European Journal of Operational Research 279 (2019) 767–781

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Production, Manufacturing, Transportation and Logistics

An exact algorithm for the block relocation problem with a stowage plan

Shunji Tanaka[a,*], Stefan Voß[b]

[a] Institute for Liberal Arts and Sciences & Department of Electrical Engineering, Kyoto University, Kyotodaigaku-Katsura, Nishikyo-ku, Kyoto City, 615–8510 Kyoto, Japan
[b] Institute of Information Systems, University of Hamburg, Von-Melle-Park 5, Hamburg 20146, Germany

ARTICLE INFO

Article history:
Received 11 October 2018
Accepted 6 June 2019
Available online 11 June 2019

Keywords:
OR in maritime industry
Combinatorial optimization
Block relocation problem with a stowage plan
Branch-and-bound algorithm
Lower bounds

ABSTRACT

In this paper we address an exact algorithm for the block relocation problem with a stowage plan. This problem is an extension of the container (block) relocation problem that aims at minimizing the total number of relocations required for retrieving containers piled up in a container yard. The difference is that a stowage plan in a vessel is given in advance and it is taken into consideration when retrieving containers. More specifically, the retrieval order of containers should be feasible with respect to a given stowage plan. We construct a branch-and-bound algorithm with iterative deepening for unrestricted and restricted variants of this problem. To this end, we propose three lower bounds on the total number of relocations. We also extend dominance rules for the container relocation problem, in order to restrict the search space and improve the search efficiency. The effectiveness of the proposed algorithm is examined by numerical experiments for benchmark instances from the literature.

1. Introduction

Container transportation is a major means of maritime freight nowadays, and the number of containers handled in sea ports has been constantly increasing in recent decades. According to the statistics of the United Nations Conference on Trade and Development, the world container port throughput is 752 million TEUs (Twenty-foot Equivalent Units) in 2017, which exhibits a 34% growth from 560 million TEUs in 2010 (UNICTAD, 2018). To handle this many containers properly without causing serious delays, effective operations in sea ports are of crucial importance.

Container terminals in sea ports provide temporal storage for connecting different transportation modes: Incoming containers via maritime and land transport are stored temporarily in a storage yard until they are retrieved for further maritime and land transport. Primarily due to space limitation, containers are usually piled up vertically in tiers in the yard. Since cranes can access only the topmost containers, relocations of containers within the yard are inevitable, unless their retrieval order is known in advance and, at the same time, containers are stacked in accordance with the order. Roughly speaking, three types of problems have been studied so far (Caserta, Schwarze, & Voß, 2011a) to reduce unproductive relocations and to achieve a better container handling performance. The stacking problem determines appropriate storage locations of incoming containers using incomplete and often stochastic information of their retrieval order. The pre-marshalling problem and the re-marshalling problem rearrange containers within the yard off-line so that no relocations are necessary when they are actually retrieved. The relocation problem retrieves containers from the yard while avoiding relocations as much as possible. This paper focuses on the relocation problem, which is referred to as the container relocation problem or the block(s) relocation problem in the literature.

In the ordinary setting of the relocation problem, each container is assigned an integer priority value, and we aim at retrieving containers in nondecreasing order of their priority values so as to minimize the total number of relocations. The majority of existing studies on the relocation problem assume that the priority values are distinct and thus the retrieval order is unique, although in practice it depends on several factors such as departure times, destinations, stowage plans, and so on. Granted that assigning the same priority value to a group of containers enables us to treat a more general situation where the retrieval order within each group is arbitrary, the current model that specifies the retrieval order by priority values is restrictive. To alleviate this, we address an extension of the relocation problem, assuming that the containers are loaded onto a vessel. In this problem, the retrieval order is determined by a stowage plan in the vessel. For example, if container X is stacked below (above) container Y in the vessel, we must re-

[*] Corresponding author.
E-mail addresses: tanaka@kuee.kyoto-u.ac.jp (S. Tanaka), stefan.voss@uni-hamburg.de (S. Voß).

https://doi.org/10.1016/j.ejor.2019.06.014
0377-2217/© 2019 Elsevier B.V. All rights reserved.

trieve container X before (resp. after) container Y from the yard. Otherwise, the retrieval order between them is arbitrary. Following Ji, Guo, Zhu, and Yang (2015), we refer to this problem as the block relocation problem with a stowage plan (BRPSP).

The purpose of this paper is to develop an efficient exact algorithm for the BRPSP. We propose three lower bounds on the total number of relocations, and construct a branch-and-bound algorithm with iterative deepening. Then, we demonstrate the effectiveness of the proposed algorithm through computational experiments.

The remainder of this paper is organized as follows. Section 2 gives a brief survey of relevant studies. Section 3 states a formal description of the BRPSP. Section 4 provides notation and definitions used throughout this paper. Properties of the retrieval order are investigated and summarized here as well. Section 5 proposes three lower bounds on the objective value. They are employed in the branch-and-bound algorithm constructed in Section 6. Section 6 also proposes dominance rules for reducing unpromising nodes in the search tree, and a simple greedy heuristic for computing an upper bound. Section 7 examines the effectiveness of the proposed algorithm as well as the lower bounds through computational experiments. Section 8 summarizes the results obtained in this paper.

2. Literature review

In this section, we briefly review the literature on the relocation problem and the pre-marshalling problem. For more details on these and related problems, please refer to the survey papers Caserta et al. (2011a) and Lehnfeld and Knust (2014).

The optimization model of the relocation problem originates from Kim and Hong (2006). They develop a branch-and-bound algorithm and a greedy heuristic for the problem. Following their study, several researchers have tackled this problem. Its NP-hardness is proved (Caserta, Schwarze, and Voß, 2012; van Brink and van der Zwaan, 2014), and heuristics (Azari, Eskandari, & Nourmohammadi, 2017; Caserta, Schwarze, & Voß, 2009; Caserta & Voß, 2009b; Caserta, Voß, & Sniedovich, 2011b; Expósito-Izquierdo, Melián-Batista, & Moreno-Vega, 2014; Forster & Bortfeldt, 2012; Jin, Zhu, & Lim, 2015; Jovanovic & Voß, 2014; Jobvanovic, Tuba & Voß, 2019; Lee & Lee, 2010; Lin, Lee, & Lee, 2015; Petering & Hussein, 2013; Ting & Wu, 2017; Tricoire, Scagnetti, & Beham, 2018; Ünlüyurt & Aydın, 2012; Zhu, Qin, Lim, & Zhang, 2012), mathematical formulations (Caserta et al., 2012; Eskandari & Azari, 2015; Expósito-Izquierdo et al., 2014; Galle, Barnhart, & Jaillet, 2018; Petering & Hussein, 2013; Wan, Liu, & Tsai, 2009; Zehendner, Caserta, Feillet, Schwarze, & Voß, 2015), de Melo da Silva, Toulouse, and Wolfler Calvo (2018b), and exact algorithms (Expósito-Izquierdo et al., 2014; 2015; Ku & Arthanari, 2016; Quispe, Lintzmayer, & Xavier, 2018; de Melo da Silva, Erdoğan, Battarra, & Strusevich, 2018a; Tanaka & Mizuno, 2018; Tanaka & Takii, 2016; Tricoire et al., 2018; Ünlüyurt & Aydın, 2012; Zehendner & Feillet, 2014; Zhu et al., 2012) have been proposed. Some of these studies aim at minimizing the total crane working time (Azari et al., 2017; Galle et al., 2018; Lee & Lee, 2010; Lin et al., 2015; da Silva Firmino, de Abreu Silva, & Times, 2019; Ünlüyurt & Aydın, 2012), but the majority adopt the total number of relocations as the objective function to be minimized. For a note on the relation of these objectives see Schwarze and Voß (2014). In addition to the objective function, the relocation problem can be classified from the following two aspects. The first classification concerns relocatable containers. In the restricted variant of the problem, we can relocate only the topmost container above the container to be retrieved next. In contrast, any of the topmost containers is relocatable in the unrestricted variant of the problem. It follows that we must determine the relocated container as well as its destination in the latter problem. Despite this difficulty, the unrestricted problem is considered in several studies (see Caserta et al., 2012; Expósito-Izquierdo et al., 2014; Forster & Bortfeldt, 2012; Galle et al., 2018; Jin et al., 2015; Kim & Hong, 2006; Petering & Hussein, 2013; de Melo da Silva et al., 2018b; Tanaka & Takii, 2016; Tricoire et al., 2018; Zhu et al., 2012). The second classification is, as already mentioned, in terms of priorities of containers. In the problem with distinct priorities, the retrieval order of containers is unique, whereas the retrieval order among containers belonging to the same group is arbitrary in the problem with duplicate (group) priorities. Only a small number of studies consider the latter problem (see Forster & Bortfeldt, 2012; Jin et al., 2015; Kim & Hong, 2006; de Melo da Silva et al., 2018a; de Melo da Silva et al., 2018b; Tanaka & Takii, 2016).

In the pre-marshalling problem, unlike the relocation problem, we do not retrieve any containers but only rearrange them within the yard in accordance with their retrieval priorities. Another difference is that retrieval priorities are in general assumed to be possibly duplicated in the pre-marshalling problem. Thus, we do not treat the problem with distinct priorities separately. The pre-marshalling problem is also well-studied in the literature, and heuristics (Bortfeldt & Forster, 2012; Caserta & Voß, 2009a; Expósito-Izquierdo, Melián-Batista, & Moreno-Vega, 2012; Hottung & Tierney, 2016; Huang & Lin, 2012; Jovanovic, Tuba, & Voß, 2017; Lee & Chao, 2009; Wang, Jin, & Lim, 2015; Wang, Jin, Zhang, & Lim, 2017), mathematical formulations (Lee & Hsu, 2007; de Melo da Silva et al., 2018b), and exact algorithms (van Brink & van der Zwaan, 2014; Tanaka & Tierney, 2018; Tierney, Pacino, & Voß, 2017; Zhang, Jiang, & Yun, 2015) have been proposed. Its NP-hardness can be proved in a similar manner to the relocation problem (van Brink & van der Zwaan, 2014; Caserta et al., 2011a).

In these studies for the relocation problem and the pre-marshalling problem, the retrieval order is determined using priorities of containers. There exist some extensions of the problems in which the retrieval order is not determined simply by the priorities. López-Plata, Expósito-Izquierdo, Lalla-Ruiz, Melián-Batista, and Moreno-Vega (2017) consider a variant of the relocation problem where a due date is given for each container and the objective is to minimize the total tardiness. In this problem, we need not always retrieve containers in EDD (earliest due date) order if a penalty is paid for the delay. Tierney and Voß (2016) study a robust version of the pre-marshalling problem where each container has to be retrieved within its due window. To the best of the authors' knowledge, an extension of the relocation problem that considers a stowage plan when retrieving containers was proposed for the first time by Ji et al. (2015). They treat this problem as a bi-level decision problem. In the first level, the retrieval order of containers is determined by a genetic algorithm. Given the retrieval order, destination stacks of relocated containers are determined by three heuristics in the second level. Here, only containers above the container to be retrieved next are relocated, meaning that their problem is the restricted variant of the BRPSP. Jovanovic, Tanaka, Nishi, and Voß (2018) construct greedy heuristics and GRASPs (Greedy Randomized Adaptive Search Procedures) for the restricted BRPSP. However, the unrestricted BRPSP has never been studied to date, although a better solution with fewer relocations is expected by removing the restriction on relocatable containers. Moreover, no exact algorithms are currently available even for the restricted BRPSP. These facts motivate us to develop an exact algorithm for both unrestricted and restricted variants of the BRPSP, which is the primary purpose of this paper.

3. Block relocation problem with a stowage plan

In this section, we give a formal description of the BRPSP addressed in this paper.

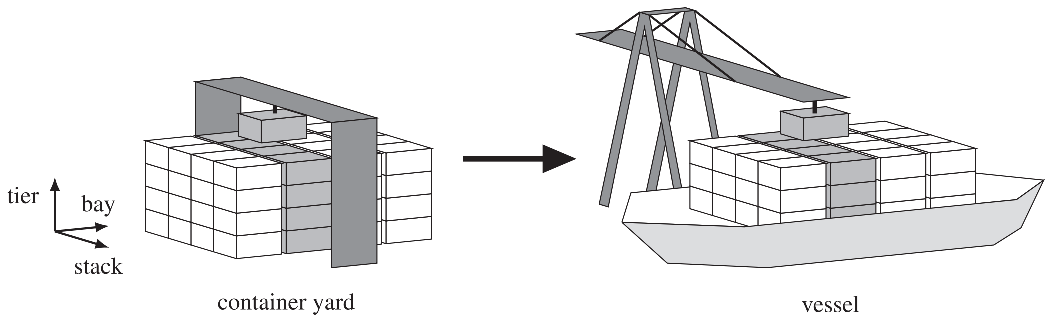


Fig. 1. Transferring containers from a bay of a container yard to a bay of a vessel.

