Computers and Operations Research 95 (2018) 12–31

Contents lists available at ScienceDirect

Computers and Operations Research

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

An exact algorithm for the unrestricted block relocation problem☆

Shunji Tanaka[a,*], Fumitaka Mizuno[b]

[a] Institute for Liberal Arts and Sciences, Kyoto University, Kyotodaigaku-Katsura, Nishikyo-ku, Kyoto 615-8510, Japan
[b] Department of Electrical Engineering, Kyoto University, Kyotodaigaku-Katsura, Nishikyo-ku, Kyoto 615-8510, Japan

ARTICLE INFO

Article history:
Received 4 November 2016
Revised 14 December 2017
Accepted 27 February 2018
Available online 28 February 2018

Keywords:
Block relocation problem
Container relocation problem
Exact algorithm
Dominance properties
Lower bound

ABSTRACT

The purpose of this study is to propose an exact algorithm for the unrestricted block relocation problem with distinct priorities. In this problem, a storage area is considered where blocks of the same size are stacked vertically in tiers. Because we can access only topmost blocks, relocations of blocks are required when other blocks are retrieved. The objective is to minimize the total number of such relocations necessary for retrieving all the blocks one by one according to a specified order. In the restricted version of this problem, only the topmost block above the target block is relocatable. On the other hand, no such restriction is imposed on the unrestricted problem, which is considered in this study. We also assume that each block is assigned a distinct retrieval priority and the retrieval order of blocks is unique. To improve the efficiency of a branch-and-bound algorithm for this problem, we propose several dominance properties to eliminate unnecessary nodes in the search tree. Furthermore, we propose a new lower bound of the total number of relocations. The effectiveness of the proposed exact algorithm is verified by numerical experiments for benchmark instances in the literature.

1. Introduction

1.1. Background

This paper addresses a retrieval problem of stacked items that typically arises in container terminals. Containers that have arrived at a container terminal via sea or land transport are stored temporarily in yards of the container terminal and wait for future transport there. Due to space limitations, containers are stacked vertically as illustrated in Fig. 1. Here, each column of containers is referred to as a stack, and a single row of stacks is referred to as a bay. When retrieval of a container is requested, a gantry crane picks it up from a bunch of containers and puts it on a truck for further transport. However, only the topmost containers of the stacks are accessible from the crane, so that containers above the target container should be relocated (reshuffled) to other places beforehand. Such containers are usually relocated to other stacks in the same bay because crane travel across bays is more time-consuming than that within the same bay. To determine destination stacks of the containers, we need to take future retrieval into consideration; otherwise, relocated containers may block the next target container and cause further relocations. Reduction of such unproductive relocations is crucial for improving the throughput of container handling in a sea port. For this reason, the container relocation problem or the block(s) relocation problem has been extensively studied in the literature. For the sake of generality, we refer to a container as a block throughout this paper. Accordingly, our problem is referred to as the block relocation problem, or, simply the BRP.

1.2. Formal description and classification of the block relocation problem

The BRP in this study is formally described as follows. Consider a bay composed of S stacks. The sth stack ($1 \le s \le S$) is referred to as stack s. In this bay, N blocks of the same size are stored in tiers. Due to the crane height, the number of blocks in each stack (the stack height) is limited to T. Each block is given a unique integer priority value between 1 and N, and the block with priority i is referred to as block i. An example of a bay with $S = 4$, $T = 3$ and $N = 10$ is illustrated in Fig. 2. All the blocks should be retrieved from the bay one by one in the increasing order of their priorities, which is achieved by the following two types of crane operations:

- Relocation
  The topmost block of a stack is moved to the top of another stack whose height does not reach the limit.

☆ Preliminary versions of this study were presented at 2015 IEEE International Conference on Automation Science and Engineering (IEEE CASE 2015), August 2015, and 2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM 2015), December 2015.

* Corresponding author.
E-mail address: tanaka@kuee.kyoto-u.ac.jp (S. Tanaka).

https://doi.org/10.1016/j.cor.2018.02.019
0305-0548/

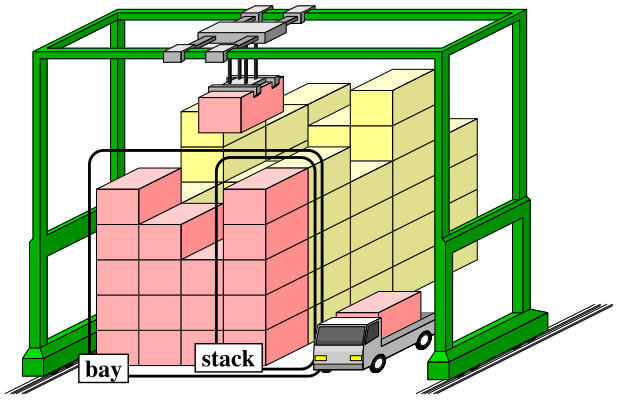


Fig. 1. Containers stored in a container yard equipped with a gantry crane.

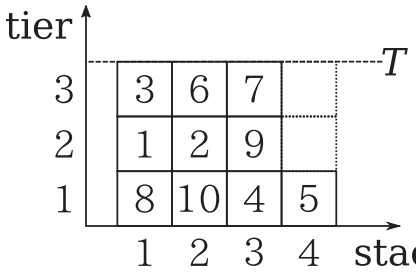


Fig. 2. A bay with $S = 4$, $T = 3$, and $N = 10$. The number in each block is its priority value.

- Retrieval
  The block to be retrieved next is removed from the bay if it is on the top of a stack.

We need not consider any retrieval explicitly because we can safely assume that blocks are retrieved as soon as they become retrievable. Therefore, a solution is represented by a sequence of relocations. The objective of the BRP is to find a shortest feasible sequence of relocations that retrieves all the blocks from the bay. An example of an optimal solution for the bay configuration in Fig. 2 is presented in Fig. 3. The total number of relocations is four, and the solution is expressed by the following sequence of relocations: Block 3 is relocated from stack 1 to stack 4, block 7 is relocated from stack 3 to stack 1, block 6 is relocated from stack 2 to stack 1, and block 9 is relocated from stack 3 to stack 2.

The BRP can be classified from two aspects. The one is priorities of blocks. In the problem with distinct priorities, all the priority values are different. It follows that the retrieval order is uniquely determined. On the other hand, some blocks are assigned the same priority value in the problem with duplicate priorities, and the retrieval order among them is arbitrary. In other words, the retrieval order is given not among individual blocks but among groups of blocks. The other aspect concerns relocatable blocks. In the unrestricted problem, all topmost blocks are relocatable, while in the restricted problem we can relocate only the topmost block of the stack where the block to be retrieved next is stored. According to this definition, however, it is still unclear which blocks are relocatable in the restricted problem with duplicate priorities, because the block to be retrieved next is not uniquely determined. To avoid this ambiguity, it is assumed that once the block to be retrieved next is determined from those with the highest (smallest) priority, it cannot be changed until it is retrieved.

1.3. Literature review

There are rich studies on the BRP and related problems as surveyed by Lehnfeld and Knust (2014). With regard to the complexity of the problem, Blasum et al. (1999) proved the NP-completeness of the tram problem that is similar to the BRP with duplicate priorities. This problem aims to retrieve tram cars of different types from sidings of a depot in a specified order without any relocations. For the problem of minimizing the total number of relocations, Caserta et al. (2012) proved the NP-hardness, following the results by König et al. (2007).

Kim and Hong (2006) first considered the BRP in the form explained in Section 1.2. They proposed a depth-first branch-and-bound algorithm and a greedy heuristic for the restricted problem with duplicate priorities. After this research, various heuristic and exact algorithms have been developed. Wan et al. (2009) provided a binary ILP formulation and several greedy heuristics for the restricted BRP with distinct priorities. They also considered the dynamic version of the BRP where blocks dynamically arrive at the bay. Caserta et al. (2009) proposed a binary matrix representation of bay configurations and constructed a greedy look-ahead heuristic using it. Their heuristic is also referred to as the min-max heuristic. Caserta and Voß (2009b) and Caserta et al. (2011) applied a metaheuristic algorithm called the Corridor Method to the restricted BRP with distinct priorities. Lee and Lee (2010) proposed a three-phase heuristic to minimize crane operation time for the restricted BRP with distinct priorities where multiple bays were considered. Ünlüyurt and Aydın (2012) proposed a depth-first branch-and-bound algorithm and greedy heuristics for the restricted BRP with distinct priorities. As Lee and Lee (2010), they considered minimization of crane operation time. Caserta et al. (2012) derived binary ILP formulations of the restricted and unrestricted BRPs with distinct priorities, respectively. Forster and Bortfeldt (2012) constructed a metaheuristic based on a tree search for the unrestricted BRP with duplicate priorities. Zhu et al. (2012) proposed a depth-first iterative deepening A* (IDA*) algorithm as well as several greedy heuristics for the restricted and unrestricted BRPs with distinct priorities. Petering and Hussein (2013) proposed an MILP formulation for the unrestricted BRP with distinct priorities. They also proposed the LA-N heuristic by extending the look-ahead heuristic in Caserta et al. (2009). Jovanovic and Voß (2014) proposed a chain heuristic for the restricted BRP with distinct priorities, which improves the look-ahead heuristic by introducing a heuristic function that takes into account future relocations. Zehendner and Feillet (2014) proposed a branch-and-price algorithm for the restricted BRP with distinct priorities. Expósito-Izquierdo et al. (2014) applied a best-first A* algorithm for the restricted and unrestricted BRPs with distinct priorities, and proposed a domain-specific knowledge based heuristic for the unrestricted BRP with distinct priorities. The same authors (Expósito-Izquierdo et al., 2015) later improved their exact algorithm for the restricted BRP with distinct priorities and corrected a mistake in the formulation of the restricted BRP proposed in Caserta et al. (2012). Jin et al. (2015) proposed a look-ahead heuristic that employs a tree search for the unrestricted problem with distinct priorities. Zehendner et al. (2015) also corrected the mistake in the formulation (Caserta et al., 2012) and improved it for the restricted BRP with distinct priorities. Lin et al. (2015) proposed a heuristic for the restricted BRP with duplicate priorities. In this problem, they considered multiple bays and assumed a crane with multiple spreaders capable of moving more than one container simultaneously. Eskandari and Azari (2015) corrected the formulation in Caserta et al. (2012) and proposed some cuts to improve it. Tanaka and Takii (2016) improved the lower bound in Zhu et al. (2012) and proposed a depth-first branch-and-bound algorithm for the restricted BRP with distinct and dupli-

