2 and A2–A3, are presented in Table 7, where $J^R := J \setminus \bigcup_{c \in C} J^u_c$. We generated five test instances for each parameter combination that results in an identical number of containers that have to be processed by each crane. This results in a total of 2160 instances. As can be seen from Table 7, the number of quay cranes n^q of our test instances varies between three and six. The current approach at the considered port is to permanently assign three to five SCs to each quay crane. Hence, we generated instances with $n^v \in \{3n^q, 4n^q, 5n^q\}$. The buffer capacity varies between three and five. As motivated in Section 1.2, the sequences of containers that have to be loaded onto or unloaded from the vessels by the quay cranes and that are known when routing the SCs are rather short. Currently, the considered port considers about 20 containers per quay crane, which is reflected in our test instances. As restacking operations are usually performed in off-peak times, the percentage of loading and unloading jobs that feature a nonempty restacking set is assumed to be rather small. It varies from 5% to 15% of the jobs that do not correspond to containers located in a buffer area at the beginning of the planning horizon.

In light of the computational complexity of MSCRB, we generated additional sets of small instances (with respect to the number of quay cranes and containers) in order to analyze auxiliary question A1. The corresponding parameters are presented in Table 8. Again, we generated five test instances for each parameter combination that results in an identical number of containers that have to be processed by each crane, i.e., a total of 100 instances.

4.2. Setup of the algorithms

We consider four variants of the heuristic framework presented in Section 3. Each variant combines a strategy of selecting SCs in the constructive procedure (Section 3.1) and a succeeding improvement procedure (Section 3.2) that guides the construction and evaluation (Section 3.2.3) of MSCRB solutions. The four variants are listed in Table 9.

Table 6
Instance generation: basic parameters.

t^q [seconds]	t^v [seconds]	t^s [seconds]	t^b [seconds]	a_c^q	a_k^v	a_j^o	a_j^r	a_{ic}^b	p^v [seconds]	$\lvert R_j\rvert$ if $R_j \neq \emptyset$
80	20	5	5	0	0	0	0	0	1	1

Table 7
Generation of real-world instances: Q1–Q2 and A2–A3.

		Loading cranes			Unloading cranes					
n^q	b_c	$\lvert J_c \setminus J_c^l\rvert$	$\lvert J_c^b\rvert$	J_c^v	$\lvert J_c\rvert$	$\lvert J_c^b\rvert$	J_c^q	n_c^b	n^v/n^q	$\lvert\{j \lvert j \in J^R, R_j \neq \emptyset\}\rvert$
3,4,5,6	3,4,5	10,14,18,22	$\lfloor 0.5b_c \rfloor$	$\emptyset$	10,14,18,22	$\lfloor 0.5b_c \rfloor$	$\emptyset$	$b_c - \lvert J_c^b\rvert$	3,4,5	$\lceil 0.05\lvert J^R\rvert \rceil$, $\lceil 0.1\lvert J^R\rvert \rceil$, $\lceil 0.15\lvert J^R\rvert \rceil$

Table 8
Generation of small instances: A1.

		Loading cranes			Unloading cranes					
n^q	b_c	$\lvert J_c \setminus J_c^l\rvert$	$\lvert J_c^b\rvert$	J_c^v	$\lvert J_c\rvert$	$\lvert J_c^b\rvert$	J_c^q	n_c^b	n^v/n^q	$\lvert\{j \lvert j \in J^R, R_j \neq \emptyset\}\rvert$
2,3	4	4,6,8,10,12	$\lfloor 0.5b_c \rfloor$	$\emptyset$	4,6,8,10,12	$\lfloor 0.5b_c \rfloor$	$\emptyset$	$b_c - \lvert J_c^b\rvert$	3,4	$\lceil 0.1\lvert J^R\rvert \rceil$

Table 9
Variants of the heuristic framework.

Name	Selection of SC	Improvement procedure
EC_g	Greedy	Ejection chain approach, Section 3.2.1
EC_r	Random	Ejection chain approach, Section 3.2.1
$3OPT_g$	Greedy	3-Opt approach, Section 3.2.2
$3OPT_r$	Random	3-Opt approach, Section 3.2.2

The parameters of the heuristic framework were set based on preliminary tests on randomly constructed instances. The improvement procedure within EC_g and EC_r is restarted four times (see Section 3.2.1), i.e., it is called five times in total. The first call is initiated with the MSCRB solution which has been determined by the constructive procedure. The succeeding calls are initiated with the MSCRB solutions that have been computed based on their respective previous calls. When restarting, the tabu list is cleared. The tabu length is set to $0.5 \cdot |V|$. A call of the ejection chain procedure is terminated, when the best solution determined within the call has not improved for $0.8 \cdot |V|$ iterations. In order to allow reasonable runtimes of the 3-Opt variants of the heuristic framework, $3OPT_g$ and $3OPT_r$ skip an edge triple with a probability of 50%. The threshold for the termination criterion introduced in Section 3.2.2 is set to the value of t^s.