Suppose that containers of the same size are stored in a container yard, and some of them are transferred onto a vessel as illustrated in Fig. 1. As is often the case with the relocation problem, only one yard bay and one vessel bay are considered because moving a crane across bays is time-consuming, compared with moves within one bay. Thus, we are to transfer N containers (container 1, container 2, …, container N) in a yard bay to a vessel bay. The yard bay is composed of S^{Y} stacks whose heights are limited to T^{Y}: The maximum number of containers that can be placed in each yard stack is T^{Y}. The vessel bay has S^{V} stacks. Each slot of the yard bay and the vessel bay is denoted by a pair of stack s and tier t as (s, t), where the tiers are numbered as 1, 2, … from bottom to top. The initial slot of container c in the yard bay is given by $(s_c^{\mathrm{Y}}, t_c^{\mathrm{Y}})$. Since we cannot place a container above an empty slot, there always exists a container d such that $(s_d^{\mathrm{Y}} = s_c^{\mathrm{Y}}) \wedge (t_d^{\mathrm{Y}} = t_c^{\mathrm{Y}} - 1)$, if container c satisfies $t_c^{\mathrm{Y}} > 1$. The destination slot of container c in the vessel bay is given in advance by $(s_c^{\mathrm{V}}, t_c^{\mathrm{V}})$. Clearly, there exists container d such that $(s_d^{\mathrm{V}} = s_c^{\mathrm{V}}) \wedge (t_d^{\mathrm{V}} = t_c^{\mathrm{V}} - 1)$ for any container c satisfying $t_c^{\mathrm{V}} > 1$. We can transfer a container to its destination slot in the vessel bay, if it is placed on top of a yard stack and, at the same time, its destination slot is on top of a vessel stack. If no such container exists, we must relocate containers. We assume that this relocation is permitted only within the yard bay and containers can never be moved again after they are placed in the vessel bay. Therefore, the following two types of operations are permitted:

- Relocation
  A container on top of a yard stack is moved to the top of another yard stack, if it is not full.
- Retrieval (Transfer)
  A container on top of a yard stack is moved to the top of a vessel stack, if this location is its destination slot.

In the restricted BRPSP, an additional constraint is imposed on relocations:

- Only the topmost container above the container to be retrieved next can be relocated.

The objective of the BRPSP is to minimize the total number of relocations necessary for transferring all the N containers from the yard bay to the vessel bay. It is worth noting that the problem becomes equivalent to the relocation problem if the vessel bay is composed only of one stack. Thus, the BRPSP is NP-hard from the NP-hardness of the relocation problem.

An initial yard bay configuration is illustrated in Fig. 2. In this figure as well as the other figures in this paper, vessel stacks are named alphabetically as A, B, and so on, and containers are denoted by their destination slots. For example, the destination slot of containers A1 and C3 are the lowest tier of vessel stack A and the third lowest tier of vessel stack C, respectively. An optimal solution of this instance as the unrestricted BRPSP is shown in Fig. 3. It is composed of four relocations: container C2 from yard stack 1 to 2, F3 from 2 to 1, B2 from 5 to 2, and A3 from 2 to 3. We need not consider retrievals explicitly when constructing a solution, since we can assume that retrievable containers are retrieved as soon as possible. Note that this solution is infeasible for the restricted BRPSP because container F3 is relocated from yard stack 2, although container E1 is retrieved next from yard stack 5 (it is retrieved first in Fig. 3f). The details are omitted here, but we can confirm that interchanging the order of the relocations of containers F3 and E1 makes the solution feasible for the restricted BRPSP without increasing the total number of relocations.

4. Notation, definitions, and precedence cycles

In this section, we first summarize basic notation and definitions used throughout this paper. Then, we investigate properties of cycles in retrieval precedence relations. Please also refer to Table 1.

4.1. Basic notation and definitions

Let us represent the current yard (and vessel) bay configuration by $\mathcal{C}$. We write $c \in \mathcal{C}$ if container c is in the yard bay, and $c \notin \mathcal{C}$ otherwise (in the vessel bay). The current slot of container $c \in \mathcal{C}$ in the yard bay is denoted by $(s_c^{\mathrm{Y}}(\mathcal{C}), t_c^{\mathrm{Y}}(\mathcal{C}))$. The set of containers placed in yard stack y in $\mathcal{C}$ is denoted by $\mathcal{Y}_y(\mathcal{C})$. The sets of containers already placed and not yet placed in the destination vessel stack s_c^{V} of container c in $\mathcal{C}$ are denoted by $\mathcal{V}_c(\mathcal{C})$ and $\check{\mathcal{V}}_c(\mathcal{C})$, respectively.

The target group is composed of containers in the yard bay whose destination slot is on top of the corresponding vessel stack. Each container in the target group is referred to as a target container. One of the target containers is retrieved from the yard bay next. We denote the target group in $\mathcal{C}$ by $\mathcal{G}(\mathcal{C})$. The target stacks are the yard stacks where at least one target container is placed. The set of target stacks in $\mathcal{C}$ is denoted by $\mathcal{T}(\mathcal{C})$. The topmost target container in a target stack y $(\in \mathcal{T}(\mathcal{C}))$ is denoted by $g_y(\mathcal{C})$. Stacking containers in target stack y are those placed above container $g_y(\mathcal{C})$. We denote by $\mathcal{S}_y(\mathcal{C})$ the set of stacking containers in target stack y. Hereafter, the argument "$(\mathcal{C})$" in each notation is omitted if there is no ambiguity.

The relocation of a container c from a yard stack s to a yard stack d is denoted by the triplet (c, s, d). A feasible solution of the problem is represented by a sequence of feasible relocations $(c_1, s_1, d_1), (c_2, s_2, d_2), \ldots, (c_n, s_n, d_n)$. We also define the relocation function $L_{csd}(\mathcal{C})$ to express the bay configuration obtained by applying (c, s, d) to $\mathcal{C}$. Similarly, the retrieval function $R(\mathcal{C})$ denotes the bay configuration after all retrievable containers are retrieved in $\mathcal{C}$.

4.2. Precedence graph and cycles

Let us consider two containers c and d in the yard bay in a bay configuration $\mathcal{C}$. If $s_c^{\mathrm{Y}}(\mathcal{C}) = s_d^{\mathrm{Y}}(\mathcal{C})$ and $t_c^{\mathrm{Y}}(\mathcal{C}) > t_d^{\mathrm{Y}}(\mathcal{C})$, that is, if

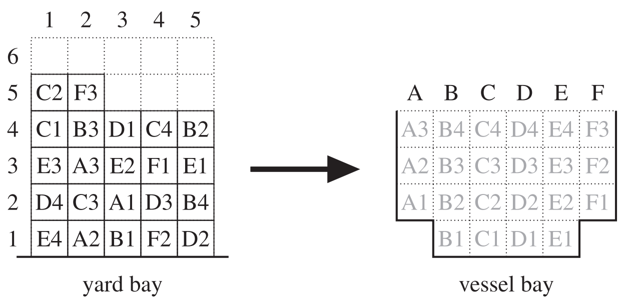


Fig. 2. An initial yard bay configuration ($S^Y = 5$, $T^Y = 6$, $S^V = 6$).

container c is placed above container d in the same yard stack, we cannot relocate or retrieve container d before container c. If $s_c^V = s_d^V$ and $t_c^V < t_d^V$, that is, if the destination slot of container c is below that of container d in the same vessel stack, we cannot retrieve container d before container c. These facts yield precedence relations between containers from top to bottom in a yard stack, and from bottom to top in a vessel stack. In Jovanovic et al. (2018), these precedence relations are investigated in detail and some properties are derived, which we introduce in the following.

Let us define a yard precedence relation $\xrightarrow{y}$ and a vessel precedence relation $\xrightarrow{v}$ in $\mathcal{C}$ as follows:

$$c \xrightarrow{y} d \quad \text{iff} \quad (s_c^Y(\mathcal{C}) = s_d^Y(\mathcal{C})) \wedge (t_c^Y(\mathcal{C}) > t_d^Y(\mathcal{C})), \tag{1}$$

$$c \xrightarrow{v} d \quad \text{iff} \quad (s_c^V = s_d^V) \wedge (t_c^V < t_d^V). \tag{2}$$

Accordingly, we introduce a digraph $G = (V, A)$, where

$$V := \{c \mid c \in \mathcal{C}\}, \tag{3}$$

$$A := \{(c, d) \in \mathcal{C} \times \mathcal{C} \mid (c \xrightarrow{y} d) \vee (c \xrightarrow{v} d)\}. \tag{4}$$

It is obvious that all yard containers in $\mathcal{C}$ can be retrieved without any relocations if the corresponding precedence graph G does not contain any cycle. In other words, cycles in G prevent us from retrieving containers and thus we must disconnect all of them by relocating containers to retrieve all the containers. To classify the cycles, we define a minimal cycle as follows: A cycle $a = \{(c_1, c_2), (c_2, c_3), \dots, (c_n, c_1)\} \subset A$ is called *minimal* iff for any $V' \subset \{c_1, c_2, \dots, c_n\}$ such that $|V'| < n$, the subgraph of G induced by V' is acyclic. In short, a cycle is minimal if nodes composing it do not form any subcycle. In Jovanovic et al. (2018), it is proved

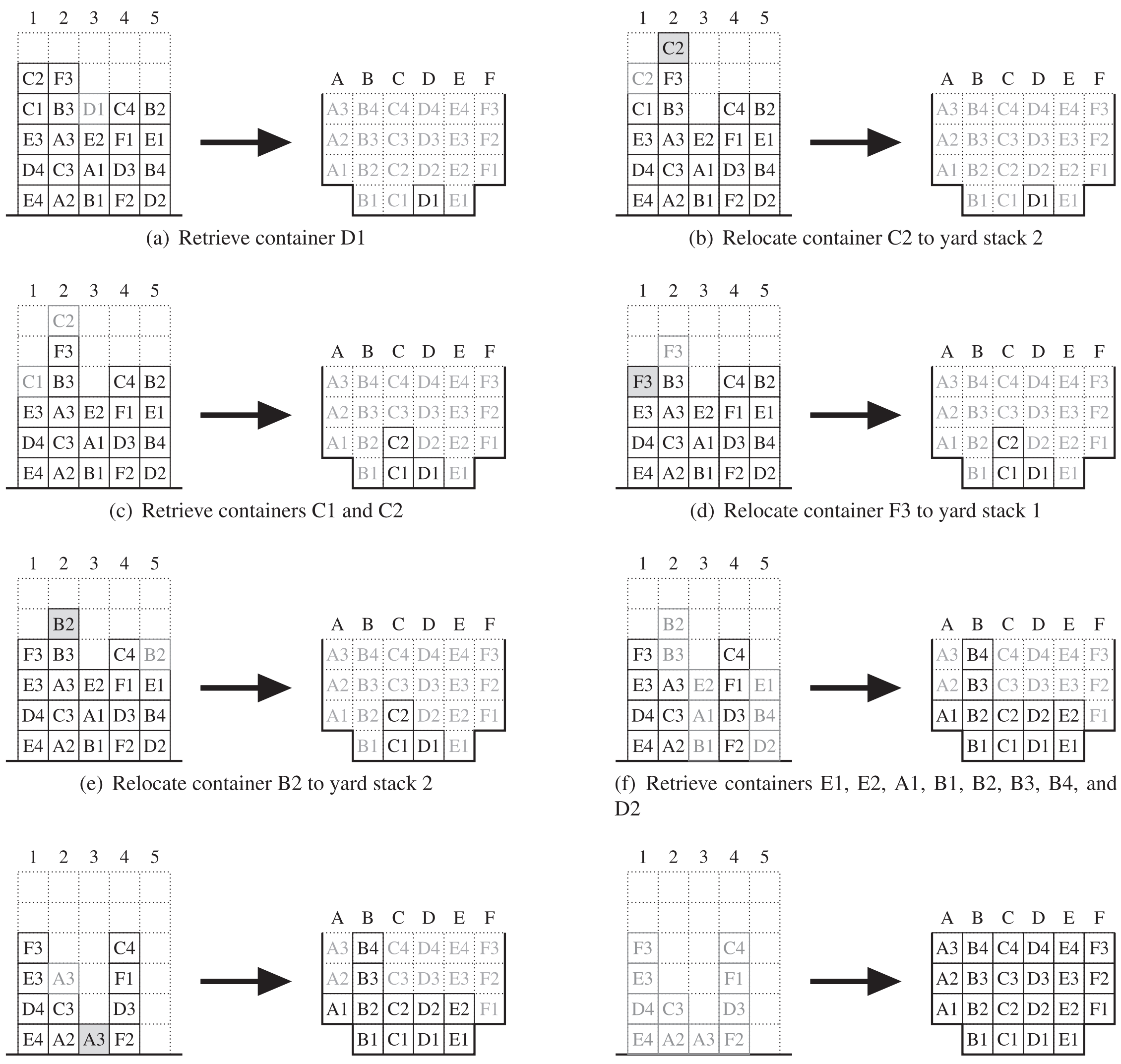
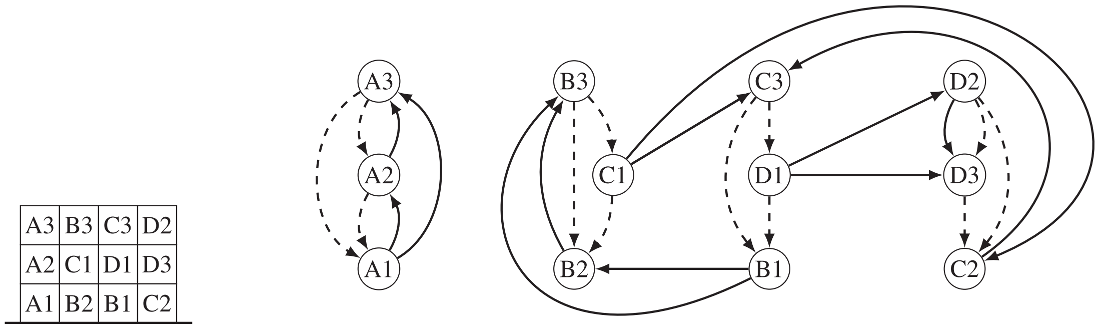

(a) Retrieve container D1
(b) Relocate container C2 to yard stack 2
(c) Retrieve containers C1 and C2
(d) Relocate container F3 to yard stack 1
(e) Relocate container B2 to yard stack 2
(f) Retrieve containers E1, E2, A1, B1, B2, B3, B4, and D2
(g) Relocate container A3 to yard stack 3
(h) Retrieve the remaining containers

Fig. 3. An optimal solution to the initial bay configuration in Fig. 2 (unrestricted).The total number of relocations is 4.