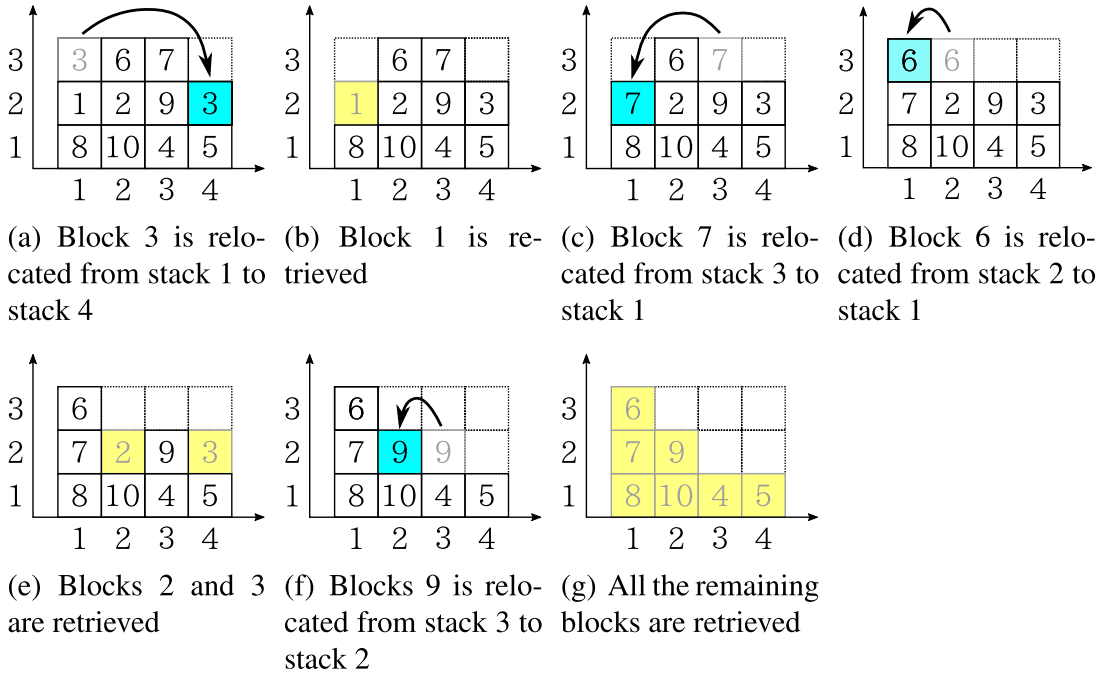


Fig. 3. An example of an optimal solution for the bay configuration in Fig. 2. The solution is expressed by the following sequence of relocations: (1) block 3 is relocated from stack 1 to stack 4, (2) block 7 is relocated from stack 3 to stack 1, (3) block 6 is relocated from stack 2 to stack 1, (4) block 9 is relocated from stack 3 to stack 2

cate priorities. Ku and Arthanari (2016b) proposed another depth-first branch-and-bound algorithm for the restricted BRP with distinct priorities. Recently, Zhang et al. (2016) considered the unrestricted BRP with distinct priorities that allows batch moves to relocate/retrieve several blocks at a time, and proposed a metaheuristic based on a tree search. Tricoire et al. (2018) proposed efficient greedy and metaheuristic algorithms for the unrestricted BRP with distinct priorities. They also proposed a new lower bound as a generalization of the lower bound by Forster and Bortfeldt (2012), and applied it in their branch-and-bound algorithm. In spite of the fact the new lower bound is always tighter than the lower bound in Forster and Bortfeldt (2012), the average computation time of the branch-and-bound algorithm with the former was almost always longer than that with the latter. de Melo et al. (2017) considered a special class of the BRP with duplicate priorities with two priority values, that is, blocks are composed of only two groups. They also considered a lexicographic bi-objective problem whose secondary objective is the expected number of relocations of the forthcoming retrieval. Galle et al. (2017) proposed another ILP formulation of the restricted BRP with distinct priorities based on binary encoding of bay configurations (Caserta et al., 2009). Their formulation currently yields the best IP approach for this class of problem.

Relocation of items is studied not only as the BRP but also in various forms. In the dynamic or online BRP (Akyüz and Lee, 2014; Zehendner et al., 2017) incoming blocks as well as outgoing blocks are considered as in Wan et al. (2009). A stochastic version of the BRP is also studied in Ku and Arthanari (2016a). The ship stowage problem (Avriel and Penn, 1993; Avriel et al., 2000; 1998; Dubrovsky et al., 2002; Tierney et al., 2014; Wang et al., 2014) also considers incoming and outgoing containers, but assumes that they are transported from ports to ports by a container vessel. When a container is unloaded from the vessel at its destination port, those placed above are unloaded together and then loaded again with containers transported from there. Slabs in place of containers are considered in the slab stack shuffling problem (Cheng and Tang, 2010; Kim et al., 2011; Singh et al., 2004; Tang et al., 2001; 2002; Tang and Ren, 2010; Tang et al., 2012). In addition to this difference in blocks, relocations are treated in different ways: Relocated slabs are moved back to the original stack in the same order immediately after the target slab is retrieved, or slabs are relocated at most once (a sufficient number of empty stacks are assumed). The pre-marshalling or re-marshalling problem (Bortfeldt and Forster, 2012; van Brink, 2014; Caserta and Voß, 2009a; Choe et al., 2011; Expósito-Izquierdo et al., 2012; Hottung and Tierney, 2016; Huang and Lin, 2012; Jovanovic et al., 2017; Lee and Chao, 2009; Lee and Hsu, 2007; Tierney et al., 2017; Voß, 2012; Wang et al., 2015; 2017; Zhang et al., 2015) does not consider retrieval operations. It aims to rearrange blocks into a desirable configuration only by relocations. Here, desirable means that no relocations are necessary for retrieving blocks. A similar problem is studied under the name of Blocks World (Gupta and Nau, 1992; Slaney and Thiébaux, 2001).

1.4. Purpose of this study

Among the existing studies for the BRP, exact approaches are summarized in Table 1. For the unrestricted BRP with distinct priorities, the ILP and MILP approaches in Caserta et al. (2012); Petering and Hussein (2013) are not competitive with dedicated exact algorithms (Expósito-Izquierdo et al., 2014; Zhu et al., 2012). In Petering and Hussein (2013), instances with up to only 9 blocks were solved to optimality, while instances with more than 30 blocks were considered in Expósito-Izquierdo et al. (2014); Zhu et al. (2012). For the restricted BRP, the branch-and-bound algorithm in Tanaka and Takii (2016) seems to be the fastest so far.

The primary purpose of this study is to construct an efficient exact algorithm for the unrestricted BRP with distinct priorities. We also improve our previous exact algorithm (Tanaka and Takii, 2016) for the restricted BRP with distinct priorities. The framework of the algorithm is the same as those in Tanaka and Takii (2016); Zhu et al. (2012). Although Zhu et al. (2012) called their algorithm an (iterative deepening) A* algorithm, we prefer a branch-and-bound algorithm for describing ours because it traverses the search tree not in best-first but depth-first order. To reduce the size of the search tree in this branch-and-bound algorithm, we first propose several dominance properties. These domi-

Table 1
Exact approaches for the BRP in the literature.

Reference	Method	Restricted		Unrestricted	
		Distinct	Duplicate	Distinct	Duplicate
Kim and Hong (2006)	b&b	✓	✓		
Wan et al. (2009)	ILP	✓			
Ünlüyurt and Aydın (2012)	b&b	✓			
Caserta et al. (2012)	ILP	✓		✓	
Zhu et al. (2012)	IDA*	✓		✓	
Petering and Hussein (2013)	MILP			✓	
Zehendner and Feillet (2014)	b&p	✓			
Expósito-Izquierdo et al. (2014)	A*, ILP	✓		✓	
Expósito-Izquierdo et al. (2015)	b&b	✓			
Zehendner et al. (2015)	ILP	✓			
Eskandari and Azari (2015)	ILP	✓			
Tanaka and Takii (2016)	b&b	✓	✓		
Ku and Arthanari (2016b)	b&b	✓			
Tricoire et al. (2018)	b&b			✓	
de Melo et al. (2017)	b&b		✓[a]		
Galle et al. (2017)	ILP	✓			
This study	b&b	✓		✓	

[a] BRP with only two priority values.

nance properties tell us that some partial sequences of relocations never appear in an optimal sequence, or, at least an optimal sequence without them exists. Therefore, they enable us to eliminate unnecessary nodes in the search tree. For the pre-marshalling problem, dominance properties have already been proposed and employed (Tierney et al., 2017; Zhang et al., 2015). However, they are not applicable to the BRP directly because the pre-marshalling problem does not take into account retrieval of blocks. This fact motivated us to develop novel dominance properties specific to the BRP. Next, we propose a new lower bound of the total number of relocations based on the lower bound by Forster and Bortfeldt (2012). It also contributes to improving the efficiency of the branch-and-bound algorithm. The effectiveness of the proposed algorithm is demonstrated by computational experiments for benchmark instances in the literature.

The remainder of this paper is organized as follows. First, notation and definitions used throughout this paper are provided in Section 2. Next, an outline of the exact algorithm is presented in Section 3. Then, the dominance properties are derived in Section 4 to eliminate unnecessary nodes in the search tree of the branch-and-bound algorithm. How to ensure their consistency is discussed in Section 5. The new lower bound is proposed in Section 6. In Section 7, the effectiveness of the proposed algorithm is demonstrated by computational experiments. Finally, the results obtained in this study and future research directions are described in Section 8.

2. Notation and definitions

In this section we introduce the notation and definitions used throughout this paper. They are summarized in Table 2.

The position of block i in a configuration $\mathscr{C}$ is denoted by $(S_i(\mathscr{C}),\ T_i(\mathscr{C}))$ when it is placed in the $T_i(\mathscr{C})$th tier of stack $S_i(\mathscr{C})$. The total number of blocks in stack s and the total number of blocks in the bay are denoted by $N_s(\mathscr{C})$ and $N(\mathscr{C})(:=\sum_{j=1}^{S} N_j(\mathscr{S}))$, respectively. The set of indices of stacks is denoted by $\mathscr{S}(:=\{1,2,\ldots,S\})$. The set of indices of slack stacks whose height is lower than T is denoted by $\mathscr{S}^{\mathrm{slack}}(\mathscr{C})(:=\{s\in\mathscr{S}\mid N_s(\mathscr{C})<T\})$. The priority of the block in the tth tier of stack s is denoted by $P_{st}(\mathscr{C})$. Since it is assumed that block i has priority i, we have $(S_{P_{st}(\mathscr{C})}(\mathscr{C}), T_{P_{st}(\mathscr{C})}(\mathscr{C})) = (s,t)$. The priority of the topmost block of stack s is denoted by $P_s^{\mathrm{top}}(\mathscr{C})(:=P_{sN_s(\mathscr{C})}(\mathscr{C}))$. The highest (smallest) priority in stack s is referred to as the priority of stack k and denoted by $Q_s(\mathscr{C})(:=\min_{1\le t\le N_s(\mathscr{C})} P_{st}(\mathscr{C}))$. If stack s is empty, $Q_s(\mathscr{C})$ is defined by $Q_s(\mathscr{C}):=\infty$. The minimum number of relocations necessary for retrieving all blocks from $\mathscr{C}$ is denoted by $f^*(\mathscr{C})$. As long as the configuration $\mathscr{C}$ currently considered is obvious, $(\mathscr{C})$ is omitted in the above notations: S_i is used in place of $S_i(\mathscr{C})$, N_s in place of $N_s(\mathscr{C})$, and so on.