In addition to the heuristic framework, we use CPLEX with a given time limit on model (4)–(26) as an additional approach for determining solutions to an instance of MSCRB. We rate the quality of a solution with objective function value $F^{alg}(I)$ returned by a specific algorithm $alg \in \{EC_g, EC_r, 3OPT_g, 3OPT_r, CPLEX\}$ for some given problem instance I of MSCRB with the ratio $F^{alg}(I)/F^*(I)$, where $F^*(I)$ is the best objective function value returned by any of the considered algorithms for problem instance I.

4.3. Auxiliary questions: evaluation of the heuristic framework

For auxiliary question A1 (small test instances, Table 8), we set the CPLEX time limit to one hour. Table 10 presents the computational results. For each set of instances, the table depicts the percentage of instances that CPLEX was able to solve to optimality (column "opt.") and the average time t_{avg} needed to compute these solutions. Additionally, it presents average quality ratios $qual_{avg}$ and computational times of the four variants of the heuristic framework when restricting the analysis to the instances that were solved to optimality by CPLEX. In the last row, the table additionally lists the overall average values of the corresponding columns.

As can be seen from Table 10, CPLEX is able to compute optimal solutions when the number of containers is sufficiently small. Furthermore, CPLEX benefits from an increasing number of SCs for a given number of quay cranes and containers. However, when aiming to compute optimal solutions, the computational results do not allow for using CPLEX for real-world instances that usually feature more than 12 containers per crane (see Section 4.1).

With respect to the heuristic framework, the greedy variants $3OPT_g$ and EC_g perform slightly better than the random variants $3OPT_r$ and EC_r. Except for the larger instances with $n^q = 3$ and $n^v = 12$, where the quality ratios of $3OPT_r$ become quite large, all variants of the heuristic framework are reasonable candidates for real-world usage, both with respect to the computational times and the solution qualities.

For the real-world test instances and auxiliary questions A2–A3, we set the CPLEX time limit to 180 seconds and make use of the best solution determined by CPLEX within this limit. Note that $180s/t^q = 2.25$, so that this is a reasonable limit for computational times in real-world online settings of MSCRB, where rescheduling is initiated with new container data whenever one of the SCs or quay cranes has completed all of its assigned jobs or operations.

With respect to auxiliary question A2, Table 11 presents the resulting average quality ratios for each group of instances with an identical number of quay cranes, containers per crane and SCs. For CPLEX, the table additionally includes the percentage of instances for which a feasible solution was found in parentheses. Again, the last row presents overall average values of the corresponding columns.

As CPLEX does not reliably return feasible solutions within the time limit, it is not an appropriate candidate for real-world usage. With respect to the heuristic framework, $3OPT_g$ and EC_g now clearly outperform their random counterparts in terms of solution quality. While EC_g benefits from an increasing number of SCs, $3OPT_g$ performs similarly well for all ratios n^v/n^q. Therefore, when not taking computational times into account, $3OPT_g$ seems most appropriate for practical applications.

In order to answer A3, Tables 12 and 13 present average and maximum runtimes of the algorithms for the real-world instances, respectively. Bold elements in the tables highlight the best value for the corresponding set of instances. It is immediately obvious that $3OPT_g$ features computational times that do not allow its

Table 10
Performance of CPLEX and the heuristic framework for small instances, CPLEX time limit: 3600 seconds.