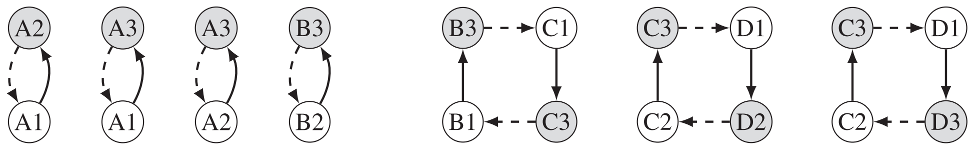
Table 1
Basic notation and definitions.

N:	Number of containers.
S^{Y}:	Number of stacks in the yard bay.
T^{Y}:	Height limit of yard stacks. The maximum number of containers that can be placed in each yard stack.
S^{V}:	Number of stacks in the vessel bay.
(s, t):	Slot in the tth tier of yard or vessel stack s.
$(s_c^{\mathrm{V}}, t_c^{\mathrm{V}})$:	Destination slot of container c in the vessel bay.
$\mathcal{C}$:	Yard bay and vessel bay configuration.
$c \in \mathcal{C}$:	Container c is in the yard bay in bay configuration $\mathcal{C}$.
$c \notin \mathcal{C}$:	Container c is in the vessel bay in bay configuration $\mathcal{C}$.
$(s_c^{\mathrm{Y}}(\mathcal{C}), t_c^{\mathrm{Y}}(\mathcal{C}))$:	Slot of container c $(\in \mathcal{C})$ in bay configuration $\mathcal{C}$.
$\mathcal{Y}_y(\mathcal{C})$:	Set of containers placed in yard stack y in bay configuration $\mathcal{C}$. $\mathcal{Y}_y(\mathcal{C}) := \{c \in \mathcal{C} \mid s_c^{\mathrm{Y}} = y\}$.
$\mathcal{V}_c(\mathcal{C})$:	Set of containers already placed in the destination vessel stack s_c^{V} of container c in bay configuration $\mathcal{C}$. $\mathcal{V}_c := \{e \notin \mathcal{C} \mid s_e^{\mathrm{V}} = s_c^{\mathrm{V}}\}$.
$\check{\mathcal{V}}_c(\mathcal{C})$:	Set of containers not yet placed in the destination vessel stack s_c^{V} of container c in bay configuration $\mathcal{C}$. $\check{\mathcal{V}}_c := \{e \in \mathcal{C} \mid s_e^{\mathrm{V}} = s_c^{\mathrm{V}}\}$.
$\mathcal{G}(\mathcal{C})$:	Target group in bay configuration $\mathcal{C}$. $\mathcal{G}(\mathcal{C}) := \{c \in \mathcal{C} \mid t_c^{\mathrm{V}} = \lvert\mathcal{V}_c(\mathcal{C})\rvert + 1\}$.
$\mathcal{T}(\mathcal{C})$:	Set of target stacks. $\mathcal{T}(\mathcal{C}) := \bigcup_{c \in \mathcal{G}(\mathcal{C})} \{s_c^{\mathrm{Y}}(\mathcal{C})\}$.
$g_y(\mathcal{C})$:	Topmost target container in target stack y $(\in \mathcal{T}(\mathcal{C}))$ in bay configuration $\mathcal{C}$. $g_y(\mathcal{C}) := \mathrm{argmax}_{c \in \mathcal{G}(\mathcal{C}) \cap \mathcal{Y}_y(\mathcal{C})} t_c^{\mathrm{Y}}(\mathcal{C})$.
$\mathcal{S}_y(\mathcal{C})$:	Stacking containers in yard stack y in bay configuration $\mathcal{C}$. $\mathcal{S}_y(\mathcal{C}) := \{c \in \mathcal{Y}_y(\mathcal{C}) \mid t_c^{\mathrm{Y}}(\mathcal{C}) > t_{g_y(\mathcal{C})}^{\mathrm{Y}}(\mathcal{C})\}$.
(c, s, d):	Relocation of container c from yard stack s to yard stack d.
$L_{csd}(\mathcal{C})$:	Relocation function. The bay configuration obtained by applying (c, s, d) to bay configuration $\mathcal{C}$.
$R(\mathcal{C})$:	Retrieval function. The bay configuration after all retrievable containers are retrieved in bay configuration $\mathcal{C}$.
$c \xrightarrow{\mathrm{y}} d$:	Yard precedence relation. Container c is above container d in the same yard stack.
$c \xrightarrow{\mathrm{v}} d$:	Vessel precedence relation. Container c should be placed below container d in the same vessel stack.
$\mathcal{M}^{2n}$:	Set of minimal $2n$-cycles.
$\mathcal{K}(a)$:	Set of $2n$-blocking containers in minimal $2n$-cycle a $(\in \mathcal{M}^{2n})$. $\mathcal{K}(a) := \{c \mid ((c, d) \in a) \wedge (c \xrightarrow{y} d)\}$. To disconnect a, one of the containers in $\mathcal{K}(a)$ has to be relocated.



(a) yard bay (b) precedence graph

Fig. 4. A bay configuration and the corresponding precedence graph (dashed arrow: yard precedence, solid arrow: vessel precedence).



(a) minimal 2-cycles (b) minimal 4-cycles

Fig. 5. Minimal cycles in the precedence graph in Fig. 4 (dark: $2n$-blocking, white: $2n$-blocked).

that G does not contain minimal cycles of an odd length. Furthermore, yard precedence and vessel precedence appear in turn along every minimal cycle. A precedence graph is illustrated in Fig. 4. It contains four minimal 2-cycles and three minimal 4-cycles, which are summarized in Fig. 5.

To disconnect a cycle, we should relocate a container in the cycle. Since yard precedence $c \xrightarrow{\mathrm{y}} d$ implies that container d cannot be relocated before container c, there are n candidate containers in a minimal $2n$-cycle (recall that half of the arcs in a minimal cycle are yard precedence). In Fig. 5, dark nodes represent candidate containers to be relocated. To distinguish them from the other n containers, we refer to the former as $2n$-blocking and the latter as $2n$-blocked. Please note that a container can be $2n$-blocking in more than one minimal $2n$-cycle. It is also the case that a container is both $2n$-blocking and $2n'$-blocking for $n' \neq n$. In Fig. 5, container A3 is 2-blocking in two minimal 2-cycles, and container C3 is 4-blocking in all the three minimal 4-cycles. Moreover, container B3 is both 2-blocking and 4-blocking. Hereafter, the set of all minimal $2n$-cycles is denoted by $\mathcal{M}^{2n}$. The set of $2n$-blocking containers in a cycle $a \in \mathcal{M}^{2n}$ is denoted by $\mathcal{K}(a)$.

5. Lower bounds

In this section, we propose three lower bounds on the total number of relocations for the unrestricted BRPSP. Since the restricted BRPSP imposes an additional constraint on relocatable containers, the optimal value of the restricted BRPSP is not smaller than that of the unrestricted BRPSP. It means that these lower bounds are valid also for the restricted BRPSP.

5.1. Cycle and blocking-based lower bounds

5.1.1. LB2c

This simple lower bound counts the total number of 2-blocking containers. To disconnect a minimal 2-cycle, we must relocate the 2-blocking container in it, as we have already seen in Section 4.2. Noting the fact that we must relocate all 2-blocking containers to disconnect all minimal 2-cycles, we can use the total number of 2-blocking containers as a lower bound on the total number of relocations. This lower bound is denoted by LB2c. Let us define the set of all 2-blocking containers by

$$\mathcal{B}^2 := \bigcup_{a\in\mathcal{M}^2} \mathcal{K}(a). \tag{5}$$

Then, we can write LB2c as LB2c $:= |\mathcal{B}^2|$. For the bay configuration in Fig. 4(a), containers A2, A3 and B3 are 2-blocking from Fig. 5(a), and relocating them disconnects all the four minimal 2-cycles. Hence, we obtain LB2c $= 3$.

5.1.2. LB2c4c

Although LB2c considers only minimal 2-cycles, we must disconnect all minimal 4-cycles as well to retrieve containers. Since some of the minimal 4-cycles may be disconnected when minimal 2-cycles are disconnected, we define $\widetilde{\mathcal{M}}^4$, the set of minimal 4-cycles that are independent of minimal 2-cycles as follows:

$$\widetilde{\mathcal{M}}^4 := \{a \in \mathcal{M}^4 \mid \mathcal{K}(a) \cap \mathcal{B}^2 = \emptyset\}. \tag{6}$$

We can see from this definition that if $a \in \mathcal{M}^4 \setminus \widetilde{\mathcal{M}}^4$, at least one of the two 4-blocking containers in cycle a is 2-blocking. Please note that in a cycle $a \in \widetilde{\mathcal{M}}^4$, a 4-*blocking* container is never 2-blocking, but a 4-*blocked* container can be 2-blocking. In Fig. 5(b), the second and third cycles belong to $\widetilde{\mathcal{M}}^4$, whereas the first one does not because container B3 is 2-blocking as well as 4-blocking. Let us denote the set of 4-blocking containers by $\mathcal{B}^4$, which is defined as

$$\mathcal{B}^4 := \bigcup_{a\in\widetilde{\mathcal{M}}^4} \mathcal{K}(a). \tag{7}$$

Next, we try to disconnect minimal 4-cycles in $\widetilde{\mathcal{M}}^4$. To obtain a lower bound on the total number of relocations, we should solve the problem of minimizing the number of 4-blocking containers in $\mathcal{B}^4$ that disconnect all minimal 4-cycles in $\widetilde{\mathcal{M}}^4$. This problem can be modeled as a minimum set cover problem: $\widetilde{\mathcal{M}}^4$ is covered by a minimum number of subsets $\widetilde{\mathcal{M}}^4_c \subset \widetilde{\mathcal{M}}^4$ $(c \in \mathcal{B}^4)$, where

$$\widetilde{\mathcal{M}}^4_c := \{a \in \widetilde{\mathcal{M}}^4 \mid c \in \mathcal{K}(a)\}. \tag{8}$$

Specifically, $\widetilde{\mathcal{M}}^4_c$ is the set of minimal 4-cycles that are disconnected by relocating 4-blocking container c. Unfortunately, the minimum set cover problem is in general NP-hard (Garey & Johnson, 1979). Therefore, we use a simple lower bound on its objective value that can be computed in polynomial time. Let $m_1, m_2, \ldots, m_{|\mathcal{B}^4|}$ be the sequence of $|\widetilde{\mathcal{M}}^4_c|$ $(c \in \mathcal{B}^4)$, sorted in nonincreasing order. Then, it is easy to check that k^* satisfying

$$\sum_{k=1}^{k^*-1} m_k < |\widetilde{\mathcal{M}}^4| \le \sum_{k=1}^{k^*} m_k \tag{9}$$

gives a lower bound on the number of 4-blocking containers necessary for disconnecting all minimal 4-cycles in $\widetilde{\mathcal{M}}^4$. In summary, the lower bound LB2c4c is given by LB2c4c $:=$ LB2c $+ k^*$. For the bay configuration in Fig. 4(a), $\mathcal{B}^4$ is composed of containers C3, D2, and D3. Among them, container C3 disconnects both the minimal 4-cycles in $\widetilde{\mathcal{M}}^4$, so that $m_1 = 2$ (container C3) and $m_2 = m_3 = 1$ (containers D2 and D3). Hence, $k^* = 1$ and LB2c4c $= 3 + 1 = 4$ in this case.

5.2. Relaxation-based lower bound LBr

It is true that LB2c and LB2c4c are not difficult to compute, but they ignore longer cycles in the precedence graph. As a result, the optimality gap tends to be large, which causes a negative impact on the search performance of exact algorithms. In this section, we propose another lower bound LBr that considerably reduces the gap at the cost of computation time.

To obtain LBr, we introduce a relaxation of the original unrestricted BRPSP where infinitely many empty yard stacks are available. In this relaxation, the destination of a relocation is always an empty yard stack. It follows that each container is relocated at most once. Since the complexity of this relaxation is not trivial, we apply a depth-first branch-and-bound algorithm to solve it. However, it is too time-consuming to enumerate all feasible relocations one by one. To reduce the search space and solve the relaxation as fast as possible, we further investigate the structure of its optimal solution.

First, we show an interesting property that the restricted and unrestricted variants of the relaxation are equivalent, although the relaxation is derived from the unrestricted BRPSP. Therefore, we only need to search such solutions that the container to be retrieved next is at first determined among the target containers, and that only stacking containers above it are relocated until it is retrieved. The following lemma states this property.

Lemma 1. *There exists an optimal solution of the relaxation where between two successive retrievals of target containers c and d, only stacking containers above container d (if any) are relocated.*

Proof. Suppose an optimal solution that does not satisfy the condition of the lemma, and let container e be the first container that breaks the condition. That is, container d is placed in a yard stack y and container e is relocated from another yard stack $z \neq y$ between the retrievals of containers c and d. Since no other container is retrieved between these two retrievals, container e does not block any retrieval from yard stack z even if it is not relocated. Hence, we can consider a new solution where the relocation of container e is delayed until container d is retrieved. This solution satisfies the condition of the lemma at least until container d is retrieved. Repeatedly delaying the relocation of a container that breaks the condition first, we reach an optimal solution satisfying the condition, which completes the proof. □

This lemma suggests that we only need to determine one by one the yard stack from which we retrieve a container next. Therefore, we can represent a solution of the relaxation by a sequence of target yard stacks. In the branch-and-bound algorithm for the relaxation, an optimal sequence of yard stacks is searched for from first to last. We use LB2c4c as a lower bound on LBr in the algorithm. To further improve its search efficiency, we apply a dominance rule.