An operation of relocating block i from stack s to stack $d\ (\neq s)$ is denoted by a triplet $(i,\ s,\ d)$. It is also denoted by a mapping L_{isd} defined over a set of configurations: The new configuration obtained by applying relocation $(i,\ s,\ d)$ to $\mathscr{C}$ is given by $L_{isd}(\mathscr{C})$. The new configuration obtained by removing all retrievable blocks one by one from $\mathscr{C}$ is denoted using a mapping R as $R(\mathscr{C})$. Configuration $\mathscr{C}$ is said to be minimal when no blocks can be retrieved from $\mathscr{C}$, that is, $\mathscr{C}=R(\mathscr{C})$.

The block with the highest (smallest) priority in the bay is referred to as the target block. The target block is the block to be retrieved next. Block i is said to be a blocking block if $i>\min_{1\le t<T_i} P_{S_i t}$, that is, a block with a higher (smaller) priority is placed under it. According to Forster and Bortfeldt (2012), a relocation is classified into BB, BG, GB, and GG relocations. The first B or G means whether the relocated block is a blocking block (B) or not (G), and the second B or G means whether it becomes a blocking block after relocation (B) or not (G).

If block i is present in $\mathscr{C}$, we write $i\in\mathscr{C}$. Relocation $(i,\ s,\ d)$ is feasible for a minimal $\mathscr{C}$ if $i\in\mathscr{C}$, $S_i(\mathscr{C})=s$, $T_i(\mathscr{C})=N_s(\mathscr{C})$, and $N_d(\mathscr{C})<T$ are all satisfied. Relocation $(i,\ s,\ d)$ is admissible for a minimal $\mathscr{C}$ if either it is feasible for $\mathscr{C}$ or $i\notin\mathscr{C}$ holds. For ease of notation, $L_{isd}(\mathscr{C})$ is defined by $L_{isd}(\mathscr{C}):=\mathscr{C}$ for $i\notin\mathscr{C}$. Relocation $(i,\ s,\ d)$ is infeasible for a minimal $\mathscr{C}$ when it is not admissible for $\mathscr{C}$.

A (partial) solution of the BRP is given by a sequence of relocations $(i_1,\ s_1,\ d_1)$, $(i_2,\ s_2,\ d_2)$, $\ldots$, $(i_n,\ s_n,\ d_n)$. We say that this sequence is feasible for a minimal $\mathscr{C}$ if, for any $k=1,\ldots,n$, relocation $(i_k,\ s_k,\ d_k)$ is feasible for configuration $\mathscr{C}_{k-1}$ where $\mathscr{C}_k:=R\circ L_{i_k s_k d_k}(\mathscr{C}_{k-1})$ and $\mathscr{C}_0:=\mathscr{C}$. Similarly, it is admissible if, for any $k=1,\ \ldots,\ n$, relocation $(i_k,\ s_k,\ d_k)$ is admissible for configuration $\mathscr{C}_{k-1}$. In addition, it is infeasible if it is not admissible.

A configuration $\mathscr{C}_{\mathrm{a}}$ dominates a configuration $\mathscr{C}_{\mathrm{b}}$ or $\mathscr{C}_{\mathrm{a}}\le\mathscr{C}_{\mathrm{b}}$, if both the following conditions are satisfied:

(i) $i\in\mathscr{C}_{\mathrm{b}}$ and $S_i(\mathscr{C}_{\mathrm{b}})=S_i(\mathscr{C}_{\mathrm{a}})$ hold for any $i\in\mathscr{C}_{\mathrm{a}}$,
(ii) $T_i(\mathscr{C}_{\mathrm{b}})<T_j(\mathscr{C}_{\mathrm{b}})$ holds for any $i,j\in\mathscr{C}_{\mathrm{a}}$ satisfying $S_i(\mathscr{C}_{\mathrm{a}})=S_j(\mathscr{C}_{\mathrm{a}})$ and $T_i(\mathscr{C}_{\mathrm{a}})<T_j(\mathscr{C}_{\mathrm{a}})$.

This definition implies that if $\mathscr{C}_{\mathrm{a}}\le\mathscr{C}_{\mathrm{b}}$ holds, $\mathscr{C}_{\mathrm{a}}$ is the same configuration as $\mathscr{C}_{\mathrm{b}}$ except for blocks i with $i\notin\mathscr{C}_{\mathrm{a}}$ and $i\in\mathscr{C}_{\mathrm{b}}$. Fig. 4 il-

Table 2
Notation and definitions.

$(S_i(\mathscr{C}), T_i(\mathscr{C}))$	The position of block i in $\mathscr{C}$. Block i is stored in the $T_i(\mathscr{C})$th tier of stack $S_i(\mathscr{C})$.
$N_s(\mathscr{C})$	The number of blocks in stack s in $\mathscr{C}$.
$N(\mathscr{C})$	The total number of blocks in the bay in $\mathscr{C}$. $N(\mathscr{C}) := \sum_{j=1}^{S} N_j(\mathscr{C})$.
$\mathscr{S}$	The set of indices of stacks. $\mathscr{S} := \{1, 2, \ldots, S\}$.
$\mathscr{S}^{\text{slack}}(\mathscr{C})$	The set of indices of slack stacks in $\mathscr{C}$. $\mathscr{S}^{\text{slack}}(\mathscr{C}) := \{s \in \mathscr{S} \mid N_s(\mathscr{C}) < T\}$. $P_{st}(\mathscr{C})$
	The priority of the block in the tth tier of stack s in $\mathscr{C}$. $(S_{P_{st}(\mathscr{C})}(\mathscr{C}), T_{P_{st}(\mathscr{C})}(\mathscr{C})) = (s, t)$.
$P_s^{\text{top}}(\mathscr{C})$	The priority of the block on the top of stack s in $\mathscr{C}$. $P_s^{\text{top}}(\mathscr{C}) := P_{sN_s(\mathscr{C})}(\mathscr{C})$.
$Q_s(\mathscr{C})$	The priority of stack s in $\mathscr{C}$. $Q_s(\mathscr{C}) := \min_{1 \le t \le N_s(\mathscr{C})} P_{st}(\mathscr{C})$ if $N_s(\mathscr{C}) > 0$. Otherwise, $Q_s(\mathscr{C}) := \infty$.
$f^*(\mathscr{C})$	The minimum number of relocations necessary for retrieving all blocks from $\mathscr{C}$. The optimal value for $\mathscr{C}$.
(i, s, d)	The relocation that relocates block i from stack s to stack d.
L_{isd}	The relocation mapping. The configuration obtained by applying (i, s, d) to $\mathscr{C}$ is given by $L_{isd}(\mathscr{C})$.
R	The retrieval mapping. The configuration obtained by removing all retrievable blocks from $\mathscr{C}$ is given by $R(\mathscr{C})$.
target block	The block to be retrieved next. The block with the highest (smallest) priority in the bay.
blocking block	A block under which a block with a higher (smaller) priority is placed. Block i is a blocking block if $i > \min_{1 \le t < T_i(\mathscr{C})} P_{S_i(\mathscr{C})t}(\mathscr{C})$.
BB relocation	A relocation (i, s, d) such that $i > Q_s(\mathscr{C}) \wedge i > Q_d(\mathscr{C})$.
BG relocation	A relocation (i, s, d) such that $i > Q_s(\mathscr{C}) \wedge i < Q_d(\mathscr{C})$.
GB relocation	A relocation (i, s, d) such that $i < Q_s(\mathscr{C}) \wedge i > Q_d(\mathscr{C})$.
GG relocation	A relocation (i, s, d) such that $i < Q_s(\mathscr{C}) \wedge i < Q_d(\mathscr{C})$.
$i \in \mathscr{C}$	Block i is present in $\mathscr{C}$.
$\mathscr{C}_a \le \mathscr{C}_b$	$\mathscr{C}_a$ dominates $\mathscr{C}_b$.

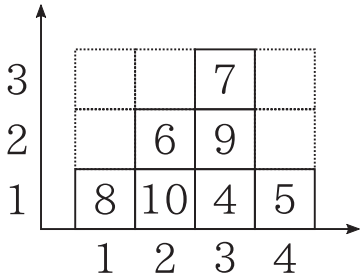


Fig. 4. A configuration dominating the configuration in Fig. 2. Blocks 1–3 are missing, but the other blocks are placed as in Fig. 2.

lustrates an example of a configuration dominating the configuration in Fig. 2.

First, we will give the following two lemmas.

Lemma 1. *Consider two configurations $\mathscr{C}_a$ and $\mathscr{C}_b$ satisfying $\mathscr{C}_a \le \mathscr{C}_b$. Then, $R(\mathscr{C}_a) \le R(\mathscr{C}_b)$ holds.*

Proof. Obvious. □

Lemma 2. *Consider two minimal configurations $\mathscr{C}_a$ and $\mathscr{C}_b$ satisfying $\mathscr{C}_a \le \mathscr{C}_b$. If a relocation (i, s, d) is feasible for $\mathscr{C}_b$, it is admissible for $\mathscr{C}_a$ and $L_{isd}(\mathscr{C}_a) \le L_{isd}(\mathscr{C}_b)$ holds.*

Proof. If $i \in \mathscr{C}_a$, block i should be on the top of stack s in $\mathscr{C}_a$. Otherwise, there exists a block j that satisfies $S_j(\mathscr{C}_a) = s$ and $T_i(\mathscr{C}_a) < T_j(\mathscr{C}_a)$. This block should also be above block i in stack s in $\mathscr{C}_b$ from the definition of $\mathscr{C}_a \le \mathscr{C}_b$, so that relocation (i, s, d) is infeasible for $\mathscr{C}_b$. However, it contradicts the feasibility of relocation (i, s, d) for $\mathscr{C}_b$. Thus $S_i(\mathscr{C}_a) = s$ and $T_i(\mathscr{C}_a) = N_s(\mathscr{C}_a)$ hold if $i \in \mathscr{C}_a$. Furthermore, $N_d(\mathscr{C}_a) \le N_d(\mathscr{C}_b) < T$ holds from $\mathscr{C}_a \le \mathscr{C}_b$ and the feasibility of relocation (i, s, d) for $\mathscr{C}_b$. Therefore, relocation (i, s, d) is feasible for $\mathscr{C}_a$ if $i \in \mathscr{C}_a$, and $L_{isd}(\mathscr{C}_a) \le L_{isd}(\mathscr{C}_b)$ holds in this case. If $i \notin \mathscr{C}_a$, relocation (i, s, d) is admissible for $\mathscr{C}_a$ and $L_{isd}(\mathscr{C}_a) = \mathscr{C}_a$ from the definition of L_{isd}. Thus, $L_{isd}(\mathscr{C}_a) \le L_{isd}(\mathscr{C}_b)$ holds also in this case. □