			CPLEX		$3OPT_g$		EC_g		$3OPT_r$		EC_r	
n^q	n^v	$\lvert J_c\rvert^*$	opt. [%]	t_{avg} [seconds]	$qual_{avg}$	t_{avg} [seconds]	$qual_{avg}$	t_{avg} [seconds]	$qual_{avg}$	t_{avg} [seconds]	$qual_{avg}$	t_{avg} [seconds]
2	6	4	100	0.016	1.006	0.001	1.006	0.026	1.02	0.001	1.071	0.028
		6	100	0.407	1.023	0.006	1.032	0.086	1.013	0.008	1.029	0.087
		8	80	6.775	1.015	0.019	1.021	0.192	1.044	0.021	1.124	0.162
		10	40	1795.513	1.058	0.051	1.081	0.332	1.039	0.185	1.151	0.319
		12	0	–	–	–	–	–	–	–	–	–
2	8	4	100	0.017	1.002	0.001	1.002	0.038	1.003	0.002	1.035	0.041
		6	100	0.135	1.005	0.005	1.003	0.101	1.015	0.007	1.003	0.107
		8	80	85.691	1.076	0.018	1.029	0.164	1.05	0.033	1.036	0.181
		10	60	368.154	1.086	0.058	1.108	0.333	1.099	0.099	1.127	0.313
		12	0	–	–	–	–	–	–	–	–	–
3	9	4	100	0.296	1.012	0.006	1.012	0.1	1.101	0.005	1.062	0.104
		6	80	2.285	1.016	0.027	1.02	0.207	1.15	0.023	1.026	0.208
		8	20	752.436	1.074	0.139	1.096	0.479	1.292	0.016	1.108	0.515
		10	0	–	–	–	–	–	–	–	–	–
		12	20	0.081	1.015	0.006	1.015	0.119	1.044	0.01	1.067	0.155
3	12	4	100	0.091	1.011	0.009	1.015	0.182	1.012	0.015	1.059	0.178
		6	100	271.102	1.016	0.019	1.011	0.341	1.185	0.014	1.044	0.343
		8	80	177.558	1.046	0.088	1.044	0.591	1.481	0.106	1.088	0.658
		10	0	–	–	–	–	–	–	–	–	–
		12	0	–	–	–	–	–	–	–	–	–
avg.			59	133.858	1.025	0.022	1.025	0.192	1.093	0.029	1.06	0.197

*: $|J_c \setminus J_c^I|$ in case of loading cranes

Table 11
Average quality ratios for real world instances, CPLEX time limit: 180 seconds.

		$3OPT_g$			EC_g			$3OPT_r$			EC_r			*CPLEX*		
n^q	$\lvert J_c\rvert^*$	$\frac{n^v}{n^q}=3$	$\frac{n^v}{n^q}=4$	$\frac{n^v}{n^q}=5$	$\frac{n^v}{n^q}=3$	$\frac{n^v}{n^q}=4$	$\frac{n^v}{n^q}=5$	$\frac{n^v}{n^q}=3$	$\frac{n^v}{n^q}=4$	$\frac{n^v}{n^q}=5$	$\frac{n^v}{n^q}=3$	$\frac{n^v}{n^q}=4$	$\frac{n^v}{n^q}=5$	$\frac{n^v}{n^q}=3$	$\frac{n^v}{n^q}=4$	$\frac{n^v}{n^q}=5$
3	10	1.063	1.048	1.03	1.09	1.068	1.037	1.169	1.221	1.206	1.182	1.146	1.095	1 (100)	1 (100)	1 (100)
	14	1.018	1.038	1.028	1.052	1.046	1.041	1.103	1.213	1.32	1.184	1.169	1.167	1.048 (60)	1.019 (84.4)	1.008 (95.6)
	18	1.006	1.006	1.007	1.049	1.034	1.021	1.183	1.206	1.306	1.212	1.218	1.192	1.113 (20)	1.066 (17.8)	1.071 (31.1)
	22	1.003	1.004	1.005	1.055	1.03	1.025	1.159	1.302	1.294	1.257	1.233	1.229	1.279 (15.6)	1.267 (13.3)	1.138 (15.6)
4	10	1.041	1.034	1.027	1.058	1.052	1.027	1.212	1.326	1.355	1.178	1.14	1.096	1.01 (100)	1.001 (95.6)	1.001 (100)
	14	1.011	1.01	1.016	1.053	1.025	1.024	1.179	1.318	1.386	1.199	1.179	1.156	1.057 (22.2)	1.061 (37.8)	1.011 (37.8)
	18	1.003	1.003	1.003	1.043	1.019	1.014	1.232	1.331	1.355	1.232	1.201	1.202	1.257 (6.7)	1.127 (6.7)	1.131 (6.7)
	22	1.001	1.001	1.002	1.043	1.02	1.011	1.255	1.42	1.376	1.247	1.258	1.211	– (0)	– (0)	3.531 (2.2)
5	10	1.012	1.029	1.021	1.035	1.034	1.023	1.235	1.358	1.423	1.168	1.128	1.103	1.031 (80)	1.008 (86.7)	1 (82.2)
	14	1.003	1.005	1.004	1.031	1.017	1.011	1.275	1.376	1.363	1.192	1.171	1.138	1.155 (11.1)	1.086 (11.1)	1.074 (24.4)
	18	1.002	1.001	1.005	1.025	1.021	1.005	1.375	1.374	1.595	1.245	1.246	1.208	– (0)	– (0)	1.439 (2.2)
	22	1.001	1.002	1.004	1.034	1.013	1.008	1.395	1.481	1.564	1.279	1.256	1.247	– (0)	– (0)	– (0)
6	10	1.007	1.014	1.012	1.02	1.016	1.012	1.293	1.431	1.486	1.166	1.132	1.08	1.045 (55.6)	1.011 (68.9)	1.007 (66.7)
	14	1.001	1.003	1.008	1.022	1.012	1.006	1.324	1.447	1.45	1.198	1.179	1.135	1.476 (2.2)	1.196 (4.4)	1.651 (8.9)
	18	1.001	1.002	1.005	1.023	1.013	1.008	1.344	1.569	1.536	1.242	1.224	1.214	– (0)	– (0)	– (0)
	22	1	1.002	1.004	1.024	1.011	1.003	1.348	1.532	1.587	1.276	1.286	1.237	– (0)	– (0)	– (0)
avg.		1.011	1.013	1.011	1.041	1.027	1.017	1.255	1.369	1.413	1.216	1.198	1.169	1.046 (29.6)	1.024 (32.9)	1.039 (35.8)