Suppose a sequence $s_1, s_2, \ldots, s_n$ of yard stacks representing a partial solution of the relaxation for a bay configuration $\mathcal{C}$. Here, we assume without loss of generality that $\mathcal{C} = R(\mathcal{C})$: All retrievable containers have been retrieved from $\mathcal{C}$. Let us denote a dummy empty yard stack by $\bullet$. Accordingly, $(c, s, \bullet)$ denotes the relocation of container c from yard stack s to a dummy empty yard stack, and $L_{cs\bullet}(\mathcal{C})$ the bay configuration obtained by applying $(c, s, \bullet)$ to $\mathcal{C}$. Let $\widehat{\mathcal{C}}_0 := \mathcal{C}$ and define $\widehat{\mathcal{C}}_i$ $(1 \le i \le n)$ by

$$\widehat{\mathcal{C}}_i := R \circ L_{c_{ik_i}s_i\bullet} \circ L_{c_{i,k_i-1}s_i\bullet} \cdots \circ L_{c_{i1}s_i\bullet}(\widehat{\mathcal{C}}_{i-1}), \tag{10}$$

where

$$\mathcal{S}_{s_i}(\widehat{\mathcal{C}}_{i-1}) =: \{c_{i1}, c_{i2}, \ldots, c_{ik_i}\}, \tag{11}$$

$$t^{Y}_{c_{i1}}(\widehat{\mathcal{C}}_{i-1}) > t^{Y}_{c_{i2}}(\widehat{\mathcal{C}}_{i-1}) > \cdots > t^{Y}_{c_{ik_i}}(\widehat{\mathcal{C}}_{i-1}). \quad (12)$$

Specifically, $\widehat{\mathcal{C}}_i$ is the bay configuration after the retrieval of the target container below stacking containers $c_{i1}, c_{i2}, \ldots, c_{ik_i}$ in yard stack s_i and subsequent retrievals that do not require relocations. By using them, the dominance rule is summarized as the following lemma:

Lemma 2. *We need not consider the partial solution represented by $s_1, \ldots, s_n$ for $\mathcal{C}$, if the following conditions are both satisfied:*

1. $\mathcal{S}_{s_n}(\widehat{\mathcal{C}}_{n-1}) = \mathcal{S}_{s_n}(\mathcal{C})$.
2. $s_1 > s_n$.

Proof. Let X and Y be the partial solutions represented by $s_1, \ldots, s_n$ and $s_n, s_1, \ldots, s_{n-1}$, respectively. We show that if X is optimal, Y is also optimal, when Conditions 1 and 2 are both satisfied. From Condition 1, the number of relocations from yard stack s_n in Y is equal to that in X. Let us define $\widehat{\mathcal{C}}'_1, \widehat{\mathcal{C}}'_2, \ldots, \widehat{\mathcal{C}}'_n$ for Y in a similar manner to $\widehat{\mathcal{C}}_1, \widehat{\mathcal{C}}_2, \ldots, \widehat{\mathcal{C}}_n$ for X, respectively. That is, $\widehat{\mathcal{C}}'_1$ is the bay configuration after the retrieval from yard stack s_n, and $\widehat{\mathcal{C}}'_i$ $(2 \le i \le n)$ after the retrieval from yard stack s_{i-1}. Then, it is easy to see that $\mathcal{S}_{s_i}(\widehat{\mathcal{C}}'_i) \subseteq \mathcal{S}_{s_i}(\widehat{\mathcal{C}}_{i-1})$ holds for $1 \le i \le n-1$. Furthermore, $c \in \widehat{\mathcal{C}}'_n$ implies $c \in \widehat{\mathcal{C}}_n$. From the optimality of X, we obtain $\mathcal{S}_{s_i}(\widehat{\mathcal{C}}'_i) = \mathcal{S}_{s_i}(\widehat{\mathcal{C}}_{i-1})$ $(1 \le i \le n-1)$ and $\widehat{\mathcal{C}}'_n = \widehat{\mathcal{C}}_n$ because otherwise, the number of relocations in Y is smaller than that in X, which contradicts the optimality of X. This proves the optimality of Y. Condition 2 is for breaking ties. □

This lemma is utilized to restrict the search space in the branch-and-bound algorithm for the relaxation. Suppose that at some node of depth $n-1$ in the search tree, we have already retrieved containers from yard stacks $s_1, \ldots, s_{n-1}$ in this order. Then, we need not consider yard stack $y \in \mathcal{T}(\widehat{\mathcal{C}}_{n-1})$ for the next retrieval at depth n, if there exists i $(1 \le i \le n-1)$ satisfying both $s_i > y$ and $\mathcal{S}_y(\widehat{\mathcal{C}}_{n-1}) = \mathcal{S}_y(\widehat{\mathcal{C}}_{i-1})$. It is because Lemma 2 is satisfied if $\widehat{\mathcal{C}}_{i-1}$ is regarded as the initial bay configuration $\mathcal{C}$ in the lemma.

5.3. Relations to a lower bound for the relocation problem

For the relocation problem, several lower bounds on the total number of relocations have been proposed. Among them, the simplest and most intuitive is the one proposed by Kim and Hong (2006). In the relocation problem, a container is called a blocking container iff another container that should be retrieved earlier is placed below it. Since it is obvious that each blocking container is relocated at least once, they use the total number of blocking containers as a lower bound on the total number of relocations, which we refer to as LB-KH. From this point of view, a direct extension of LB-KH to the BRPSP would be LB2c that counts the total number of 2-blocking containers. To improve LB2c, we take into account 4-blocking containers as well in LB2c4c. We may further improve LB2c4c by considering minimal 6-cycles, 8-cycles, and so on; however, it is difficult to develop an efficient algorithm for the lower bound in this direction. To enable this, we need to interpret LB-KH from a different viewpoint. In LB-KH, a blocking container is assumed to be relocated exactly once and never twice or more. It is equivalent to consider infinitely many empty stacks and to relocate each of the blocking containers to an empty stack. This is the same idea as the relaxation introduced for LBr. In this sense, LBr is another extension of LB-KH to our problem. Since each container is relocated at most once also in LB2c and LB2c4c, we can relate them with the relaxation. Noting that LBr solves the relaxation to optimality, we see that LB2c and LB2c4c only yield its lower bounds. From these observations, it is obvious that LB2c ≤ LB2c4c ≤ LBr holds.

6. Exact algorithm

In this section, we develop an exact algorithm for the BRPSP. We adopt a branch-and-bound algorithm with iterative deepening as the exact approach, because it has already been proved to be effective for similar problems such as the relocation problem (Tanaka & Mizuno, 2018; Tanaka & Takii, 2016; Zhu et al., 2012) and the pre-marshalling problem (Tanaka & Tierney, 2018; Tierney et al., 2017). The framework is the same as those in Tanaka and Takii (2016), Tanaka and Tierney (2018), Tanaka and Mizuno (2018), so that we only present its overview here.

6.1. Branch-and-bound algorithm with iterative deepening

The branch-and-bound algorithm searches for an optimal sequence of relocations in the depth-first manner. Thus, a node at depth k represents a partial sequence of feasible relocations of length k. It also represents the bay configuration obtained by applying the relocations and retrieving as many containers as possible. We repeatedly employ this branch-and-bound algorithm with the maximum search depth increased one by one. First, it is set to the initial lower bound and the branch-and-bound algorithm is applied, in order to search for an optimal solution whose objective value is equal to the initial lower bound. The algorithm is terminated if such a solution is found. Otherwise, we apply the branch-and-bound algorithm again with the maximum search depth increased by one. It is repeated until an optimal solution is found or the time limit is reached.

6.2. Dominance rules

We utilize dominance rules to reduce the search space in the branch-and-bound algorithm. They are derived from those for the relocation problem (Tanaka & Mizuno, 2018), except for the last one.

We first provide dominance rules for the unrestricted BRPSP. They enable us to exclude a partial sequence of relocations $(c_1, s_1, d_1), (c_2, s_2, d_2), \ldots, (c_n, s_n, d_n)$ when searching for an optimal sequence for an initial bay configuration $\mathcal{C}$. Roughly speaking, Transitive Relocation Rules A and B combine two relocations (c_1, s_1, d_1) and (c_1, d_1, d_n) into one relocation (c_1, s_1, d_n), Transitive Relocation Rule C replaces (c_1, s_1, d_1) and (c_1, d_1, d_n) with (c_1, s_1, d'_1) and (c_1, d'_1, d_n), respectively. The Independent Relocation Rule interchanges the order of two independent relocations (c_1, s_1, d_1) and (c_n, s_n, d_n). Retrieval Rule A eliminates (c_1, s_1, d_1) if container c_1 can be retrieved without the relocation. Retrieval Rule B replaces (c_1, s_1, d_1) with (c_1, s_1, d'_1) if container c_1 can be retrieved anyway. In the following lemmas, we assume without loss of generality that $\mathcal{C} = R(\mathcal{C})$ holds, and define $\mathcal{C}_j$ $(0 \le j \le n)$ by $\mathcal{C}_0 := \mathcal{C}$ and $\mathcal{C}_j := R \circ L_{c_j s_j d_j}(\mathcal{C}_{j-1})$.

Lemma 3. *(Transitive Relocation Rule A) We need not consider the sequence $(c_1, s_1, d_1), (c_2, s_2, d_2), \ldots, (c_n, s_n, d_n)$ for $\mathcal{C}$, if the following conditions are all satisfied:*

1. $s_n = d_1,\ c_n = c_1$,
2. $c_1 \notin \{c_2, \ldots, c_{n-1}\}$,
3. $s_1 \notin \{s_2, d_2, \ldots, s_{n-1}, d_{n-1}\}$,
4. $|\mathcal{Y}_{s_1}(\mathcal{C}_{n-1})| = |\mathcal{Y}_{s_1}(\mathcal{C})| - 1$.

Lemma 4. *(Transitive Relocation Rule B) We need not consider the sequence $(c_1, s_1, d_1), (c_2, s_2, d_2), \ldots, (c_n, s_n, d_n)$ for $\mathcal{C}$, if the following conditions are all satisfied:*

1. $s_n = d_1,\ c_n = c_1$,
2. $c_1 \notin \{c_2, \ldots, c_{n-1}\}$,
3. $d_n \notin \{s_1, d_1, \ldots, s_{n-1}, d_{n-1}\}$,
4. $|\mathcal{Y}_{d_n}(\mathcal{C}_{n-1})| = |\mathcal{Y}_{d_n}(\mathcal{C})|$.

Lemma 5. *(Transitive Relocation Rule C) We need not consider the sequence* $(c_1, s_1, d_1), (c_2, s_2, d_2), \dots, (c_n, s_n, d_n)$ *if the following conditions are all satisfied for some* d'_1*:*

1. $s_n = d_1,\ c_n = c_1$,
2. $c_1 \notin \{c_2, \dots, c_{n-1}\}$,
3. $d'_1 \notin \{s_1, d_1, s_2, d_2, \dots, s_n, d_n\}$,
4. $|\mathcal{Y}_{d'_1}(\mathcal{C}_{n-1})| = |\mathcal{Y}_{d'_1}(\mathcal{C})|$.
5. $(|\mathcal{Y}_{d'_1}(\mathcal{C}_{n-1})| < |\mathcal{Y}_{d_1}(\mathcal{C}_{n-1})| - 1) \vee ((|\mathcal{Y}_{d'_1}(\mathcal{C}_{n-1})| = |\mathcal{Y}_{d_1}(\mathcal{C}_{n-1})| - 1) \wedge (d'_1 < d_1))$.

Lemma 6. *(Independent Relocation Rule) We need not consider the sequence* $(c_1, s_1, d_1), (c_2, s_2, d_2), \dots, (c_n, s_n, d_n)$ *for* $\mathcal{C}$*, if the following conditions are all satisfied:*

1. $\{s_1, d_1\} \cap \{s_2, d_2, \dots, s_n, d_n\} = \emptyset$,
2. $|\mathcal{Y}_{s_1}(\mathcal{C}_{n-1})| = |\mathcal{Y}_{s_1}(\mathcal{C})| - 1$,
3. $s_1 > s_n$.

Lemma 7. *(Retrieval Rule A) We need not consider the sequence* $(c_1, s_1, d_1), (c_2, s_2, d_2), \dots, (c_n, s_n, d_n)$ *for* $\mathcal{C}$*, if the following conditions are all satisfied:*

1. $c_1 \notin \{c_2, \dots, c_{n-1}, c_n\}$,
2. $(c_1 \in \mathcal{C}_{n-1}) \wedge (c_1 \notin \mathcal{C}_n)$,
3. $s_1 \notin \{s_2, d_2, \dots, s_n, d_n\}$,
4. $|\mathcal{Y}_{s_1}(\overline{\mathcal{C}}_{n-1})| = |\mathcal{Y}_{s_1}(\mathcal{C})| - 1$.