Corollary 1. *$f^*(\mathscr{C}_a) \le f^*(\mathscr{C}_b)$ holds if two minimal configurations $\mathscr{C}_a$ and $\mathscr{C}_b$ satisfy $\mathscr{C}_a \le \mathscr{C}_b$.*

Proof. From Lemmas 1 to 2, an optimal sequence for $\mathscr{C}_b$ is admissible for $\mathscr{C}_a$. □

Fig. 5 confirms that the optimal sequence (3, 1, 4), (7, 3, 1), (6, 2, 1), (9, 3, 2) in Fig. 3 for the configuration in Fig. 2 ($\mathscr{C}_b$) is admissible for the configuration in Fig. 4 ($\mathscr{C}_a$), which dominates $\mathscr{C}_b$. We note that every intermediate configuration in Fig. 5 dominates the corresponding configuration in Fig. 3 as Lemmas 1 and 2 claim. In this case, a sequence (7, 3, 1), (6, 2, 1), (9, 3, 2) with only three relocations is feasible for $\mathscr{C}_a$ and $f^*(\mathscr{C}_a) < f^*(\mathscr{C}_b)$ holds.

Suppose that a feasible sequence of n_b relocations for a minimal $\mathscr{C}$ yields $\mathscr{C}_b$ and there exists another admissible sequence of n_a relocations for $\mathscr{C}$ that yields $\mathscr{C}_a$. If $n_a \le n_b$ and $\mathscr{C}_a \le \mathscr{C}_b$, $f^*(\mathscr{C}) \le n_a + f^*(\mathscr{C}_a) \le n_b + f^*(\mathscr{C}_b)$ holds from Corollary 1. Thus the former sequence is not better than the latter, and we say that the former sequence is dominated by the latter sequence, or, the latter dominates the former. If $n_a < n_b$, the former sequence is never optimal. In this case, we say that the former is strictly dominated by the latter.

3. Exact algorithm

In this section we describe an outline of the proposed exact algorithm.

3.1. Outline of algorithm

The framework is basically the same as those in Tanaka and Takii (2016); Zhu et al. (2012). The main loop of the exact algorithm is described as Algorithm 1. It first computes the initial

Algorithm 1 Exact algorithm.

```
1: procedure SOLVE(𝒞)
2:     𝒞 ← R(𝒞)
3:     LB ← LOWERBOUND(𝒞)
4:     𝒳 ← HEURISTIC(𝒞)
5:     UB^tent ← LB
6:     while UB^tent < |𝒳| do
7:         𝒳 ← BRANCHANDBOUND(UB^tent, 𝒳, ∅, 𝒞)
8:         UB^tent ← UB^tent + 1
9:     return 𝒳
```

lower bound LB (line 3), and next initializes the current best solution $\mathscr{X}$ by a greedy heuristic explained later (line 4). Then, the tentative upper bound UB^{tent} is initialized by LB (line 5), and the algorithm searches for a solution whose objective value is equal to UB^{tent} by a branch-and-bound algorithm (line 7). If such a solution is found, the algorithm returns $\mathscr{X}$ as an optimal solution and terminates (lines 6 and 9). Otherwise, the optimal value is greater than UB^{tent}. Hence, UB^{tent} is increased by one (line 8) and the branch-and-bound algorithm is applied again.

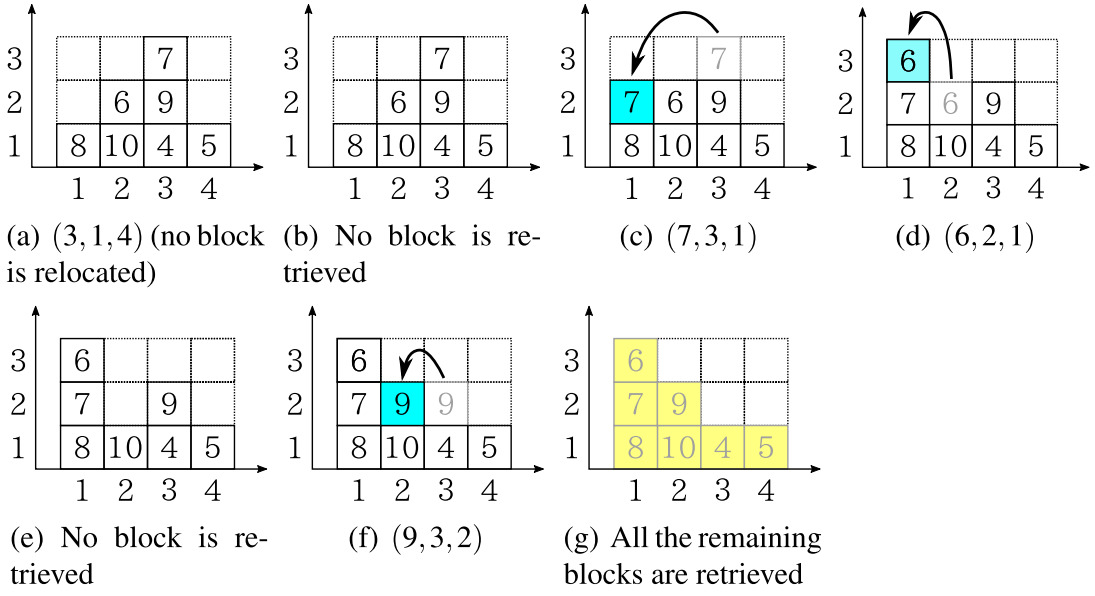


Fig. 5. Sequence (3, 1, 4), (7, 3, 1), (6, 2, 1), (9, 3, 2) in Fig. 3 applied to the configuration in Fig. 4.

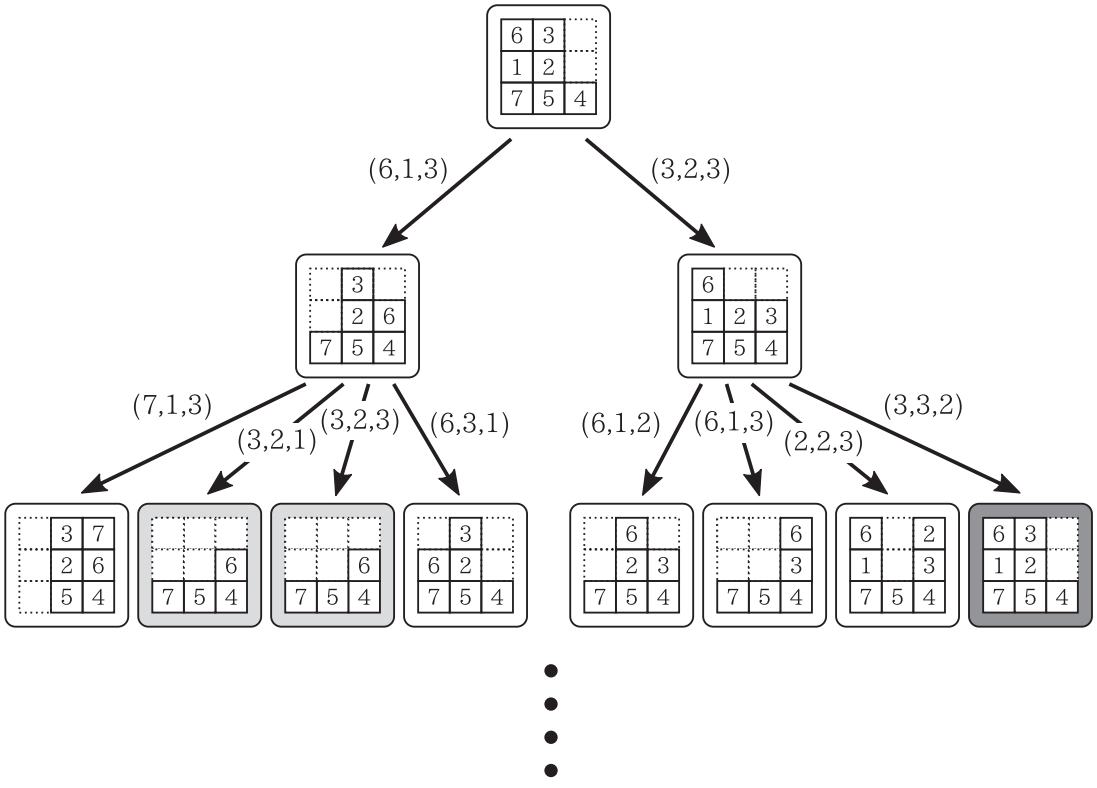


Fig. 6. An example of the search tree in the branch-and-bound algorithm.

In the branch-and-bound algorithm, an optimal sequence of relocations is searched for from first to last. Thus, a node at depth k in the search tree represents a configuration reached by k relocations from the initial configuration. An example of the search tree is provided in Fig. 6. We note that the same configuration as the initial one is obtained by sequence (3, 2, 3), (3, 3, 2): relocating block 3 from stack 2 to stack 3 and then stack 3 to stack 2. Obviously, such a (partial) sequence is never optimal. In other words, sequence (3, 2, 3), (3, 3, 2) is strictly dominated, and we can eliminate the node. We also note that two sequences (6, 1, 3), (3, 2, 1) and (6, 1, 3), (3, 2, 3) (sequences (3, 2, 1) and (3, 2, 3) in practice) yield the same configuration. In this case, each of the two sequences is dominated by the other, and we can eliminate either node, but not both. We will first derive several dominance properties in Section 4 to detect such a dominated sequence and eliminate the corresponding node. Since some of these properties only tell us whether a sequence is dominated or not, we will explain in Section 5 which of the two sequences (3, 2, 1) and (3, 2, 3) is adopted as well as how to ensure consistency among the dominance properties.