*: $|J_c \setminus J_c^I|$ in case of loading cranes

Table 12
Average runtimes in seconds for real world instances, CPLEX time limit: 180 seconds.

		$3OPT_g$			EC_g			$3OPT_r$			EC_r			*CPLEX*		
n^q	$\lvert J_c\rvert^*$	$\frac{n^v}{n^q}=3$	$\frac{n^v}{n^q}=4$	$\frac{n^v}{n^q}=5$	$\frac{n^v}{n^q}=3$	$\frac{n^v}{n^q}=4$	$\frac{n^v}{n^q}=5$	$\frac{n^v}{n^q}=3$	$\frac{n^v}{n^q}=4$	$\frac{n^v}{n^q}=5$	$\frac{n^v}{n^q}=3$	$\frac{n^v}{n^q}=4$	$\frac{n^v}{n^q}=5$	$\frac{n^v}{n^q}=3$	$\frac{n^v}{n^q}=4$	$\frac{n^v}{n^q}=5$
3	10	0.5	0.5	0.4	0.9	1.1	1.3	**0.4**	**0.2**	**0.2**	1	1.1	1.4	180	171.3	114.8
	14	3	2.2	2.8	2.4	2.7	3.1	**2**	**0.9**	**0.5**	2.8	2.9	3.4	180	180	176.8
	18	12.1	10.9	9.3	5.6	6.3	6.8	**5.4**	**4.1**	**2.8**	6.1	7.1	7.9	180	180	180
	22	36.4	31.6	30.6	**9.4**	12.1	12.1	21.4	**10.7**	**6.3**	11.1	13.3	14.1	180	180	180
4	10	1.4	1.9	1.2	2.1	2.5	2.9	**0.8**	**0.5**	**0.2**	2.3	2.8	3.3	180	169.2	108.8
	14	11.2	10.5	10.1	5.9	6.5	8.1	**5.8**	**1.9**	**1.6**	6.6	7.4	9.2	180	180	180
	18	49.9	30.4	28.2	**13.5**	15.5	17.5	18.2	**5.4**	**5**	14.9	17.8	19.9	180	180	180
	22	138.4	117.5	49.5	**24.5**	29.5	30.4	42.3	**10.8**	**8.5**	28.3	35	35.7			180
5	10	4.8	3.9	3.5	4.1	5.1	5.7	**1.4**	**0.3**	**0.4**	4.8	5.6	6.6	180	176.7	134.8
	14	32.9	22.3	15.1	12.4	14.3	16.3	**8.4**	**3.2**	**1.3**	14.3	16.7	18.3	180	180	180
	18	81.7	100.4	42.7	26.5	32	35.2	**17.1**	**12.3**	**2**	31.2	38	40.1			180
	22	315.3	186.1	132.9	**56**	62.6	67.2	77.9	**22**	**5.1**	67.1	72.7	78.6			
6	10	10.2	4	5.8	7.5	9.1	10.8	**2.4**	**0.6**	**0.6**	8.6	9.7	12.2	180	175.7	153.9
	14	51.7	44	24.8	21.5	24.8	28.4	**6**	**3.9**	**1.4**	24.7	28.9	32.4	180	180	180
	18	191.5	164.5	132.3	48.5	56	67.1	**28.7**	**5.2**	**6.6**	58.3	64.5	77			
	22	523.6	405.1	134.2	94.5	110	117.4	**60.5**	**24.9**	**9.6**	113.7	129.5	131.5			

*: $|J_c \setminus J_c^I|$ in case of loading cranes

Table 13
Maximum runtimes in seconds for real world instances.