Here, $\overline{\mathcal{C}}_{n-1}$ *is the bay configuration just before container* c_1 *is retrieved (between* $\mathcal{C}_{n-1}$ *and* $\mathcal{C}_n$ *from Condition 2).*

Lemma 8. *(Retrieval Rule B) We need not consider the sequence* $(c_1, s_1, d_1), (c_2, s_2, d_2), \dots, (c_n, s_n, d_n)$ *if the following conditions are all satisfied for some* d'_1*:*

1. $c_1 \notin \{c_2, \dots, c_{n-1}, c_n\}$,
2. $(c_1 \in \mathcal{C}_{n-1}) \wedge (c_1 \notin \mathcal{C}_n)$,
3. $d'_1 \notin \{s_1, d_1, s_2, d_2, \dots, s_n, d_n\}$,
4. $|\mathcal{Y}_{d'_1}(\overline{\mathcal{C}}_{n-1})| = |\mathcal{Y}_{d'_1}(\mathcal{C})|$,
5. $(|\mathcal{Y}_{d'_1}(\overline{\mathcal{C}}_{n-1})| < |\mathcal{Y}_{d_1}(\mathcal{C}_{n-1})| - 1) \vee ((|\mathcal{Y}_{d'_1}(\overline{\mathcal{C}}_{n-1})| = |\mathcal{Y}_{d_1}(\mathcal{C}_{n-1})| - 1) \wedge (d'_1 < d_1))$.

Brief proofs of these lemmas are presented in Appendix A. The primary difference from the dominance rules in Tanaka and Mizuno (2018) is how to break ties. In Tanaka and Mizuno (2018), ties are broken by the stack priority. The priority of a stack in the relocation problem is defined by the smallest priority value of containers placed in the stack. In our problem, however, we cannot define it in a similar manner. Therefore, ties are broken simply by the yard stack number in Condition 3 of Lemma 6. We also use the yard stack height in Condition 5 of Lemmas 5 and 8, and ties are broken first by the yard stack height, and next by the yard stack number.

Condition 4 in Lemma 7 becomes a little complicated, compared with those in the corresponding dominance rule for the relocation problem. It is because container c_1 should not block another retrieval from yard stack s_1 even if it is not relocated from there, as such retrievals may enable retrievals of other containers including container c_1 itself in the original sequence. To this end, Condition 4 is introduced so that no container is retrieved from yard stack s_1 prior to container c_1 regardless of whether it is relocated or not. Similarly, Condition 4 in Lemma 8 ensures that no container is retrieved from yard stack d'_1 prior to container c_1.

As is the case with the restricted relocation problem, only Lemmas 4 and 8 are applicable to the restricted BRPSP due to the constraint on relocatable containers. On the other hand, we can derive a new dominance rule for the restricted BRPSP, which is similar to Lemma 2. Suppose a partial sequence of feasible relocations $(c_{11}, s_1, d_{11}), \dots, (c_{1k_1}, s_1, d_{1k_1}), (c_{21}, s_2, d_{21}), \dots, (c_{2k_2}, s_2, d_{2k_2}), \dots, (c_{n1}, s_n, d_{n1}), \dots, (c_{nk_n}, s_n, d_{nk_n})$, where $k_1, k_2, \dots, k_n$ stacking containers are relocated from target stacks $s_1, s_2, \dots, s_n$, respectively. Let us define $\mathcal{C}_{ij}$ $(1 \le i \le n,\ 0 \le j \le k_i)$ by

$$\mathcal{C}_{ij} := \begin{cases} \mathcal{C}_{i-1,k_{i-1}} & j = 0, \\ L_{c_{ij}s_id_{ij}}(\mathcal{C}_{i,j-1}), & 1 \le j \le k_i - 1, \\ R \circ L_{c_{ij}s_id_{ij}}(\mathcal{C}_{i,j-1}), & j = k_i, \end{cases} \tag{13}$$

where $\mathcal{C}_{0k_0} := \mathcal{C}$. For this sequence of relocations, the following lemma holds.

Lemma 9. *(Target Stack Rule) We need not consider the sequence* $(c_{11}, s_1, d_{11}), \dots, (c_{1k_1}, s_1, d_{1k_1}), (c_{21}, s_2, d_{21}), \dots, (c_{2k_2}, s_2, d_{2k_2}), \dots, (c_{n1}, s_n, d_{n1}), \dots, (c_{nk_n}, s_n, d_{nk_n})$ *for* $\mathcal{C}$*, if the following conditions are all satisfied:*

1. $\mathcal{S}_{s_n}(\mathcal{C}_{n0}) = \mathcal{S}_{s_n}(\mathcal{C})$,
2. $\{d_{n1}, \dots, d_{nk_n}\} \cap \{d_{11}, \dots, d_{1k_1}, \dots, d_{n-1,1}, \dots, d_{n-1,k_{n-1}}\} = \emptyset$.
3. $\mathcal{Y}_{d_{nj}}(\mathcal{C}_{n0}) = \mathcal{Y}_{d_{nj}}(\mathcal{C})$ *for all* j $(1 \le j \le k_n)$.
4. $s_1 > s_n$

Proof. Let us denote by X and Y the original sequence and a new sequence $(c_{n1}, s_n, d_{n1}), \dots, (c_{nk_n}, s_n, d_{nk_n}), (c_{11}, s_1, d_{11}), \dots, (c_{1k_1}, s_1, d_{1k_1}), (c_{n-1,1}, s_{n-1}, d_{n-1,1}), \dots, (c_{n-1,k_{n-1}}, s_{n-1}, d_{n-1,k_{n-1}})$, respectively. We show that if X is optimal, then Y is also optimal when Conditions 1–4 are all satisfied. Clearly, Conditions 1 and 4 correspond to Conditions 1 and 2 in Lemma 2, respectively. From Conditions 2 and 3, the destination stacks $d_{n1}, \dots, d_{nk_n}$ of stacking containers in yard stack s_n do not change before a container is retrieved from yard stack s_{n-1} in X. It follows that retrieving a container from yard stack s_n before retrieving a container from yard stack s_1 does not block any retrievals and relocations from (or to) yard stacks $d_{n1}, \dots, d_{nk_n}$. Therefore, the optimality of X implies the optimality of Y as in Lemma 2. □

6.3. Upper bound heuristic

An upper bound is computed at promising nodes in the course of the branch-and-bound algorithm using a simple greedy heuristic. It is an improved version of MBW4CB+MM4CB proposed in Jovanovic et al. (2018) for the restricted BRPSP. Therefore, it yields a valid upper bound also for the unrestricted BRPSP. The heuristic is composed of the following two decisions:

(A) Determine the yard stack among the target stacks from which the topmost target container is retrieved next.
(B) Determine the destination of each stacking container in that yard stack.

To determine the source yard stack s from which the topmost target container g_s is retrieved next, we evaluate several objectives in lexicographic order. We choose the yard stack that optimizes the first objective function. If there is more than one candidate, we choose the one that optimizes the second objective function, and so on. Let us denote the topmost container in yard stack y by p_y. Then, the objectives used for each target yard stack $y \in \mathcal{T}$ are described as follows:

1. Minimize $|\mathcal{S}_y \setminus (\mathcal{B}^2 \cup \mathcal{B}^4)|$, the number of stacking containers that are neither 2-blocking nor 4-blocking.
2. Minimize $|\mathcal{S}_y \cap \mathcal{B}^4|$, the number of 4-blocking stacking containers.
3. Minimize $|\mathcal{S}_y \cap \mathcal{B}^2|$, the number of 2-blocking stacking containers.
4. Maximize $\min_{e \in \mathcal{S}_y}(t_e^{\mathrm{V}} - |\mathcal{V}_e|)$, the minimum number of containers that should be retrieved before retrieving each stacking container e.
5. Maximize $|\check{\mathcal{V}}_{g_y}|$, the number of containers that are not yet relocated to the vessel stack of the topmost target container g_y.

Table 2
Comparison of upper bound heuristics for initial bay configurations (**bold**: best results).

N	S^V	S^Y	T^Y	greedy heuristics in Jovanovic et al. (2018)					proposed
				LT+MBW	MM+MBW	MM4CB+MBW	MM+MBW4CB	MM4CB+MBW4CB	
10	3	3	6	2.350	**2.175**	2.225	**2.175**	2.225	2.225
16	3	4	6	6.100	5.600	5.675	5.525	5.600	**5.450**
16	3	3	8	8.975	8.200	**8.150**	8.200	8.250	8.425
31	3	8	6	13.275	12.350	12.175	11.925	11.575	**11.275**
31	3	6	8	20.075	16.800	17.425	17.000	17.250	**16.175**
46	3	12	6	23.000	21.700	19.100	20.525	18.775	**18.175**
46	3	9	8	33.625	28.350	28.925	27.900	28.625	**27.175**
19	5	5	6	3.925	3.525	3.675	3.500	3.500	**3.350**
19	5	4	8	5.875	5.375	5.400	5.075	5.100	**4.950**
29	5	8	6	7.775	7.825	6.850	7.075	6.675	**6.475**
29	5	6	8	11.975	10.275	9.950	**9.400**	9.675	**9.400**
54	5	14	6	21.125	20.200	18.325	18.225	16.875	**16.400**
54	5	11	8	28.225	24.975	23.600	23.300	22.375	**20.850**
79	5	20	6	36.950	34.975	29.400	31.875	28.125	**27.300**
79	5	15	8	53.950	46.000	41.450	44.125	40.300	**38.800**
50	10	13	6	11.275	11.800	10.675	10.150	9.550	**9.150**
50	10	10	8	14.050	13.650	12.175	12.050	11.400	**11.250**
70	10	18	6	21.125	20.350	18.600	17.325	16.725	**15.875**
70	10	14	8	26.825	24.300	22.800	21.850	20.825	**20.050**
120	10	30	6	45.900	47.150	40.950	42.200	39.050	**36.075**
120	10	23	8	66.350	62.225	52.050	58.275	51.175	**48.625**
170	10	43	6	72.225	75.850	62.725	71.600	61.075	**56.550**
170	10	32	8	107.250	100.125	84.425	93.900	79.675	**79.500**
94	15	24	6	23.875	23.625	20.900	20.750	19.175	**18.550**
94	15	18	8	34.375	31.800	29.050	29.075	26.925	**24.875**
124	15	31	6	37.950	38.025	35.200	35.800	33.150	**29.850**
124	15	24	8	53.700	51.350	43.850	45.675	41.775	**40.100**
199	15	50	6	76.050	80.575	67.900	72.350	66.825	**61.550**
199	15	38	8	106.250	105.100	87.300	95.725	84.525	**79.450**
274	15	69	6	118.375	124.275	107.750	116.775	104.475	**92.450**
274	15	52	8	169.025	165.850	136.775	152.025	132.400	**125.725**
150	20	38	6	43.400	44.850	39.700	40.175	39.050	**35.375**
150	20	29	8	61.025	58.850	51.475	52.500	48.300	**44.250**
190	20	48	6	62.675	65.250	57.150	58.800	57.350	**49.075**
190	20	36	8	89.175	84.550	74.075	79.600	72.125	**65.450**
290	20	73	6	112.250	122.875	106.825	115.825	102.500	**91.950**
290	20	55	8	164.375	159.600	136.175	149.700	131.775	**122.225**
390	20	98	6	174.575	194.475	164.125	184.875	164.075	**141.500**
390	20	74	8	247.125	246.975	203.475	234.200	199.325	**185.275**

6. Minimize $\min_{e \in (\mathcal{Y}_y \setminus \mathcal{S}_y)\setminus\{g_y\}} (t_e^V - |\mathcal{V}_e|)$, the minimum number of containers that should be retrieved before retrieving each container e placed below the topmost target container g_y.

The difference from the retrieval heuristic MBW4CB is that three objectives 4–6 are introduced in our heuristic. They contribute to breaking ties that often occur only with the first three objectives. The fourth objective is to avoid relocating stacking containers that will become available for retrieval in the near future. Note that $(t_e^V - |\mathcal{V}_e|)$ is the number of containers that should be placed below container e in its vessel stack s_e^V. The fifth objective is to retrieve as soon as possible a target container that blocks many containers. It maximizes the number of containers that should be placed above target container g_y in its vessel stack. The last objective is for evaluating containers placed below target container g_y in the yard stack. A smaller value of $(t_e^V - |\mathcal{V}_e|)$ implies that container e is blocked by fewer containers. Therefore, it is expected to become retrievable soon after retrieving container g_y.

After source yard stack s is determined, we next determine the destination yard stack d of each stacking container $c \in \mathcal{S}_s$ by the following procedure:

(i) If there exists at least one yard stack y where (a) container c does not become 2-blocking or 4-blocking, and (b) its topmost container p_y satisfies $c \overset{V}{\to} p_y$, the yard stack that minimizes $t_{p_y}^V$ is chosen among such yard stacks.

(ii) Otherwise, the following objectives are evaluated in lexicographic order for each yard stack $y \neq s$ satisfying $|\mathcal{Y}_y| < T$:

1. Avoid making container c 2-blocking after relocation.
2. Avoid making container c 4-blocking after relocation.
3. Maximize $\min_{e \in \mathcal{Y}_y} (t_e^V - |\mathcal{V}_e|)$, the minimum number of containers that should be retrieved before retrieving each container e in the yard stack.

The relocation heuristic MM4CB is exactly the same as (ii), and the difference is (i). Even if container c does not become 2-blocking or 4-blocking when relocated to a yard stack y, it may cause a new blocking there. However, it at least does not block topmost container p_y due to Condition (b). In this case, such a yard stack is preferred for keeping the other yard stacks as potential destinations for future relocations. If we focus only on vessel stack s_c^V $(= s_{p_y}^V)$, t_c^V and $t_{p_y}^V$ are interpreted as the priorities of containers c and p_y, respectively, where a smaller value means a higher priority. Since the priority of yard stack y decreases from $t_{p_y}^V$ to t_c^V by placing container c (note that $t_c^V < t_{p_y}^V$ holds from $c \overset{V}{\to} p_y$), the yard stack that minimizes the decrease is chosen among those satisfying Conditions (a) and (b). This can avoid using another yard stack with a larger priority that has a higher potential for accepting a relocation.