In the branch-and-bound algorithm, Algorithm 2 is called recursively, where $\mathscr{X}^{*}$ is the current best sequence of relocations, $\mathscr{X}^{\text{cur}}$

Algorithm 2 Branch-and-bound algorithm.

```
1: procedure BranchAndBound(UB^tent, 𝒳, 𝒳^cur, 𝒞^cur)
2:     𝒜 ← ∅
3:     for all feasible relocations (i, s, d) do
4:         𝒳^child ← 𝒳^cur ∪ {(i, s, d)}
5:         if 𝒳^child is not dominated then
6:             𝒞^child ← R ∘ L_isd(𝒞^cur)
7:             if N(𝒞^child) = 0 then
8:                 return 𝒳^child
9:             LB^child ← LowerBound(𝒞^child)
10:            if |𝒳^child| + LB^child = UB^tent − 1 then
11:                𝒳^heur ← Heuristic(𝒞^child)
12:                if |𝒳^child| + |𝒳^heur| < |𝒳| then
13:                    𝒳 ← 𝒳^child ∪ 𝒳^heur
14:                    if |𝒳| = UB^tent then
15:                        return 𝒳
16:            if |𝒳^child| + LB^child ≤ UB^tent then
17:                𝒜 ← 𝒜 ∪ {(LB^child, 𝒳^child, 𝒞^child)}
18:    Sort the elements of 𝒜 in the nondecreasing order of the first key
19:    for all (LB^child, 𝒳^child, 𝒞^child) ∈ 𝒜 do
20:        𝒳 ← BranchAndBound(UB^tent, 𝒳, 𝒳^child, 𝒞^child)
21:        if |𝒳| = UB^tent then
22:            return 𝒳
23:    return 𝒳
```

the partial sequence of relocations, and $\mathscr{C}^{\text{cur}}$ the configuration obtained by $\mathscr{X}^{\text{cur}}$. To branch from the current node, all feasible relocations for $\mathscr{C}^{\text{cur}}$ are generated (line 3). Each of them is appended to $\mathscr{X}^{\text{cur}}$ (line 4) in order to obtain a sequence $\mathscr{X}^{\text{child}}$ for a child node. Then, whether it is dominated or not is checked (line 5). More specifically, the existence of a dominating sequence for every subsequence of $\mathscr{X}^{\text{child}}$ ending at its last element is checked to see whether $\mathscr{X}^{\text{child}}$ can be eliminated or not. Unless $\mathscr{X}^{\text{child}}$ is dominated, the corresponding configuration $\mathscr{C}^{\text{child}}$ is also generated (line 6). If no block is left in $\mathscr{C}^{\text{child}}$ (line 7), $\mathscr{X}^{\text{child}}$ is a feasible solution, so that the algorithm returns it (line 8). Otherwise, a lower bound is computed (line 9). Only child nodes whose lower bound is less than the current upper bound are added to list $\mathscr{A}$ (lines 16–17). To improve the current best solution, the greedy heuristic in Section 3.2 is applied to $\mathscr{C}^{\text{child}}$ when $|\mathscr{X}^{\text{child}}| + \text{LB}^{\text{child}} = \text{UB}^{\text{tent}} - 1$ (lines 10–15). This condition aims to:

1. filter out unpromising child nodes with a large lower bound,
2. avoid applying the heuristic to the same node more than once.

Suppose that the heuristic is applied to the nodes whose lower bound is small enough to satisfy $|\mathscr{X}^{\text{child}}| + \text{LB}^{\text{child}} \le \text{UB}^{\text{tent}} - K$ for some fixed integer K. Because the branch-and-bound algorithm is repeatedly employed with UB^{tent} increased, the heuristic should have already been applied to the same nodes at a previous iteration if $|\mathscr{X}^{\text{child}}| + \text{LB}^{\text{child}} \le (\text{UB}^{\text{tent}} - 1) - K$. It follows that we only need to consider nodes with $|\mathscr{X}^{\text{child}}| + \text{LB}^{\text{child}} = \text{UB}^{\text{tent}} - K$, and K is chosen as 1 in the algorithm. After child nodes at depth $(k+1)$ are generated, they are traversed one by one (lines 19–22). Thus the depth-first strategy is adopted in this branch-and-bound algorithm. Among the nodes at depth $(k+1)$, the one with the smallest lower bound is traversed first (line 8). When a solution whose objective value is equal to UB^{tent} is found, the algorithm immediately terminates (lines 14 and 15, and lines 21 and 22).

3.2. Greedy heuristic

The heuristic used for computing upper bounds is a slightly simplified version of PU2 in Zhu et al. (2012). This heuristic determines the source and destination stacks of the next relocation as follows:

1. Determine the source stack s. Let $s^{\text{T}} := \arg\min_{j \in \mathscr{S}} Q_j$ (the target block is placed in stack s^{T}).
   (a) If a BG relocation exists for the topmost block $P_{s^{\text{T}}}^{\text{top}}$ of stack s^{T}, $s := s^{\text{T}}$.
   (b) If no BG relocation exists for block $P_{s^{\text{T}}}^{\text{top}}$, but a GG relocation of another topmost block $P_{s'}^{\text{top}}$ enables it, $s := s'$. If there is more than one candidate for stack s', the stack with the largest $N_{s'}$ is chosen.
   (c) Otherwise, $s := s^{\text{T}}$.
2. Determine the destination stack d.
   (a) If a BG or GG relocation exists for block P_s^{top}, the stack d with the highest (smallest) stack priority Q_d is chosen among those satisfying $P_s^{\text{top}} < Q_d$.
   (b) If no BG or GG relocation exists for block P_s^{top}, let stacks d' and d'' be the slack stacks with the lowest (largest) and the second lowest (second largest) stack priorities, respectively.
      (i) If $N_{d'} = T - 1$ and stack d'' does exist, $d := d''$.
      (ii) Otherwise, $d := d'$.

In this procedure, 2(a) chooses the destination stack that minimizes the change of the stack priority. On the other hand, 2(b) chooses the stack that possibly delays future relocations of block P_s^{top}. 2(b)(i) tries not to fill stack d' in order to accept a BG or GG relocation in the future.

Algorithm 3 describes the heuristic in more detail. The source

Algorithm 3 Greedy heuristic.

```
1: procedure Heuristic(𝒞)
2:     𝒳 ← ∅
3:     while N(𝒞) > 0 do
4:         s^T ← arg min_{j∈𝒮} Q_j(𝒞)
5:         s^max ← arg max_{j∈𝒮^slack(𝒞)} Q_j(𝒞)
6:         s ← s^T
7:         i ← P_s^top(𝒞)
8:         if i > Q_{s^max}(𝒞) then
9:             ℬ ← ∅
10:            for all j ∈ 𝒮 \ {s^T} do
11:                if i < min_{1≤k≤N_j(𝒞)−1} P_jk(𝒞) ∧ P_j^top(𝒞) < Q_{s^max}(𝒞) then
12:                    ℬ ← ℬ ∪ {j}
13:            if ℬ ≠ ∅ then
14:                s ← arg max_{j∈ℬ} N_j(𝒞)
15:                i ← P_s^top(𝒞)
16:        if i < Q_{s^max}(𝒞) then
17:            d ← arg min_{j∈𝒮^slack(𝒞) ∧ Q_j(𝒞)>i} Q_j(𝒞)
18:        else if N_{s^max}(𝒞) < T − 1 ∨ |𝒮^slack(𝒞)| = 1 then
19:            d ← s^max
20:        else
21:            d ← arg max_{j∈𝒮^slack(𝒞)\{s^max}} Q_j(𝒞)
22:        𝒳 ← 𝒳 ∪ {(i, s, d)}
23:        𝒞 ← R ∘ L_isd(𝒞)
24:    return 𝒳
```

stack and thus the block to be relocated is determined on lines 6–15, and its destination stack is determined on lines 16–21. The algorithm first tries to relocate $P_{s^{\text{T}}}^{\text{top}}$ (lines 6 and 7). If $P_{s^{\text{T}}}^{\text{top}} > Q_{s^{\max}} = \max_{j \in \mathscr{S}^{\text{slack}}} Q_j$ holds (line 8), no BG relocation exists for this block. In this case, the algorithm checks whether a GG relocation of the topmost block in a stack $j \neq s^{\text{T}}$ enables it (lines 9–12). If such a block is found, it is relocated where ties are broken by the stack

height (lines 13–15). Otherwise, block $P_{s^T}^{top}$ is relocated. Next, the destination stack of the block (block i) is determined. If a BG or GG relocation exists for this block (line 16), the stack with the highest (smallest) priority is chosen, as long as the relocation remains BG or GG, respectively (line 17). If no BG or GG relocation exists, the stack with the lowest (largest) priority is chosen, as long as this relocation does not make the stack full (lines 18–19). If it makes that stack full, the stack with the second lowest (largest) priority is chosen (line 21).

4. Dominance properties

As presented in the preceding section, the proposed branch-and-bound algorithm checks dominance properties (line 5 in Algorithm 2) to suppress generation of unnecessary nodes in the search tree. A simple example is sequence (3, 2, 3), (3, 3, 2) in Fig. 6, which relocates block 3 from stack 2 to stack 3 and immediately relocates it back to stack 2. Because the same configuration as the original one is generated, we need not consider this sequence in the search tree. In other words, this sequence is dominated. Some dominance properties have already been proposed for the pre-marshalling problem (Tierney et al., 2017; Zhang et al., 2015). However, they are not applicable directly to our problem due to existence of block retrieval. Indeed, the above example is not always true for the BRP: Sequence (6, 1, 3), (6, 3, 1) is not dominated in Fig. 6 because block 1 is retrieved after relocation (6, 1, 3). In this section, we propose several dominance properties specific to the unrestricted BRP that take into account retrieval of blocks appropriately. Some of them are also applicable to the restricted BRP, and we employ them to improve our previous algorithm (Tanaka and Takii, 2016).

The dominance properties are classified into three types. Theorems 1–3 concern a pair of transitive relocations. Theorems 1 and 2 assert that a pair of relocations from stack A to stack B and from stack B to stack C can be combined into one relocation from stack A to stack C, whereas Theorem 3 claims that the pair can be replaced by another pair from stack A to stack D and from stack D to stack C. Theorems 4 and 5 are for a pair of independent relocations. If block X is relocated from stack A to stack B and then block Y is relocated from stack C to stack D, their order can be interchanged. The last two theorems, Theorems 6 and 7 are on retrieval of a block. If block X is retrieved after it is relocated from stack A to stack B, it need not be relocated (Theorem 6), or, it can be relocated to another stack C (Theorem 7). In the following, a feasible sequence $(i_1, s_1, d_1), (i_2, s_2, d_2), \ldots, (i_n, s_n, d_n)$ for a minimal configuration $\mathscr{C}$ is assumed, and $\mathscr{C}_k$ $(k = 0, \ldots, n)$ are defined by

$$\mathscr{C}_k := \begin{cases} R \circ L_{i_k s_k d_k}(\mathscr{C}_{k-1}), & 1 \le k \le n, \\ \mathscr{C}, & k = 0. \end{cases} \tag{1}$$

Our goal is to provide conditions for the existence of a sequence dominating sequence $(i_1, s_1, d_1), (i_2, s_2, d_2), \ldots, (i_n, s_n, d_n)$.