		$3OPT_g$			EC_g			$3OPT_r$			EC_r		
n^q	$\lvert J_c\rvert^*$	$\frac{n^v}{n^q}=3$	$\frac{n^v}{n^q}=4$	$\frac{n^v}{n^q}=5$	$\frac{n^v}{n^q}=3$	$\frac{n^v}{n^q}=4$	$\frac{n^v}{n^q}=5$	$\frac{n^v}{n^q}=3$	$\frac{n^v}{n^q}=4$	$\frac{n^v}{n^q}=5$	$\frac{n^v}{n^q}=3$	$\frac{n^v}{n^q}=4$	$\frac{n^v}{n^q}=5$
3	10	2.4	2.7	2.1	**1.5**	1.5	1.8	**1.5**	**0.8**	**1.7**	1.4	1.6	1.8
	14	12.8	11.4	16.6	**3.8**	**4.1**	4.7	9.1	5.5	**3.3**	3.9	4.4	5.1
	18	28	39.6	44.1	**8.4**	**8.7**	**10.3**	23.8	33.6	25.9	9.7	9.8	11
	22	145.2	123.4	228.3	**14.2**	**18.1**	**18.1**	77.1	68.9	53	18.3	21.4	21.4
4	10	5.3	6.3	7.1	**3**	**3.5**	3.9	3.6	5	**1.2**	**3**	3.7	4.3
	14	35.5	72.3	68.8	**8.7**	**9.5**	**11.5**	45.7	16.2	28.2	9.8	11.3	13.6
	18	290.3	201.2	175.1	**18.7**	**22.4**	**27.1**	126.6	58.5	65.5	21.5	26.9	29.4
	22	512.1	529.3	224.1	**39**	**45.4**	**50.4**	779.4	177.4	127.9	45.6	50.2	51.9
5	10	13.6	17.3	25.2	**5.7**	6.5	8.3	6.6	**1.7**	**3.3**	6.3	7.7	8.3
	14	122.6	144.3	110.7	**17.4**	**21.9**	24.1	56.8	35.5	**10.4**	19	24.2	25.1
	18	289.9	449.9	266.9	**39.7**	**52.4**	57	322.1	103.5	**18.3**	47.5	65.9	60.7
	22	1188.2	650.6	1466.4	**83.1**	**96.1**	99.4	1157.3	365.3	**67**	95.7	122	115.7
6	10	33.9	25.3	44.6	**11.3**	13.8	15.4	21.3	**4.4**	**14.3**	13.1	12.1	16
	14	367.4	328.5	146.1	**28.1**	**33.1**	41.4	34.8	39.7	**16.7**	34	41	43
	18	758	833.9	1039.6	**77.8**	**77.9**	98.5	226.6	60.3	**83.4**	85.9	96.6	108.9
	22	2649.9	2053.1	1455.6	**156**	**154.9**	174.5	339.2	300.3	**72.8**	165.8	201	182.7