6.4. Implementation of lower bounds in the branch-and-bound algorithm

A lower bound is computed at every node in the search tree of the branch-and-bound algorithm. To this end, we need to

Table 3
Comparison of lower bounds for initial bay configurations (*italic*: average over instances for which LBr is obtained).

N	S^V	S^Y	T^Y	LB2c	LB2c4c	LBr			
						solved	LB	CPU time [s]	
								ave	max
10	3	3	6	1.600	1.825	40	1.900	0.00	0.00
16	3	4	6	3.500	4.350	40	4.400	0.00	0.00
16	3	3	8	4.325	5.375	40	5.400	0.00	0.00
31	3	8	6	6.125	8.525	40	9.000	0.00	0.00
31	3	6	8	8.450	10.825	40	11.175	0.00	0.00
46	3	12	6	9.500	13.625	40	14.550	0.00	0.00
46	3	9	8	12.800	16.925	40	18.050	0.00	0.00
19	5	5	6	2.225	2.900	40	3.050	0.00	0.00
19	5	4	8	2.950	4.025	40	4.150	0.00	0.00
29	5	8	6	3.450	5.200	40	5.500	0.00	0.00
29	5	6	8	4.425	6.525	40	7.050	0.00	0.00
54	5	14	6	6.725	11.400	40	13.150	0.00	0.00
54	5	11	8	9.200	14.050	40	16.025	0.00	0.00
79	5	20	6	9.950	17.450	40	21.250	0.00	0.00
79	5	15	8	14.200	22.550	40	27.175	0.00	0.00
50	10	13	6	3.875	6.200	40	7.425	0.00	0.00
50	10	10	8	4.000	7.300	40	8.525	0.00	0.00
70	10	18	6	4.525	9.275	40	11.675	0.00	0.00
70	10	14	8	6.050	11.725	40	14.625	0.00	0.00
120	10	30	6	8.450	18.325	40	25.450	0.05	0.20
120	10	23	8	12.300	24.050	40	33.150	0.06	0.35
170	10	43	6	11.150	27.075	40	39.325	4.13	36.33
170	10	32	8	15.650	34.225	40	50.725	5.82	30.34
94	15	24	6	4.525	9.000	40	12.500	0.00	0.06
94	15	18	8	6.650	12.650	40	17.200	0.00	0.03
124	15	31	6	5.725	13.125	40	19.775	0.15	0.80
124	15	24	8	7.800	17.800	40	26.275	0.34	4.40
199	15	50	6	9.375	23.950	40	38.600	211.95	1312.81
199	15	38	8	12.425	31.400	39	*49.051*	364.78	1800.00
274	15	69	6	13.200	35.575	0	–	1800.00	1800.00
274	15	52	8	17.950	47.575	0	–	1800.00	1800.00
150	20	38	6	5.825	13.700	40	21.350	4.42	37.08
150	20	29	8	7.650	17.475	40	27.975	7.50	73.82
190	20	48	6	6.700	18.100	39	*29.923*	255.26	1800.00
190	20	36	8	9.425	24.675	38	*40.368*	464.65	1800.00
290	20	73	6	10.475	30.950	0	–	1800.00	1800.00
290	20	55	8	14.300	41.900	0	–	1800.00	1800.00
390	20	98	6	13.350	45.300	0	–	1800.00	1800.00
390	20	74	8	19.625	60.400	0	–	1800.00	1800.00

identify minimal 2- and 4-cycles. They are necessary also for the upper bound heuristic. To make the computation as efficient as possible, we note that the bay configuration at the current node does not change greatly from its parent node: Only a single container is relocated. It is true that all retrievable containers should also be retrieved, but they do not affect cycles because containers in a cycle can never be retrieved without relocating at least one container in that cycle, meaning that the retrieved containers do not belong to any cycle. For simplicity, we assume that no containers are retrieved, and suppose that container c is relocated to yard stack y at the current node. Since container c is placed on top of a yard stack at both parent and current nodes, it is never $2n$-blocked. We can easily check whether container c becomes 2-blocking after the relocation, by scanning containers placed in yard stack y. Its time complexity is $O(T^Y)$. With regard to minimal 4-cycles, we assume that a Boolean value b_{ij} is given for every pair of containers i and j ($1 \le i < j \le N$) at the parent node. It takes 1 iff containers i and j are 4-blocking containers in the same minimal 4-cycle. To update b_{ij} at the current node, we first set $b_{jc} := 0$ ($1 \le j < c$) and $b_{cj} := 0$ ($c < j \le N$). Then, we check whether there exists a quadruplet (c, d, e, f) such that containers c, d, e and f yield a minimal 4-cycle in this order. If it is found, we let $b_{ce} := 1$ if $c < e$, and $b_{ec} := 1$ otherwise, unless container e is 2-blocking, noting that container c is not $2n$-blocked and hence 4-blocking. The cycle is in the form $c \xrightarrow{y} d \xrightarrow{v} e \xrightarrow{y} f \xrightarrow{v} c$ (cf. Fig. 5(b)), and it can be rewritten from (1) and (2) as

$$(s^Y_d = y) \wedge (s^V_e = s^V_d) \wedge (t^V_e > t^V_d) \wedge (s^Y_f = s^Y_e) \wedge (t^Y_f < t^Y_e) \wedge (s^V_c = s^V_f) \wedge (t^V_c > t^V_f). \tag{14}$$

To check this relation, we must enumerate all permutations of $d \in \mathcal{Y}_y$, $e \in \check{\mathcal{V}}_d$, and $f \in \{f' \in \mathcal{Y}_{s^Y_e} \mid t^Y_{f'} < t^Y_e\}$. Therefore, the time complexity is given by $O((T^Y)^2 T^V)$, where T^V is the maximum height of vessel stacks. Please note that we compute not $\widetilde{\mathcal{M}}^4$ but its subset in this implementation. Although some pair of containers may be 4-blocking in more than one minimal 4-cycle, we simply check whether it is 4-blocking or not in at least one minimal 4-cycle, ignoring the number of its occurrences. We use this subset in place of $\widetilde{\mathcal{M}}^4$ in the procedure explained in Section 5.1.2. However, it does not violate the validity of LB2c4c as a lower bound. It is because all cycles in $\widetilde{\mathcal{M}}^4$ are disconnected anyway by disconnecting all cycles in the subset.

7. Computational experiments

In this section, we examine the effectiveness of the proposed lower bounds and the exact algorithm using them by computational experiments.

Table 4
Comparison of the branch-and-bound algorithm for LBr with and without Lemma 2 (**bold**: LBr is successfully obtained for all 40 instances).

N	S^V	S^Y	T^Y	without Lemma 2			with Lemma 2		
				solved	CPU time [s]		solved	CPU time [s]	
					ave	max		ave	max
10	3	3	6	**40**	0.00	0.00	**40**	0.00	0.00
16	3	4	6	**40**	0.00	0.00	**40**	0.00	0.00
16	3	3	8	**40**	0.00	0.00	**40**	0.00	0.00
31	3	8	6	**40**	0.00	0.00	**40**	0.00	0.00
31	3	6	8	**40**	0.00	0.00	**40**	0.00	0.00
46	3	12	6	**40**	0.00	0.00	**40**	0.00	0.00
46	3	9	8	**40**	0.00	0.00	**40**	0.00	0.00
19	5	5	6	**40**	0.00	0.00	**40**	0.00	0.00
19	5	4	8	**40**	0.00	0.00	**40**	0.00	0.00
29	5	8	6	**40**	0.00	0.00	**40**	0.00	0.00
29	5	6	8	**40**	0.00	0.00	**40**	0.00	0.00
54	5	14	6	**40**	0.00	0.00	**40**	0.00	0.00
54	5	11	8	**40**	0.00	0.00	**40**	0.00	0.00
79	5	20	6	**40**	0.10	2.90	**40**	0.00	0.00
79	5	15	8	**40**	0.17	5.12	**40**	0.00	0.00
50	10	13	6	**40**	0.00	0.00	**40**	0.00	0.00
50	10	10	8	**40**	0.00	0.00	**40**	0.00	0.00
70	10	18	6	**40**	0.02	0.41	**40**	0.00	0.00
70	10	14	8	**40**	0.01	0.07	**40**	0.00	0.00
120	10	30	6	25	853.97	1800.00	**40**	0.05	0.20
120	10	23	8	18	1113.84	1800.00	**40**	0.06	0.35
170	10	43	6	0	1800.00	1800.00	**40**	4.13	36.33
170	10	32	8	0	1800.00	1800.00	**40**	5.82	30.34
94	15	24	6	39	50.07	1800.00	**40**	0.00	0.06
94	15	18	8	**40**	15.40	313.54	**40**	0.00	0.03
124	15	31	6	19	1078.32	1800.00	**40**	0.15	0.80
124	15	24	8	11	1415.04	1800.00	**40**	0.34	4.40
199	15	50	6	0	1800.00	1800.00	**40**	211.95	1312.81
199	15	38	8	0	1800.00	1800.00	39	364.78	1800.00
274	15	69	6	0	1800.00	1800.00	0	1800.00	1800.00
274	15	52	8	0	1800.00	1800.00	0	1800.00	1800.00
150	20	38	6	10	1430.73	1800.00	**40**	4.42	37.08
150	20	29	8	0	1800.00	1800.00	**40**	7.50	73.82
190	20	48	6	0	1800.00	1800.00	39	255.26	1800.00
190	20	36	8	0	1800.00	1800.00	38	464.65	1800.00
290	20	73	6	0	1800.00	1800.00	0	1800.00	1800.00
290	20	55	8	0	1800.00	1800.00	0	1800.00	1800.00
390	20	98	6	0	1800.00	1800.00	0	1800.00	1800.00
390	20	74	8	0	1800.00	1800.00	0	1800.00	1800.00

Benchmark instances are derived from Jovanovic et al. (2018)[1]. In this dataset, the number of vessel stacks (S^V) and the minimum height of the vessel stacks are chosen from {3, 5, 10, 15, 20} and {3, 5, 10, 15}, respectively. The shape of the vessel bay is symmetric: The stack height increases from left to center one by one, and then decreases from center to right one by one. The height of the yard stacks (T^Y) is either 6 or 8. The number of containers (N) and the number of yard stacks (S^Y) are determined so that the occupancy rate of the yard bay becomes around 66%. The instances are characterized by four parameters (N, S^V, S^Y, T^Y), and the dataset contains 40 randomly generated instances for each combination.

The algorithms are coded in C[2] and the program is run on a desktop computer with an Intel Core i7-6700K CPU (4.0GHz) and 64GB RAM. The time limit is set to 1800s for each instance.

7.1. Comparison of upper bound heuristics for initial bay configurations

First, we examine the effectiveness of the proposed upper bound heuristic for initial bay configurations. In Table 2, average objective values over each set of 40 instances are presented. The results of the five greedy heuristics LT+MBW, MM+MBW, MM4CB+MBW, MM+MBW4CB, and MM4CB+MBW4CB from Jovanovic et al. (2018) are shown in the corresponding columns, and “proposed” stands for the upper bound heuristic proposed in this study. We can confirm that the new heuristic almost always yields the best results. The differences from the previous heuristics are large especially for large-sized instances, and it brings us a 10% reduction in the objective value in several cases.

7.2. Comparison of lower bounds LB2c, LB2c4c, and LBr for initial bay configurations

We next compare the three lower bounds LB2c, LB2c4c, and LBr. In Table 3, the average values of LB2c and LB2c4c over 40 instances are presented in “LB2c” and “LB2c4c”, respectively. With regard to LBr, the branch-and-bound algorithm reaches the time limit of 1800s for some instances. Hence, the number of instances for which LBr is obtained successfully is given in “solved”, and “LB” denotes the average value of LBr over “solved” instances. We also provide the average and maximum computation times in seconds for LBr in “ave” and “max”, respectively (note that “ave” and “max” are always over 40 instances). The computation times of LB2c and LB2c4c are all less than 0.01s and so are omitted in the table. For small-sized instances with $N \leq 31$, the differences among the lower bounds are not pronounced. They become more significant as the instance size becomes larger, and LB2c gives only a poor lower

[1] Available from http://mail.ipb.ac.rs/~rakaj/brlp/brlp.htm.

[2] The source code is available from https://sites.google.com/site/shunjitanaka/brpsp.

Table 5
The number of instances solved to optimality and computation time by the branch-and-bound algorithm for the unrestricted BRPSP (**bold**: all 40 instances are solved to optimality).