4.1. Dominance properties for transitive relocations

First, three dominance properties on a pair of transitive relocations are provided.

Theorem 1 (Transitive Relocation Rule A). *Sequence* $(i_2, s_2, d_2), \ldots, (i_{n-1}, s_{n-1}, d_{n-1}), (i_1, s_1, d_n)$ *or* $(i_2, s_2, d_2), \ldots, (i_{n-1}, s_{n-1}, d_{n-1})$ *strictly dominates sequence* $(i_1, s_1, d_1), (i_2, s_2, d_2), \ldots, (i_n, s_n, d_n)$ *if all the following conditions are satisfied:*

(TA1) $s_n = d_1,\ i_n = i_1$,
(TA2) $i_1 \notin \{i_2, \ldots, i_{n-1}\}$,
(TA3) $s_1 \notin \{s_2, d_2, \ldots, s_{n-1}, d_{n-1}\}$,
(TA4) $N_{s_1}(\mathscr{C}_{n-1}) = N_{s_1}(\mathscr{C}) - 1$.

Proof. First, suppose that $d_n \neq s_1$. From (TA3) and (TA4), stack s_1 is not affected by sequence $(i_1, s_1, d_1), \ldots, (i_{n-1}, s_{n-1}, d_{n-1})$ for $\mathscr{C}$ after block i_1 is relocated to stack d_1. The configuration of stack s_1 in $\mathscr{C}_1, \ldots, \mathscr{C}_{n-1}$ is the same as that in $\mathscr{C}$ except the absence of block i_1. It follows that block i_1 does not interfere any retrieval caused by this sequence for $\mathscr{C}$, even if it is not relocated to stack d_1. Furthermore, from (TA2), block i_1 is not relocated by sequence $(i_2, s_2, d_2), \ldots, (i_{n-1}, s_{n-1}, d_{n-1})$. Thus this sequence is admissible for $\mathscr{C}$. If the resulting configuration is denoted by $\widehat{\mathscr{C}}_{n-1}$, it satisfies $\widehat{\mathscr{C}}_{n-1} \le L_{i_1 d_1 s_1}(\mathscr{C}_{n-1})$. Therefore, from Lemmas 1 to 2 as well as (TA1), we obtain

$$\begin{aligned} R \circ L_{i_1 s_1 d_n}(\widehat{\mathscr{C}}_{n-1}) &\le R \circ L_{i_1 s_1 d_n} \circ L_{i_1 d_1 s_1}(\mathscr{C}_{n-1}) \\ &= R \circ L_{i_1 d_1 d_n}(\mathscr{C}_{n-1}) = R \circ L_{i_n s_n d_n}(\mathscr{C}_{n-1}) = \mathscr{C}_n. \end{aligned} \tag{2}$$

This inequality implies that sequence $(i_2, s_2, d_2), \ldots, (i_{n-1}, s_{n-1}, d_{n-1}), (i_1, s_1, d_n)$ is admissible for $\mathscr{C}$ and the resulting configuration $R \circ L_{i_1 s_1 d_n}(\widehat{\mathscr{C}}_{n-1})$ dominates $\mathscr{C}_n$.

It is easy to see that the above argument also holds true when $d_n = s_1$. In this case, sequence $(i_2, s_2, d_2), \ldots, (i_{n-1}, s_{n-1}, d_{n-1})$ strictly dominates sequence $(i_1, s_1, d_1), (i_2, s_2, d_2), \ldots, (i_{n-1}, s_{n-1}, d_{n-1}), (i_n, s_n, d_n)$. □

In the example of Fig. 7, sequence (5, 2, 3), (7, 1, 4), (7, 4, 1), (5, 3, 4) satisfies the conditions in Theorem 1, and is dominated by sequence (7, 1, 4), (7, 4, 1), (5, 2, 4). In this case, the dominating sequence is not feasible but admissible. Moreover, the resulting configuration is not identical to that by the dominated sequence.

Theorem 2 (Transitive Relocation Rule B). *Sequence* $(i_1, s_1, d_n), (i_2, s_2, d_2), \ldots, (i_{n-1}, s_{n-1}, d_{n-1})$ *strictly dominates sequence* $(i_1, s_1, d_1), (i_2, s_2, d_2), \ldots, (i_n, s_n, d_n)$ *if all the following conditions are satisfied:*

(TB1) $s_n = d_1,\ i_n = i_1$,
(TB2) $i_1 \notin \{i_2, \ldots, i_{n-1}\}$,
(TB3) $d_n \notin \{s_1, d_1, \ldots, s_{n-1}, d_{n-1}\}$,
(TB4) $N_{d_n}(\mathscr{C}_{n-1}) = N_{d_n}(\mathscr{C})$.

Proof. From (TB4) and the feasibility of relocation (i_n, s_n, d_n) for $\mathscr{C}_{n-1}$, $N_{d_n}(\mathscr{C}) = N_{d_n}(\mathscr{C}_{n-1}) < T$ holds. This inequality ensures the feasibility of relocation (i_1, s_1, d_n) for $\mathscr{C}$, so that $\widehat{\mathscr{C}}_1$ is defined by $\widehat{\mathscr{C}}_1 := R \circ L_{i_1 s_1 d_n}(\mathscr{C})$. Because (TB3) and (TB4) imply that stack d_n is not affected by sequence $(i_1, s_1, d_1), \ldots, (i_{n-1}, s_{n-1}, d_{n-1})$ for $\mathscr{C}$, block i_1 does not interfere any retrieval caused by this sequence for $\mathscr{C}$, even if it is relocated to stack d_n beforehand. Moreover, from (TB2), block i_1 is not relocated by sequence $(i_2, s_2, d_2), \ldots, (i_{n-1}, s_{n-1}, d_{n-1})$. Therefore, this sequence is admissible for $\widehat{\mathscr{C}}_1$. If the resulting configuration is denoted by $\widehat{\mathscr{C}}_{n-1}$, it satisfies $\widehat{\mathscr{C}}_{n-1} \le L_{i_1 d_1 d_n}(\mathscr{C}_{n-1})$. By noting (TB1), we obtain

$$\widehat{\mathscr{C}}_{n-1} = R(\widehat{\mathscr{C}}_{n-1}) \le R \circ L_{i_1 d_1 d_n}(\mathscr{C}_{n-1}) = \mathscr{C}_n. \tag{3}$$

This completes the proof. □

An example of Theorem 2 is presented in Fig. 8, where sequence (6, 2, 3), (8, 1, 2), (6, 3, 4) is dominated by sequence (6, 2, 4), (8, 1, 2).

Theorem 3 (Transitive Relocation Rule C). *Sequence* $(i_1, s_1, d_1'), (i_2, s_2, d_2), \ldots, (i_{n-1}, s_{n-1}, d_{n-1},) (i_1, d_1', d_n)$ *dominates sequence* $(i_1, s_1, d_1), (i_2, s_2, d_2), \ldots, (i_n, s_n, d_n)$ *if all the following conditions are satisfied:*

(TC1) $s_n = d_1,\ i_n = i_1$,
(TC2) $i_1 \notin \{i_2, \ldots, i_{n-1}\}$,
(TC3) $d_1' \notin \{s_1, d_1, s_2, d_2, \ldots, s_{n-1}, d_{n-1}, d_n\}$,
(TC4) $N_{d_1'}(\mathscr{C}_{n-1}) = N_{d_1'}(\mathscr{C}) < T$.

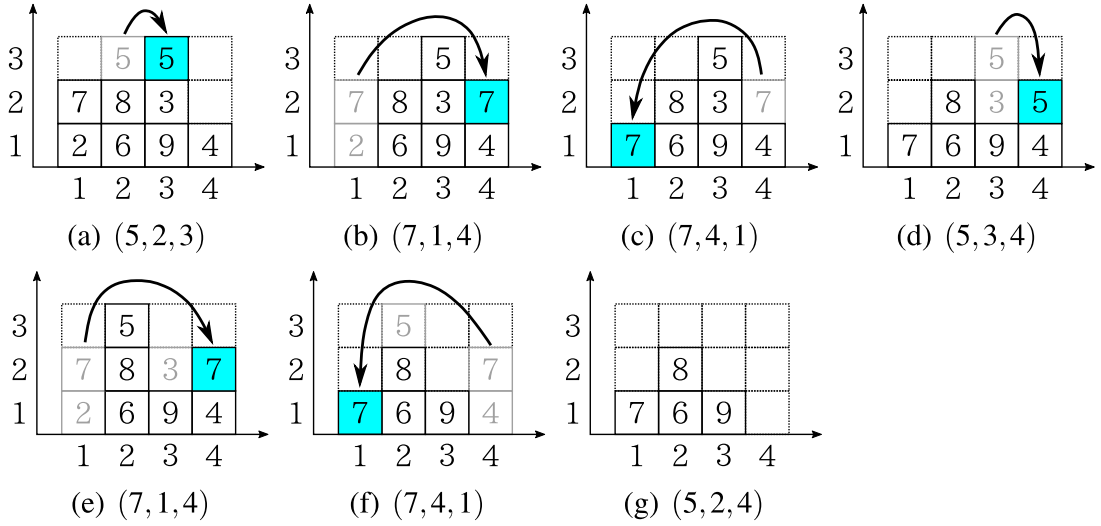


Fig. 7. An example of Theorem 1. (a)–(d): dominated sequence (5, 2, 3), (7, 1, 4), (7, 4, 1), (5, 3, 4), (e)–(g): dominating sequence (7, 1, 4), (7, 4, 1), (5, 2, 4).