*: $\lvert J_c \setminus J_c^l\rvert$ in case of loading cranes

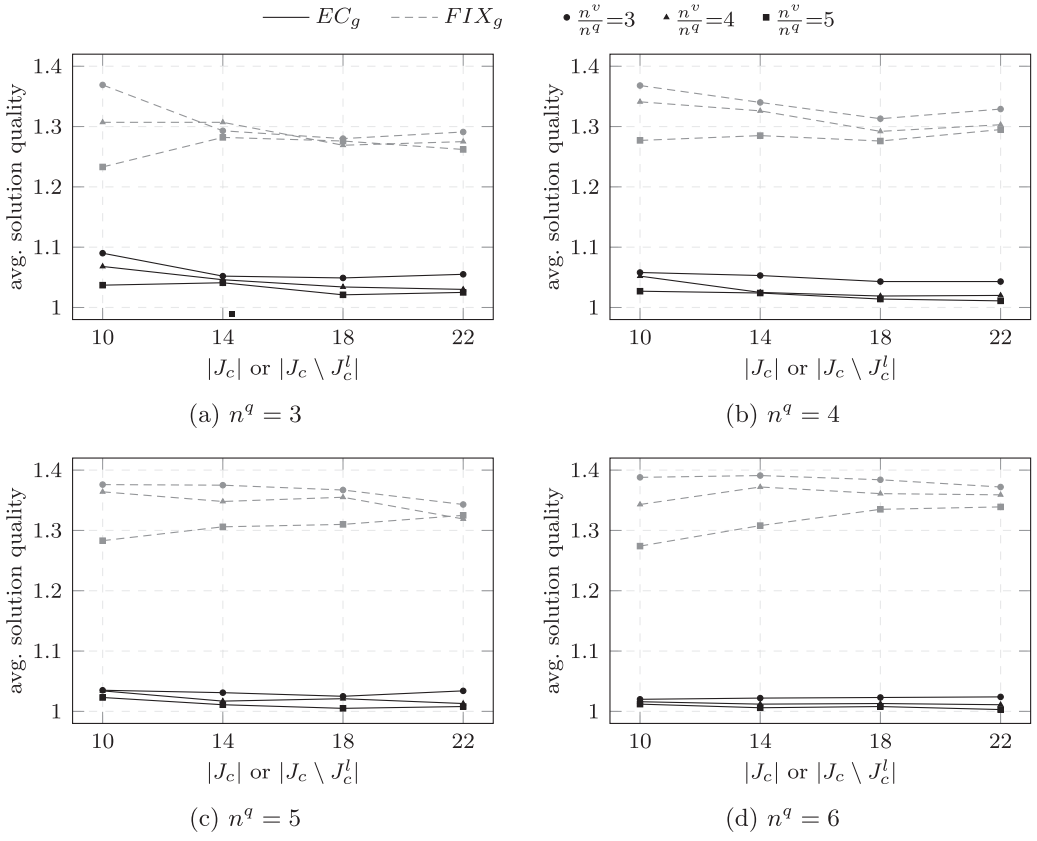

Fig. 11. Impact of relaxing fixed SC assignments: cranes and SCs.

usage in real-world online settings of MSCRB in most cases. All calls of EC_g, however, feature runtimes that are smaller than the 180 seconds time limit motivated above. Additionally, we found that the effect of varying buffer capacities on the average runtimes of EC_g is fairly small. Hence, when balancing runtimes and quality ratios, EC_g is the most appropriate setup of the heuristic framework in real-world settings.

4.4. Impact of relaxing fixed SC assignments

Based on our results in Section 4.3, we can now analyze our main research questions by solely focussing on EC_g. As described above, the current approach at the considered port is to permanently assign three to five SCs to each quay crane. That is, each SC solely processes jobs that are associated to its assigned crane. We simulate this approach by a heuristic referred to as FIX_g. Basically, this heuristic corresponds to sequentially calling the greedy version of our constructive procedure (Section 3.1) for each crane, its associated jobs (including the restacking jobs), and its associated SCs.

Fig. 11 illustrates the effect of relaxing fixed SC assignments by comparing the average quality ratios of EC_g and FIX_g for different groups of instances with a varying number of quay cranes and SCs. The depicted quality ratios take into account all variants of the heuristic framework and CPLEX as analyzed in Section 4.3.

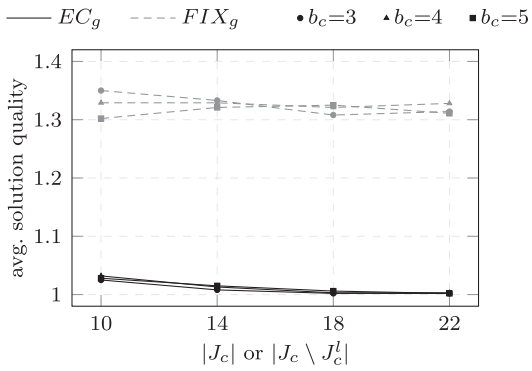

Fig. 12. Impact of relaxing fixed SC assignments: buffer capacity.

It can be seen that relaxing fixed SC assignments has a significant positive effect (research question Q1) on the objective function value of MSCRB: the difference of the quality ratios varies between about 0.2 and 0.37. With respect to Q2, we find that the number of containers per crane plays a minor role for the achievable time savings, whereas the differences in the quality ratios tend to increase for an increasing number of quay cranes (and, thus, jobs) as well as for a decreasing number of SCs, where the latter effect is more distinct for larger numbers of quay cranes.