N	S^V	S^Y	T^Y	u-bb(LB2c)			u-bb(LB2c4c)			u-bb(LBr)			
				solved	CPU time [s]		solved	CPU time [s]		best	solved	CPU time [s]	
					ave	max		ave	max			ave	max
10	3	3	6	**40**	0.00	0.00	**40**	0.00	0.00	**2.150**	**40**	0.00	0.00
16	3	4	6	**40**	0.00	0.00	**40**	0.00	0.00	**4.950**	**40**	0.00	0.00
16	3	3	8	**40**	0.00	0.00	**40**	0.00	0.00	**7.225**	**40**	0.00	0.00
31	3	8	6	38	222.35	1800.00	**40**	0.04	1.01	**9.625**	**40**	0.00	0.10
31	3	6	8	39	112.36	1800.00	**40**	0.07	1.69	**12.875**	**40**	0.01	0.10
46	3	12	6	1	1755.53	1800.00	35	289.59	1800.00	**15.300**	**40**	2.51	34.02
46	3	9	8	1	1787.85	1800.00	36	355.63	1800.00	**19.825**	**40**	23.29	595.28
19	5	5	6	**40**	0.00	0.01	**40**	0.00	0.00	**3.200**	**40**	0.00	0.00
19	5	4	8	**40**	0.00	0.01	**40**	0.00	0.00	**4.575**	**40**	0.00	0.00
29	5	8	6	**40**	0.74	17.63	**40**	0.00	0.02	**5.775**	**40**	0.00	0.00
29	5	6	8	**40**	1.02	26.31	**40**	0.00	0.01	**7.650**	**40**	0.00	0.00
54	5	14	6	0	1800.00	1800.00	23	956.19	1800.00	**13.450**	**40**	1.83	46.89
54	5	11	8	1	1781.06	1800.00	23	877.57	1800.00	**16.725**	**40**	5.86	73.26
79	5	20	6	0	1800.00	1800.00	0	1800.00	1800.00	22.275	27	632.69	1800.00
79	5	15	8	0	1800.00	1800.00	0	1800.00	1800.00	30.000	18	1091.29	1800.00
50	10	13	6	22	855.83	1800.00	37	226.72	1800.00	**7.450**	**40**	0.00	0.02
50	10	10	8	18	1067.75	1800.00	38	160.51	1800.00	**8.875**	**40**	0.02	0.26
70	10	18	6	2	1713.50	1800.00	6	1570.84	1800.00	**11.700**	**40**	0.60	20.98
70	10	14	8	0	1800.00	1800.00	6	1600.39	1800.00	14.950	39	51.27	1800.00
120	10	30	6	0	1800.00	1800.00	0	1800.00	1800.00	26.375	22	842.52	1800.00
120	10	23	8	0	1800.00	1800.00	0	1800.00	1800.00	37.175	7	1526.67	1800.00
170	10	43	6	0	1800.00	1800.00	0	1800.00	1800.00	47.150	3	1747.71	1800.00
170	10	32	8	0	1800.00	1800.00	0	1800.00	1800.00	66.800	0	1800.00	1800.00
94	15	24	6	1	1773.75	1800.00	1	1756.06	1800.00	12.750	38	114.05	1800.00
94	15	18	8	0	1800.00	1800.00	1	1760.06	1800.00	17.600	34	411.91	1800.00
124	15	31	6	0	1800.00	1800.00	0	1800.00	1800.00	20.775	26	724.90	1800.00
124	15	24	8	0	1800.00	1800.00	0	1800.00	1800.00	28.500	18	1083.95	1800.00
199	15	50	6	0	1800.00	1800.00	0	1800.00	1800.00	58.425	0	1800.00	1800.00
199	15	38	8	0	1800.00	1800.00	0	1800.00	1800.00	77.000	0	1800.00	1800.00
150	20	38	6	0	1800.00	1800.00	0	1800.00	1800.00	25.400	18	1104.00	1800.00
150	20	29	8	0	1800.00	1800.00	0	1800.00	1800.00	35.075	8	1623.30	1800.00
190	20	48	6	0	1800.00	1800.00	0	1800.00	1800.00	46.300	1	1792.10	1800.00
190	20	36	8	0	1800.00	1800.00	0	1800.00	1800.00	63.900	0	1800.00	1800.00
total				403			566				899		

bound when $N \geq 46$. LB2c4c is much better than LB2c even in this case, but LBr is providing further improvements, granted that it becomes intractable when $N \geq 190$.

To further investigate the impact of the dominance rule (Lemma 2) on the branch-and-bound algorithm for LBr, we show the computation time with and without the dominance rules in Table 4. We can see that the dominance rule greatly contributes to reducing the computation time for larger-sized instances. Without it, we fail in obtaining LBr for one instance with $N = 94$, whereas it enables us to obtain LBr for all instances with $N \leq 170$.

7.3. Branch-and-bound algorithm for the unrestricted BRPSP

We already know that LBr is considerably better than LB2c and LB2c4c. However, its effectiveness as a lower bound in exact algorithms is still unclear because it requires longer computation time than the other two. To investigate the effectiveness, we apply the proposed branch-and-bound algorithm with the lower bound being changed. The results for the unrestricted BRPSP are presented in Table 5. In this table, "u-bb(LB2c)", "u-bb(LB2c4c)", and "u-bb(LBr)" indicate the branch-and-bound algorithm with LB2c, LB2c4c and LBr, respectively. The number of instances solved to optimality is provided in "solved", and "ave" and "max" stand for the average and maximum computation times, respectively. The average objective value of optimal (or best on instances not solved to optimality) solutions is shown in "best". It now turns out that LBr is more appropriate than LB2c and LB2c4c as a lower bound in the proposed branch-and-bound algorithm. Indeed, u-bb(LBr) is capable of solving instances with $N \leq 70$ to optimality, the size of which is twice as large as $N \leq 29$ of u-bb(LB2c) and $N \leq 31$ of u-bb(LB2c4c). Comparing Tables 3 and 5, we notice that LBr yields such a good lower bound that the optimality gap is less than one on average for instances with $N \leq 70$. On the other hand, the initial upper bound is not very tight, according to Table 2. It seems difficult to obtain a good solution only by simple greedy heuristics. We also notice that the difference between the initial upper bound and the optimal (or best) objective value obtained by the branch-and-bound algorithm is relatively small for instances with $N \geq 150$. The branch-and-bound algorithm can explore only a few nodes due to long computation times for LBr, and thus fails to find good solutions for these instances.

7.4. Branch-and-bound algorithm for the restricted BRPSP

In Table 6, we summarize the results by the branch-and-bound algorithm for the restricted BRPSP ("r-bb(*)"). In "best in Jovanovic et al. (2018)", we provide best solutions obtained by all greedy heuristics and GRASPs in Jovanovic et al. (2018). We can verify that LBr is better than LB2c and LB2c4c also for the restricted BRPSP. Comparing Table 6 with Table 5, we find that r-bb(LBr) solves more instances to optimality than u-bb(LBr): 978 instances by r-bb(LBr), as opposed to 899 instances by u-bb(LBr). It is not surprising that the restricted problem is easier to solve than the unrestricted problem because the constraint on relocatable containers greatly reduces the search space. Nevertheless, the differences in optimal values of restricted and unrestricted problems are minor at least

Table 6
The number of instances solved to optimality and computation time by the branch-and-bound algorithm for the restricted BRPSP (**bold**: all 40 instances are solved to optimality).

N	S^V	S^Y	T^Y	best in Jovanovic et al. (2018)	r-bb(LB2c)			r-bb(LB2c4c)			r-bb(LBr)			
					solved	CPU time [s]		solved	CPU time [s]		best	solved	CPU time [s]	
						ave	max		ave	max			ave	max
10	3	3	6	2.150	**40**	0.00	0.00	**40**	0.00	0.00	**2.150**	**40**	0.00	0.00
16	3	4	6	5.075	**40**	0.00	0.00	**40**	0.00	0.00	**5.025**	**40**	0.00	0.00
16	3	3	8	7.575	**40**	0.00	0.00	**40**	0.00	0.00	**7.300**	**40**	0.00	0.00
31	3	8	6	10.225	**40**	0.46	11.12	**40**	0.00	0.01	**9.650**	**40**	0.00	0.00
31	3	6	8	14.325	**40**	0.47	5.23	**40**	0.01	0.07	**13.325**	**40**	0.02	0.85
46	3	12	6	16.725	20	1043.99	1800.00	**40**	0.45	10.05	**15.325**	**40**	0.00	0.03
46	3	9	8	22.625	10	1454.10	1800.00	**40**	10.77	179.85	**20.525**	**40**	1.11	36.14
19	5	5	6	3.225	**40**	0.00	0.00	**40**	0.00	0.00	**3.200**	**40**	0.00	0.00
19	5	4	8	4.800	**40**	0.00	0.00	**40**	0.00	0.00	**4.575**	**40**	0.00	0.00
29	5	8	6	6.100	**40**	0.01	0.08	**40**	0.00	0.00	**5.800**	**40**	0.00	0.00
29	5	6	8	8.175	**40**	0.02	0.46	**40**	0.00	0.00	**7.650**	**40**	0.00	0.00
54	5	14	6	14.475	7	1555.73	1800.00	38	123.43	1800.00	**13.450**	**40**	0.04	1.55
54	5	11	8	18.700	7	1630.02	1800.00	**40**	123.61	1341.91	**16.900**	**40**	0.06	0.72
79	5	20	6	24.625	0	1800.00	1800.00	2	1749.87	1800.00	21.925	37	162.22	1800.00
79	5	15	8	33.675	0	1800.00	1800.00	2	1734.90	1800.00	28.850	37	238.62	1800.00
50	10	13	6	7.775	37	239.98	1800.00	**40**	20.99	798.41	**7.450**	**40**	0.00	0.00
50	10	10	8	9.375	34	345.35	1800.00	**40**	0.65	9.39	**8.875**	**40**	0.00	0.02
70	10	18	6	13.250	4	1651.09	1800.00	24	826.20	1800.00	**11.725**	**40**	0.02	0.22
70	10	14	8	16.750	4	1644.14	1800.00	17	1151.37	1800.00	**14.950**	**40**	0.23	6.24
120	10	30	6	30.525	0	1800.00	1800.00	0	1800.00	1800.00	26.850	23	865.08	1800.00
120	10	23	8	40.825	0	1800.00	1800.00	0	1800.00	1800.00	37.325	14	1296.40	1800.00
170	10	43	6	48.200	0	1800.00	1800.00	0	1800.00	1800.00	43.775	13	1359.35	1800.00
170	10	32	8	64.675	0	1800.00	1800.00	0	1800.00	1800.00	66.850	2	1723.10	1800.00
94	15	24	6	14.625	1	1755.82	1800.00	6	1628.67	1800.00	12.600	39	69.83	1800.00
94	15	18	8	20.325	1	1757.61	1800.00	3	1692.40	1800.00	**17.350**	**40**	3.29	77.00
124	15	31	6	24.300	0	1800.00	1800.00	0	1800.00	1800.00	20.175	33	411.58	1800.00
124	15	24	8	32.375	0	1800.00	1800.00	0	1800.00	1800.00	28.225	22	894.40	1800.00
199	15	50	6	51.500	0	1800.00	1800.00	0	1800.00	1800.00	56.975	0	1800.00	1800.00
199	15	38	8	66.425	0	1800.00	1800.00	0	1800.00	1800.00	76.200	0	1800.00	1800.00
150	20	38	6	27.650	0	1800.00	1800.00	0	1800.00	1800.00	23.650	24	880.58	1800.00
150	20	29	8	36.650	0	1800.00	1800.00	0	1800.00	1800.00	32.525	14	1290.68	1800.00
190	20	48	6	41.675	0	1800.00	1800.00	0	1800.00	1800.00	45.200	0	1800.00	1800.00
190	20	36	8	55.375	0	1800.00	1800.00	0	1800.00	1800.00	63.650	0	1800.00	1800.00
total					485			652				978		

for the instances solved to optimality. For medium- and large-sized instances with $N \geq 79$, the best objective value of the restricted problem is sometimes better than that of the unrestricted problem due to the poor performance of the branch-and-bound algorithm for the unrestricted problem. In practical situations, the restricted BRPSP might be sufficient and we need not solve the unrestricted BRPSP. Compared with our algorithm, the greedy heuristics and GRASPs in Jovanovic et al. (2018) find optimal and near-optimal solutions for small-sized instances, but they become less efficient as the instance size becomes larger. In particular, the gap from our best objective value is almost 5 on average for instances with (N, S^V, S^Y, T^Y) = (79, 5, 15, 8). It is thought that there is still room for improvement in heuristic and metaheuristic approaches.

8. Conclusion

This paper addressed the block relocation problem with a stowage plan and constructed an exact algorithm for both restricted and unrestricted variants of the problem. To this end, we proposed three lower bounds: LB2c and LB2c4c based on precedence cycles, and LBr based on a relaxation of the original problem. The results of the computational experiments indicate that the branch-and-bound algorithm with LBr is capable of solving small- and medium-sized instances to optimality in a reasonable computation time. However, LBr is not easy to compute, which leads to the ineffectiveness of the branch-and-bound algorithm for large-sized instances. It would be interesting to develop faster lower bounds as well as better algorithms for LBr. It would also be worthwhile to construct tree search-based metaheuristics such as a beam search algorithm. These topics are left for future research.

Appendix A. Brief proofs of dominance rules

A.1. *Proof of Lemma 3*

From Condition 2, container c_1 is not relocated by the sequence $(c_2, s_2, d_2), \dots, (c_{n-1}, s_{n-1}, d_{n-1})$. From Condition 4, container c_1 does not block any retrieval from yard stack s_1 even if it is not relocated. Furthermore, from Condition 3, container c_1 does not interfere any relocation in the sequence even if it is not relocated. Therefore, we can move (c_1, s_1, d_1) just before (c_n, s_n, d_n) and combine (c_1, s_1, d_1) and (c_n, s_n, d_n) into (c_1, s_1, d_n) from Condition 1. In other words, there exists a better sequence $(c_2, s_2, d_2), \dots, (c_{n-1}, s_{n-1}, d_{n-1}), (c_1, s_1, d_n)$. Strictly speaking, it can be infeasible. Container c_1 may block some retrieval from yard stack d_1 when it is relocated by (c_1, s_1, d_1). In this case, it is possible that container c_1 is retrieved by the sequence $(c_2, s_2, d_2), \dots, (c_{n-1}, s_{n-1}, d_{n-1})$. However, it does not matter at all because it merely means that a further better sequence $(c_2, s_2, d_2), \dots, (c_{n-1}, s_{n-1}, d_{n-1})$ exists. The same situation also arises in the other dominance rules, but it is ignored for the sake of brevity in the following proofs. For more detailed discussions, please refer to Tanaka and Mizuno (2018).