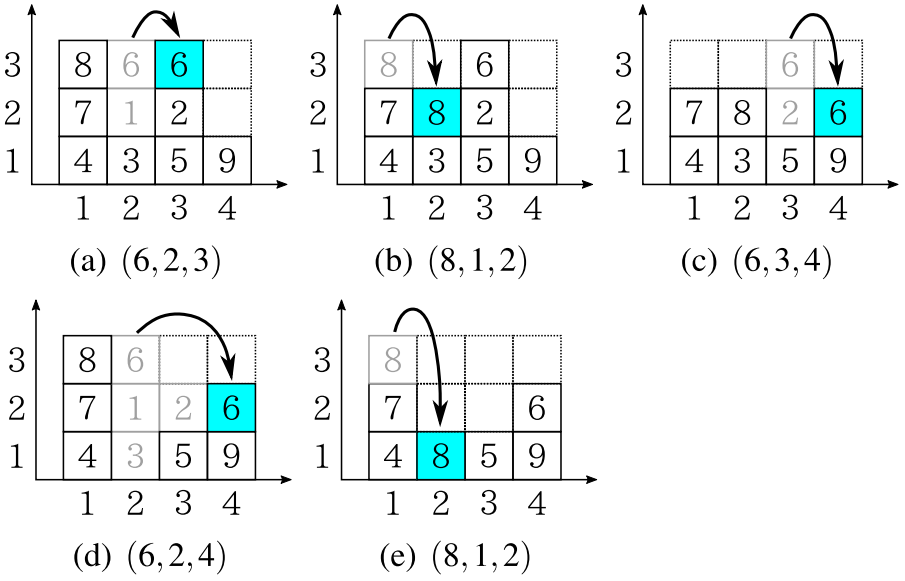


Fig. 8. An example of Theorem 2. (a)–(c): dominated sequence (6, 2, 3), (8, 1, 2), (6, 3, 4), (d) and (e): dominating sequence (6, 2, 4), (8, 1, 2)

Proof. From (TC4), relocation (i_1, s_1, d_1') is feasible for $\mathscr{C}$. Thus let us define $\widehat{\mathscr{C}_1} := R \circ L_{i_1 s_1 d_1'}(\mathscr{C})$. Because (TC3) and (TC4) imply that stack d_1' is not affected by sequence $(i_1, s_1, d_1), \ldots, (i_{n-1}, s_{n-1}, d_{n-1})$ for $\mathscr{C}$, block i_1 does not interfere any retrieval caused by this sequence for $\mathscr{C}$, even if it is relocated to stack d_1' instead of stack d_1. Moreover, from (TC2), block i_1 is not relocated by sequence $(i_2, s_2, d_2), \ldots, (i_{n-1}, s_{n-1}, d_{n-1})$. Therefore, this sequence is admissible for $\widehat{\mathscr{C}_1}$. If the resulting configuration is denoted by by $\widehat{\mathscr{C}}_{n-1}$, it satisfies $\widehat{\mathscr{C}}_{n-1} \le L_{i_1 d_1 d_1'}(\mathscr{C}_{n-1})$. By noting (TC1), we obtain

$$R \circ L_{i_1 d_1' d_n}(\widehat{\mathscr{C}}_{n-1}) \le R \circ L_{i_1 d_1' d_n} \circ L_{i_1 d_1 d_1'}(\mathscr{C}_{n-1}) = R \circ L_{i_1 d_1 d_n}(\mathscr{C}_{n-1}) = \mathscr{C}_n. \tag{4}$$

It follows that sequence (i_1, s_1, d_1'), (i_2, s_2, d_2), $\ldots$, $(i_{n-1}, s_{n-1}, d_{n-1})$, (i_1, d_1', d_n) is admissible for $\mathscr{C}$, and the resulting configuration $R \circ L_{i_1 d_1' d_n}(\widehat{\mathscr{C}}_{n-1})$ dominates $\mathscr{C}_n$. □

In the example of Fig. 9, sequence (7, 1, 2), (4, 1, 4), (7, 2, 4) is dominated by sequence (7, 1, 3), (4, 1, 4), (7, 3, 4).

4.2. Dominance properties for independent relocations

Next, two dominance properties on a pair of independent relocations are presented.

Theorem 4 (Independent Relocation Rule A). *Sequence* $(i_2, s_2, d_2), \ldots, (i_n, s_n, d_n), (i_1, s_1, d_1)$ *dominates sequence* $(i_1, s_1, d_1), (i_2, s_2, d_2), \ldots, (i_n, s_n, d_n)$ *if all the following conditions are satisfied:*

(IA1) $\{s_1, d_1\} \cap \{s_2, d_2, \ldots, s_n, d_n\} = \emptyset$,
(IA2) $N_{s_1}(\mathscr{C}_{n-1}) = N_{s_1}(\mathscr{C}) - 1$.

Proof. From (IA1) and (IA2), stack s_1 is not affected by sequence $(i_1, s_1, d_1), \ldots, (i_{n-1}, s_{n-1}, d_{n-1})$ for $\mathscr{C}$ after block i_1 is relocated to stack d_1. The configuration of stack s_1 in $\mathscr{C}_1, \ldots, \mathscr{C}_{n-1}$ is the same as that in $\mathscr{C}$, except the absence of block i_1. Therefore, block i_1 does not interfere any retrieval caused by sequence $(i_1, s_1, d_1), \ldots, (i_{n-1}, s_{n-1}, d_{n-1})$ for $\mathscr{C}$, even if it is not relocated to stack d_1. Hence $(i_2, s_2, d_2), \ldots, (i_n, s_n, d_n)$ is admissible for $\mathscr{C}$. Let us denote the resulting configuration by $\widehat{\mathscr{C}}_{n-1}$. If $i_1 \notin \widehat{\mathscr{C}}_{n-1}$, it is obvious that relocation (i_1, s_1, d_1) is admissible for $\widehat{\mathscr{C}}_{n-1}$, and $R \circ L_{i_1 s_1 d_1}(\widehat{\mathscr{C}}_{n-1}) = \widehat{\mathscr{C}}_{n-1} \le \mathscr{C}_n$ holds. If $i_1 \in \widehat{\mathscr{C}}_{n-1}$ and

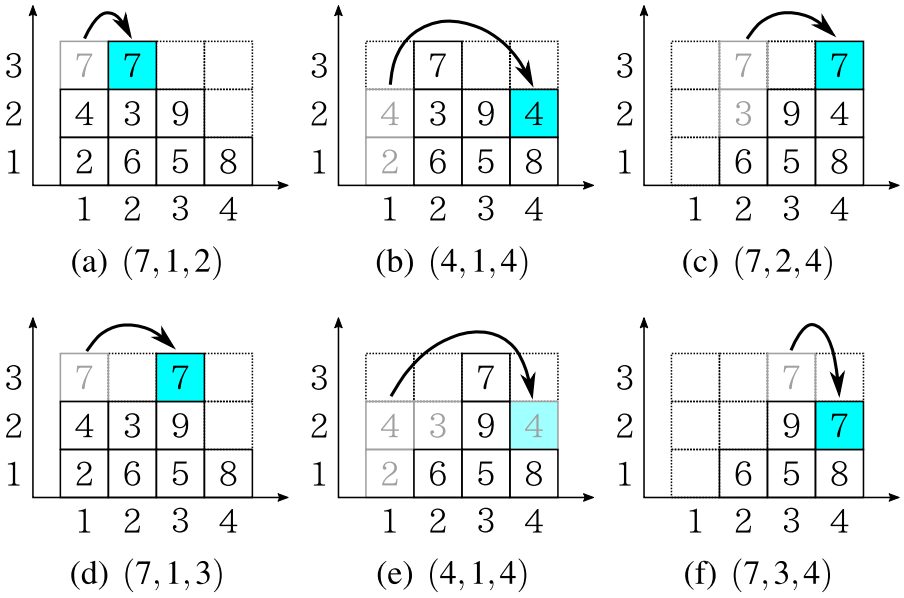


Fig. 9. An example of Theorem 3. (a)–(c): dominated sequence (7, 1, 2), (4, 1, 4), (7, 2, 4), (d)–(f): dominating sequence (7, 1, 3), (4, 1, 4), (7, 3, 4)

$i_1 \in \mathscr{C}_n$, block i_1 is on the top of stack d_1 in $\mathscr{C}_n$ and $N_{d_1}(\mathscr{C}_n) = N_{d_1}(\mathscr{C}) + 1 \leq T$ holds, so that relocation (i_1, s_1, d_1) is feasible for $\widehat{\mathscr{C}}_{n-1}$ from $N_{d_1}(\widehat{\mathscr{C}}_{n-1}) \leq N_{d_1}(\mathscr{C}) = N_{d_1}(\mathscr{C}_n) - 1 < T$. Furthermore, $R \circ L_{i_1 s_1 d_1}(\widehat{\mathscr{C}}_{n-1}) \leq R \circ L_{i_n s_n d_n}(\mathscr{C}_{n-1}) = \mathscr{C}_n$ holds. If $i_1 \in \widehat{\mathscr{C}}_{n-1}$ and $i_1 \notin \mathscr{C}_n$, $i_1 \in \mathscr{C}_{n-1}$ holds; otherwise, block i_1 should also be retrieved by sequence $(i_2, s_2, d_2), \ldots, (i_{n-1}, s_{n-1}, d_{n-1})$ for $\mathscr{C}$ because block i_1 does not interfere any retrieval even if it is not relocated to stack d_1, which contradicts $i_1 \in \widehat{\mathscr{C}}_{n-1}$. Because block i_1 is retrieved by the retrieval from $L_{i_n s_n d_n}(\mathscr{C}_{n-1})$, relocation (i_1, s_1, d_1) is feasible for $\widehat{\mathscr{C}}_{n-1}$ from $N_{d_1}(\widehat{\mathscr{C}}_{n-1}) \leq N_{d_1}(\mathscr{C}) = N_{d_1}(\mathscr{C}_{n-1}) - 1 < T$. Therefore, $R \circ L_{i_1 s_1 d_1}(\widehat{\mathscr{C}}_{n-1}) \leq R \circ L_{i_n s_n d_n}(\mathscr{C}_{n-1}) = \mathscr{C}_n$ holds also in this case. To summarize, $(i_2, s_2, d_2), \ldots, (i_n, s_n, d_n), (i_1, s_1, d_1)$ is admissible for $\mathscr{C}$ and the resulting configuration $R \circ L_{i_1 s_1 d_1}(\widehat{\mathscr{C}}_{n-1})$ dominates $\mathscr{C}_n$. □

Theorem 5 (Independent Relocation Rule B). *Sequence* (i_n, s_n, d_n), $(i_1, s_1, d_1), \ldots, (i_{n-1}, s_{n-1}, d_{n-1})$ *dominates sequence* $(i_1, s_1, d_1), (i_2, s_2, d_2), \ldots, (i_n, s_n, d_n)$ *if all the following conditions are satisfied:*

(IB1) $\{s_n, d_n\} \cap \{s_1, d_1, \ldots, s_{n-1}, d_{n-1}\} = \emptyset$,
(IB2) $N_{s_n}(\mathscr{C}_{n-1}) = N_{s_n}(\mathscr{C})$,
(IB3) $N_{d_n}(\mathscr{C}_{n-1}) = N_{d_n}(\mathscr{C})$.

Proof. From (IB1), (IB2) and (IB3), stacks s_n and d_n are unaffected by sequence $(i_1, s_1, d_1), \ldots, (i_{n-1}, s_{n-1}, d_{n-1})$ for $\mathscr{C}$. Therefore, sequence $(i_n, s_n, d_n), (i_1, s_1, d_1), \ldots, (i_{n-1}, s_{n-1}, d_{n-1})$ is admissible for $\mathscr{C}$ from the feasibility of relocation (i_n, s_n, d_n) for $\mathscr{C}_{n-1}$. If the resulting configuration is denoted by $\widehat{\mathscr{C}}_n$, $\widehat{\mathscr{C}}_n \leq L_{i_n s_n d_n}(\mathscr{C}_{n-1})$ holds, and we obtain

$$\widehat{\mathscr{C}}_n = R(\widehat{\mathscr{C}}_n) \leq R \circ L_{i_n s_n d_n}(\mathscr{C}_{n-1}) = \mathscr{C}_n. \tag{5}$$

This completes the proof. □

Fig. 10 illustrates an example of Theorems 4 and 5. Both the theorems are applicable and sequence (9, 1, 2), (7, 3, 4) is proved to be dominated by sequence (7, 3, 4), (9, 1, 2).