Fig. 12 presents the average quality ratios when focussing on varying buffer capacities in line with Fig. 11. As can be seen, the effect of varying capacities on the achievable savings is rather small.

5. Conclusion

In this paper, we have considered a SC routing problem that arises at container ports where quay cranes are equipped with buffer areas of limited capacity that allow short time storage of containers. The SCs are in charge of transporting containers between these buffer areas and the stacking areas of the port, while having to respect given unloading and loading sequences of the containers at the quay cranes. The objective of the routing problem represents the overall goal to minimize the turnaround times of the vessels at the port. In contrast to the current approach at the considered port, we do not fix the assignment of SCs to quay cranes.

We have provided a proof for the strong NP-hardness of the problem under consideration. Furthermore, we have presented an integer program based on an asymmetric traveling salesman problem with precedence constraints. We have then introduced a heuristic framework, that decomposes the problem into a routing component and a component that handles the time variables and the buffer capacities. Computational tests have provided evidence for the applicability of a variant of this framework that applies an ejection chain approach for the routing component in real-world online settings. Furthermore, seeing significant improvements in the objective function value when freely routing the SCs, we have shown that the relaxation of the fixed SC assignments is an interesting strategy to be considered by port authorities.

There remain several interesting questions to be answered in future research. Potential limitations of our model could, for example, be evaluated in a simulation study. Other relevant questions are concerned with generalizations or modifications of the problem studied in this article in order to include more details of the port layout, other transportation vehicles (AGVs, YTs, ALVs, etc.), or more general loading or unloading strategies at the quay cranes. In order to achieve high crane utilization rates, for example, some terminal operators seek to implement crane double cycling strategies. Due to the fact that this results in cranes and SCs dropping containers in the same buffer areas, this oftentimes results in deadlock situations in practice (see, for example, Carlo et al., 2014b; Lehmann, Grunow, & Günther, 2006). In order to prevent or resolve such deadlocks, our model and algorithms may be extended appropriately. Similarly, one can explicitly model waiting periods of SCs in order to prevent congestion situations or include one-way travel settings or other details of the port layout that are frequently applied in practice. Further research could also be undertaken to investigate the potential of integrating decisions of higher level optimization problems into our model, e.g., container sequencing decisions at the quay cranes or decisions on destination locations of containers in the storage areas, which we have treated as parameters is this study.

References

Angeloudis, P., & Bell, M. (2010). An uncertainty-aware AGV assignment algorithm for automated container terminals. *Transportation Research Part E: Logistics and Transportation Review, 46*(3), 354–366.

Bierwirth, C., & Meisel, F. (2010). A survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research, 202*(3), 615–627.

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.

Bish, E. K. (2003). A multiple-crane-constrained scheduling problem in a container terminal. *European Journal of Operational Research, 144*(1), 83–107.

Bish, E. K., Chen, F. Y., Leong, Y. T., Nelson, B. L., Ng, J. W. C., & Simchi-Levi, D. (2005). Dispatching vehicles in a mega container terminal. *OR Spectrum, 27*(4), 491–506.

Böse, J., Reiners, T., Steenken, D., & Voß, S. (2000). Vehicle dispatching at seaport container terminals using evolutionary algorithms. In *Proceedings of the 33rd annual Hawaii international conference on system sciences* (pp. 1–10). IEEE.

Briskorn, D., Drexl, A., & Hartmann, S. (2006). Inventory-based dispatching of automated guided vehicles on container terminals. *OR Spectrum, 28*(4), 195–214.

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

Cavique, L., Rego, C., & Themido, I. (1999). Subgraph ejection chains and tabu search for the crew scheduling problem. *Journal of the Operational Research Society, 50*(6), 608–616.

Dorndorf, U., Jaehn, F., & Pesch, E. (2008). Modelling robust flight-gate scheduling as a clique partitioning problem. *Transportation Science, 42*(3), 292–301.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability - A guide to the theory of NP-completeness*. New York: Freeman.

Glover, F. (1996). Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics, 65*(1), 223–253.

Goodchild, A. V., & Daganzo, C. F. (2007). Crane double cycling in container ports: Planning methods and evaluation. *Transportation Research Part B: Methodological, 41*(8), 875–891.