A.2. *Proof of Lemma 4*

From Condition 2, container c_1 is not relocated by the sequence $(c_2, s_2, d_2), \dots, (c_{n-1}, s_{n-1}, d_{n-1})$. From Condition 4, container

c_1 does not block any retrieval from yard stack d_n even if it is relocated not to yard stack d_1 but to d_n. From Condition 3, container c_1 does not interfere any relocation in the sequence even if it is relocated to d_n. Therefore, a better sequence (c_1, s_1, d_n), ..., $(c_{n-1}, s_{n-1}, d_{n-1})$ exists.

A.3. Proof of Lemma 5

From Condition 2, container c_1 is not relocated by the sequence (c_2, s_2, d_2), ..., $(c_{n-1}, s_{n-1}, d_{n-1})$. From Condition 4, container c_1 does not block any retrieval from yard stack d'_1 even if it is relocated not to yard stack d_1 but to d'_1. From Condition 3, container c_1 does not interfere any relocation in the sequence even if it is relocated to d'_1. Therefore, there exists another sequence (c_1, s_1, d'_1), ..., $(c_{n-1}, s_{n-1}, d_{n-1})$, (c_n, d'_1, d_n) where container c_1 is first relocated to d'_1 and then to d_n, which is at least as good as the original sequence. Condition 5 is for tie-breaking.

A.4. Proof of Lemma 6

From Condition 1, container c_1 is not relocated by the sequence (c_2, s_2, d_2), ..., $(c_{n-1}, s_{n-1}, d_{n-1})$. It also ensures that the sequence does not relocate any container from or to yard stacks s_1 and d_1. From Condition 2, container c_1 does not block any retrieval from yard stack s_1 by the sequence even if it is not relocated to d_1. Therefore, we can move (c_1, s_1, d_1) just before (c_n, s_n, d_n). Again from Condition 1, $\{s_1, d_1\} \cap \{s_n, d_n\} = \emptyset$ and thus $c_1 \neq c_n$. If (c_1, s_1, d_1) is moved further after (c_n, s_n, d_n), some retrieval from yard stack s_1 and successive retrievals enabled by (c_n, s_n, d_n) may be blocked by container c_1. However, they are enabled anyway by (c_1, s_1, d_1). Thus, the sequence (c_2, s_2, d_2), ..., (c_n, s_n, d_n), (c_1, s_1, d_1) is as good as the original sequence. Condition 3 is for tie-breaking.

A.5. Proof of Lemma 7

From Condition 1, container c_1 is not relocated by the sequence (c_2, s_2, d_2), ..., (c_n, s_n, d_n). It is retrieved after (c_n, s_n, d_n) from Condition 2. From Condition 4, container c_1 does not block any retrieval from yard stack s_1 even if it is not relocated from there. From Condition 3, container c_1 does not interfere any relocation in the sequence even if it is not relocated. Furthermore, container c_1 is on top of s_1 in this case when container c_n is relocated, so that container c_1 is retrieved by this sequence. Therefore, there exists a better sequence (c_2, s_2, d_2), ..., $(c_{n-1}, s_{n-1}, d_{n-1})$, ($c_n$, s_n, d_n).

A.6. Proof of Lemma 8

From Condition 1, container c_1 is not relocated by the sequence (c_2, s_2, d_2), ..., (c_n, s_n, d_n). It is retrieved after (c_n, s_n, d_n) from Condition 2. From Condition 4, container c_1 does not block any retrieval from yard stack d'_1 even if it is relocated not to yard stack d_1 but to d'_1. From Condition 3, container c_1 does not interfere any relocation in the sequence even if it is relocated to d'_1. Furthermore, container c_1 is on top of d'_1 in this case when container c_n is relocated, so that container c_1 is retrieved by the sequence (c_1, s_1, d'_1), (c_2, s_2, d_2), ..., (c_n, s_n, d_n). Therefore, it is as good as the original sequence. Condition 5 is for tie-breaking.

References

United Nations Conference on Trade and Development (UNICTAD). Container port throughput, annual, 2010–2017. http://unctadstat.unctad.org/wds/TableViewer/tableView.aspx?ReportId=13321 (accessed September 10, 2018).

Azari, E., Eskandari, H., & Nourmohammadi, A. (2017). Decreasing the crane working time in retrieving the containers from a bay. *Scientia Iranica. Transaction E: Industrial Engineering, 24*, 309–318.

Bortfeldt, A., & Forster, F. (2012). A tree search procedure for the container pre-marshalling problem. *European Journal of Operational Research, 217*, 531–540.

van Brink, M., & van der Zwaan, R. (2014). A branch and price procedure for the container premarshalling problem. *Lecture Notes in Computer Science, 8737*, 798–809.

Caserta, M., Schwarze, S., & Voß, S. (2009). A new binary description of the blocks relocation problem and benefits in a look ahead heuristic. *Lecture Notes in Computer Science, 5482*, 37–48.

Caserta, M., Schwarze, S., & Voß, S. (2011a). Container rehandling at maritime container terminals. In J. W. Böse (Ed.), *Handbook of terminal planning. operations research/computer science interfaces series 49* (pp. 247–269). New York: Springer.

Caserta, M., Schwarze, S., & Voß, S. (2012). A mathematical formulation and complexity considerations for the blocks relocation problem. *European Journal of Operational Research, 219*, 96–104.

Caserta, M., & Voß, S. (2009a). A corridor method-based algorithm for the pre-marshalling problem. *Lecture Notes in Computer Science, 5484*, 788–797.

Caserta, M., & Voß, S. (2009b). Corridor selection and fine tuning for the corridor method. *Lecture Notes in Computer Science, 5851*, 163–175.

Caserta, M., Voß, S., & Sniedovich, M. (2011b). Applying the corridor method to a blocks relocation problem. *OR Spectrum, 33*, 915–929.

Eskandari, H., & Azari, E. (2015). Notes on mathematical formulation and complexity considerations for blocks relocation problem. *Scientia Iranica. Transaction E: Industrial Engineering, 22*, 2722–2728.

Expósito-Izquierdo, C., Melián-Batista, B., & Moreno-Vega, J. M. (2014). A domain-specific knowledge-based heuristic for the Blocks Relocation Problem. *Advanced Engineering Informatics, 28*, 327–343.

Expósito-Izquierdo, C., Melián-Batista, B., & Moreno-Vega, J. M. (2015). An exact approach for the Blocks Relocation Problem. *Expert Systems with Applications, 42*, 6408–6422.

Expósito-Izquierdo, C., Melián-Batista, B., & Moreno-Vega, M. (2012). Pre-Marshalling Problem: Heuristic solution method and instances generator. *Expert Systems with Applications, 39*, 8337–8349.

Forster, F., & Bortfeldt, A. (2012). A tree search procedure for the container relocation problem. *Computers & Operations Research, 39*, 299–309.

Galle, V., Barnhart, C., & Jaillet, P. (2018). A new binary formulation of the restricted container relocation problem based on a binary encoding of configurations. *European Journal of Operational Research, 267*, 467–477.

Garey, M. R., & Johnson, D. S. (1979). Computers and intractability: A guide to the theory of NP-completeness. New York: W.H. Freeman and Company. 223

Hottung, A., & Tierney, K. (2016). A biased random-key genetic algorithm for the container pre-marshalling problem. *Computers & Operations Research, 75*, 83–102.

Huang, S. H., & Lin, T. H. (2012). Heuristic algorithms for container pre-marshalling problems. *Computers & Industrial Engineering, 62*, 13–20.

Ji, M., Guo, W., Zhu, H., & Yang, Y. (2015). Optimization of loading sequence and rehandling strategy for multi-quay crane operations in container terminals. *Transportation Research Part E: Logistics and Transportation Review, 80*, 1–19.

Jin, B., Zhu, W., & Lim, A. (2015). Solving the container relocation problem by an improved greedy look-ahead heuristic. *European Journal of Operational Research, 240*, 837–847.

Jovanovic, R., Tanaka, S., Nishi, T., & Voß, S. (2018). A GRASP approach for solving the Blocks Relocation Problem with Stowage Plan. *Flexible Services and Manufacturing Journal*. doi:10.1007/s10696-018-9320-3. Available online.

Jovanovic, R., Tuba, M., & Voß, S. (2017). A multi-heuristic approach for solving the pre-marshalling problem. *Central European Journal of Operations Research, 24*, 1–28.

Jovanovic, R., & Voß, S. (2014). A chain heuristic for the Blocks Relocation Problem. *Computers & Industrial Engineering, 75*, 79–86.

Jovanovic, R., Tuba, M., & Voß, S. (2019). An efficient ant colony optimization algorithm for the Blocks Relocation Problem. *European Journal of Operational Research, 274*, 78–90.

Kim, K. H., & Hong, G. P. (2006). A heuristic rule for relocating blocks. *Computers & Operations Research, 33*, 940–954.

Ku, D., & Arthanari, T. S. (2016). On the abstraction method for the container relocation problem. *Computers & Operations Research, 68*, 110–122.

Lee, Y., & Chao, S.-L. (2009). A neighborhood search heuristic for pre-marshalling export containers. *European Journal of Operational Research, 196*, 468–475.

Lee, Y., & Hsu, N.-Y. (2007). An optimization model for the container pre-marshalling problem. *Computers & Operations Research, 34*, 3295–3313.

Lee, Y., & Lee, Y.-J. (2010). A heuristic for retrieving containers from a yard. *Computers & Operations Research, 37*, 1139–1147.

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

Lin, D.-Y., Lee, Y.-J., & Lee, Y. (2015). The container retrieval problem with respect to relocation. *Transportation Research Part C: Emerging Technologies, 52*, 132–143.

López-Plata, I., Expósito-Izquierdo, C., Lalla-Ruiz, E., Melián-Batista, B., & Moreno-Vega, J. M. (2017). Minimizing the waiting times of block retrieval operations in stacking facilities. *Computers & Industrial Engineering, 103*, 70–84.

Petering, M. E. H., & Hussein, M. I. (2013). A new mixed integer program and extended look-ahead heuristic algorithm for the block relocation problem. *European Journal of Operational Research, 231*, 120–130.

Quispe, K. E. Y., Lintzmayer, C. N., & Xavier, E. C. (2018). An exact algorithm for the blocks relocation problem with new lower bounds. *Computers & Operations Research, 99*, 206–217.

Schwarze, S., & Voß, S. (2014). *Some remarks on alternative objectives for the blocks relocation problem*. University of Hamburg.

de Melo da Silva, M., Erdoğan, G., Battarra, M., & Strusevich, V. (2018a). The block retrieval problem. *European Journal of Operational Research, 265*, 931–950.

de Melo da Silva, M., Toulouse, S., & Wolfler Calvo, R. (2018b). A new effective unified model for solving the pre-marshalling and block relocation problems. *European Journal of Operational Research, 271*, 40–56.

da Silva Firmino, A., de Abreu Silva, R. M., & Times, V. C. (2019). A reactive GRASP metaheuristic for the container retrieval problem to reduce crane's working time. *Journal of Heuristics, 25*, 141–173.

Tanaka, S., & Mizuno, F. (2018). An exact algorithm for the unrestricted block relocation problem. *Computers & Operations Research, 95*, 12–31.

Tanaka, S., & Takii, K. (2016). A faster branch-and-bound algorithm for the block relocation problem. *IEEE Transactions on Automation Science and Engineering, 13*, 181–190.

Tanaka, S., & Tierney, K. (2018). Solving real-world sized container pre-marshalling problems with an iterative deepening branch-and-bound algorithm. *European Journal of Operational Research, 264*, 165–180.

Tierney, K., Pacino, D., & Voß, S. (2017). Solving the pre-marshalling problem to optimality with A* and IDA*. *Flexible Services and Manufacturing Journal, 29*, 223–259.

Tierney, K., & Voß, S. (2016). Solving the robust container pre-marshalling problem. *Lecture Notes in Computer Science, 9855*, 131–145.

Ting, C.-J., & Wu, K.-C. (2017). Optimizing container relocation operations at container yards with beam search. *Transportation Research Part E: Logistics and Transportation Review, 103*, 17–31.

Tricoire, F., Scagnetti, J., & Beham, A. (2018). New insights on the block relocation problem. *Computers & Operations Research, 89*, 127–139.

Ünlüyurt, T., & Aydın, C. (2012). Improved rehandling strategies for the container retrieval process. *Journal of Advanced Transportation, 46*, 378–393.

Wan, Y.-w., Liu, J., & Tsai, P.-C. (2009). The assignment of storage locations to containers for a container stack. *Naval Research Logistics, 56*, 699–713.

Wang, N., Jin, B., & Lim, A. (2015). Target-guided algorithms for the container pre–marshalling problem. *Omega, 53*, 67–77.

Wang, N., Jin, B., Zhang, Z., & Lim, A. (2017). A feasibility-based heuristic for the container pre-marshalling problem. *European Journal of Operational Research, 256*, 90–101.

Zehendner, E., Caserta, M., Feillet, D., Schwarze, S., & Voß, S. (2015). An improved mathematical formulation for the blocks relocation problem. *European Journal of Operational Research, 245*, 415–422.

Zehendner, E., & Feillet, D. (2014). A branch and price approach for the container relocation problem. *International Journal of Production Research, 52*, 7159–7176.

Zhang, R., Jiang, Z.-Z., & Yun, W. Y. (2015). Stack pre-marshalling problem: A heuristic-guided branch-and-bound algorithm. *International Journal of Industrial Engineering, 22*, 509–523.

Zhu, W., Qin, H., Lim, A., & Zhang, H. (2012). Iterative deepening A* algorithms for the container relocation problem. *IEEE Transactions on Automation Science and Engineering, 9*, 710–722.