Grunow, M., Günther, H.-O., & Lehmann, M. (2004). Dispatching multi-load AGVs in highly automated seaport container terminals. *OR Spectrum, 26*(2), 211–235.

Grunow, M., Günther, H.-O., & Lehmann, M. (2006). Strategies for dispatching AGVs at automated seaport container terminals. *OR Spectrum, 28*(4), 587–610.

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

Kim, K. H., & Bae, J. W. (2004). A look-ahead dispatching method for automated guided vehicles in automated port container terminals. *Transportation Science, 38*(2), 224–234.

Kim, K. H., & Kim, K. Y. (1999). Routing straddle carriers for the loading operation of containers using a beam search algorithm. *Computers & Industrial Engineering, 36*(1), 109–136.

Korte, B., & Vygen, J. (2012). *Combinatorial optimization - Theory and algorithms* (5th). Berlin: Springer.

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., Boysen, N., & Pesch, E. (2017). Which items should be stored together? A basic partition problem to assign storage space in group-based storage systems. *IISE Transactions, 49*(1), 13–30.

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.

Kress, D., Meiswinkel, S., & Pesch, E. (2015). The partitioning min-max weighted matching problem. *European Journal of Operational Research, 247*(3), 745–754.

Kuzmicz, K. A., & Pesch, E. (2019). Approaches to empty container repositioning problems in the context of Eurasian intermodal transportation. *Omega, 85*, 194–213.

Laporte, G. (1992). The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research, 59*(2), 231–247.

Lee, C.-Y., & Song, D.-P. (2017). Ocean container transport in global supply chains: Overview and research opportunities. *Transportation Research Part B: Methodological, 95*, 442–474.
Lee, D.-H., Cao, J. X., Shi, Q., & Chen, J. H. (2009). A heuristic algorithm for yard truck scheduling and storage allocation problems. *Transportation Research Part E: Logistics and Transportation Review, 45*(5), 810–820.
Lehmann, M., Grunow, M., & Günther, H.-O. (2006). Deadlock handling for real-time control of AGVs at automated container terminals. *OR Spectrum, 28*(4), 631–657.
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.
Meiswinkel, S. (2018). *On combinatorial optimization and mechanism design problems arising at container ports*. Springer.
Ng, W. C., Mak, K. L., & Zhang, Y. X. (2007). Scheduling trucks in container terminals using a genetic algorithm. *Engineering Optimization, 39*(1), 33–47.
Nguyen, V. D., & Kim, K. H. (2009). A dispatching method for automated lifting vehicles in automated port container terminals. *Computers & Industrial Engineering, 56*(3), 1002–1020.
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.
Pesch, E., & Glover, F. (1997). TSP ejection chains. *Discrete Applied Mathematics, 76*(1), 165–181.
Rego, C., & Glover, F. (2010). Ejection chain and filter-and-fan methods in combinatorial optimization. *Annals of Operations Research, 175*(1), 77–105.
Rego, C., James, T., & Glover, F. (2010). An ejection chain algorithm for the quadratic assignment problem. *Networks, 56*(3), 188–206.
Skinner, B., Yuan, S., Huang, S., Liu, D., Cai, B., Dissanayake, G., Lau, H., Bott, A., & Pagac, D. (2013). Optimisation for job scheduling at automated container terminals using genetic algorithm. *Computers & Industrial Engineering, 64*(1), 511–523.
Soriguera, F., & Espinet, D. (2006). A simulation model for straddle carrier operational assessment in a marine container terminal. *Journal of Maritime Research, 3*(2), 19–34.
Stahlbock, R., & Voß, S. (2008). Operations research at container terminals: a literature update. *OR Spectrum, 30*(1), 1–52.
Stahlbock, R., & Voß, S. (2008). Vehicle routing problems and container terminal operations–an update of research. In B. Golden, S. Raghavan, & E. Wasil (Eds.), *The vehicle routing problem: Latest advances and new challenges* (pp. 551–589). New York: Springer.
Steenken, D., Voß, S., & Stahlbock, R. (2004). Container terminal operation and operations research - a classification and literature review. *OR Spectrum, 26*(1), 3–49.
Vis, I. F. A., de Koster, R. M. B. M., & Savelsbergh, M. W. P. (2005). Minimum vehicle fleet size under time-window constraints at a container terminal. *Transportation Science, 39*(2), 249–260.
Yagiura, M., Ibaraki, T., & Glover, F. (2004). An ejection chain approach for the generalized assignment problem. *INFORMS Journal on Computing, 16*(2), 133–151.