4.3. Dominance properties for retrieval

Finally, two dominance properties on retrieval are proved.

Theorem 6 (Retrieval Rule A). *Sequence* $(i_2, s_2, d_2), \ldots, (i_n, s_n, d_n)$ *strictly dominates sequence* $(i_1, s_1, d_1), (i_2, s_2, d_2), \ldots, (i_n, s_n, d_n)$ *if all the following conditions are satisfied:*

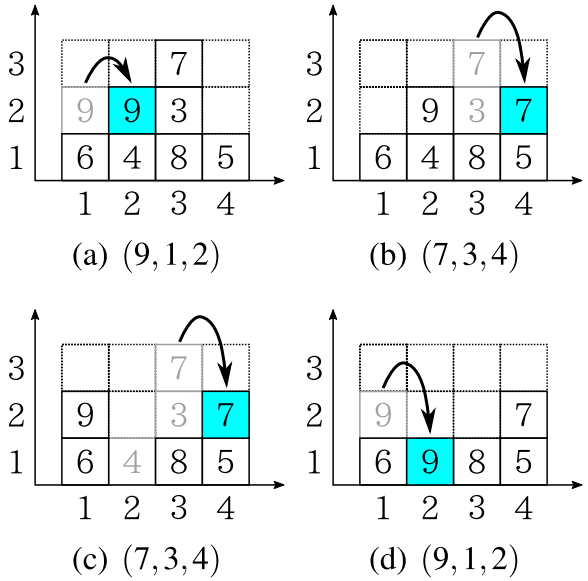


Fig. 10. An example of Theorems 4 and 5. (a) and (b): dominated sequence (9, 1, 2), (7, 3, 4), (c) and (d): dominating sequence (7, 3, 4), (9, 1, 2)

(RA1) $i_1 \notin \{i_2, \ldots, i_{n-1}, i_n\}$,
(RA2) $i_1 \in \mathscr{C}_{n-1} \wedge i_1 \notin \mathscr{C}_n$,
(RA3) $s_1 \notin \{s_2, d_2, \ldots, s_n, d_n\}$,
(RA4) $Q_{s_1}(\mathscr{C}) = i_1$.

Proof. From (RA4), the priority of block i_1 is higher (smaller) than the other blocks in stack s_1 in $\mathscr{C}$. Thus these blocks should appear in $L_{i_n s_n d_n}(\mathscr{C}_{n-1})$, because (RA2) implies that block i_1, which should be retrieved before them, is retrieved by the retrieval from $L_{i_n s_n d_n}(\mathscr{C}_{n-1})$. Furthermore, from (RA3), no block is relocated from or to stack s_1 by sequence $(i_2, s_2, d_2), \ldots, (i_n, s_n, d_n)$. Therefore, the block configuration of stack s_1 in $\mathscr{C}_1, \ldots, \mathscr{C}_{n-1}$, and $L_{i_n s_n d_n}(\mathscr{C}_{n-1})$ is the same as that in $\mathscr{C}$ except the absence of block i_1. It follows that block i_1 does not interfere any retrieval caused by sequence $(i_1, s_1, d_1), \ldots, (i_n, s_n, d_n)$ for $\mathscr{C}$ even if it is not relocated to stack d_1, except the retrieval from $L_{i_n s_n d_n}(\mathscr{C}_{n-1})$. Because (RA1) ensures that block i_1 is not relocated by sequence $(i_2, s_2, d_2), \ldots, (i_n, s_n,$

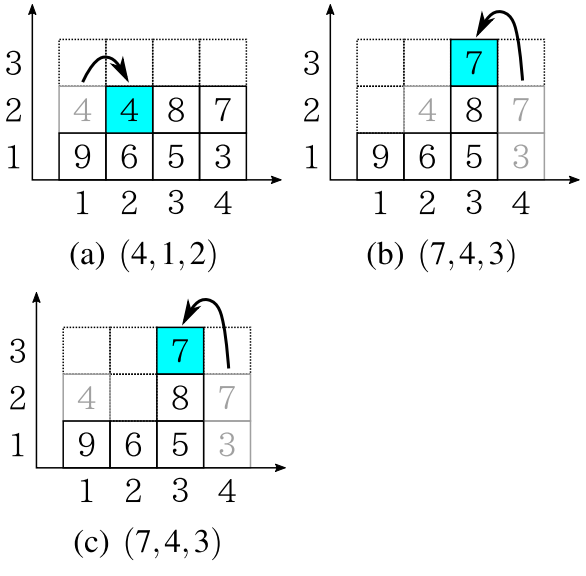


Fig. 11. An example of Theorem 6. (a) and (b): dominated sequence (4, 1, 2), (7, 4, 3), (c): dominating sequence (7, 4, 3)

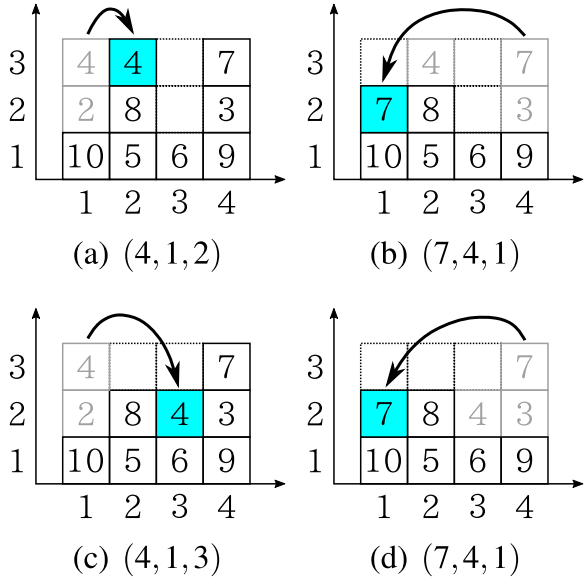


Fig. 12. An example of Theorem 7. (a) and (b): sequence (4, 1, 2), (7, 4, 1), (c) and (d): sequence (4, 1, 3), (7, 4, 1). The two sequences dominate each other

d_n), this sequence is admissible for $\mathscr{C}$. By further noting that no block is placed on block i_1 by this sequence for $\mathscr{C}$, we can see that block i_1 is retrieved also in the resulting configuration, so that it dominates $\mathscr{C}_n$. □

Fig. 11 provides an example of Theorem 11 where sequence (4, 1, 2), (7, 4, 3) is dominated by sequence (7, 4, 3).

Theorem 7 (Retrieval Rule B)**.** *Sequence* (i_1, s_1, d'_1), (i_2, s_2, d_2), … , (i_n, s_n, d_n) *dominates sequence* (i_1, s_1, d_1), (i_2, s_2, d_2), … , (i_n, s_n, d_n) *if all the following conditions are satisfied:*

(RB1) $i_1 \notin \{i_2, \dots, i_{n-1}, i_n\}$,
(RB2) $i_1 \in \mathscr{C}_{n-1} \wedge i_1 \notin \mathscr{C}_n$,
(RB3) $d'_1 \notin \{s_1, d_1, \dots, s_n, d_n\}$,
(RB4) $N_{d'_1}(\mathscr{C}) < T$,
(RB5) $Q_{d'_1}(\mathscr{C}) > i_1$.

Proof. From (RB5), blocks in stack d'_1 in $\mathscr{C}$ are not retrieved before block i_1 that, from (RB2), should be retrieved by the retrieval for $L_{i_n s_n d_n}(\mathscr{C}_{n-1})$. Thus these blocks should appear in $L_{i_n s_n d_n}(\mathscr{C}_{n-1})$. Furthermore, from (RB3), no block is relocated from or to stack d'_1 by sequence (i_1, s_1, d_1), … , (i_n, s_n, d_n) for $\mathscr{C}$. It follows that the block configuration of stack d'_1 in $\mathscr{C}$, $\mathscr{C}_1$, … , $\mathscr{C}_{n-1}$, and $L_{i_n s_n d_n}(\mathscr{C}_{n-1})$ is the same. This fact together with the feasibility of relocation (i_1, s_1, d'_1) for $\mathscr{C}$ from (RB4) ensures that block i_1 does not interfere any retrieval caused by sequence (i_1, s_1, d_1), … , (i_n, s_n, d_n) for $\mathscr{C}$ even if it is relocated to stack d'_1 instead of stack d_1, except the retrieval from $L_{i_n s_n d_n}(\mathscr{C}_{n-1})$. Therefore, sequence (i_1, s_1, d'_1), (i_2, s_2, d_2), … , (i_n, s_n, d_n) is admissible for $\mathscr{C}$. By further noting that no block is placed on block i_1 by this sequence for $\mathscr{C}$, we can see that block i_1 is retrieved also in the resulting configuration, so that it dominates $\mathscr{C}_n$. □

In Fig. 12, sequence (4, 1, 2), (7, 4, 1) dominates sequence (4, 1, 3), (7, 4, 1) and vice versa according to Theorem 7. Therefore, not both of the sequences should be forbidden in the branch-and-bound algorithm as noted in Section 3.1.

4.4. Dominance properties for the restricted BRP

Although Theorems 1–7 are originally meant for the unrestricted BRP, some of them are applicable also to the restricted BRP. However, it should be noted that source stacks of relocations are uniquely determined in the restricted BRP. For example, in Theorem 1, sequence (i_2, s_2, d_2), … , $(i_{n-1}, s_{n-1}, d_{n-1})$, (i_1, s_1, d_n) is admissible for $\mathscr{C}$ in the case of the unrestricted BRP. In order for this sequence to be admissible for the restricted BRP, the target block should be in stack s_1 when relocation (i_1, s_1, d_n) is performed. However, it is not the case in general and the sequence can be infeasible. In view of this, only Theorems 2 and 7 are applicable to the restricted BRP. Theorem 2 holds without any modification, whereas (RB4) should be replaced by the following condition in Theorem 7:

(RB4') $N_{d'_1}(\mathscr{C}) \le N_{d_1}(\mathscr{C})$.

In (RB2) of Theorem 7, block i_1 in stack d_1 is assumed to be retrieved from $L_{i_n s_n d_n}(\mathscr{C}_{n-1})$. It follows that block i_1 becomes the target block at some point, and only blocks above it are relocatable after that in the restricted BRP. To consider an admissible dominating sequence, all the blocks relocated to (from) stack d_1 after block i_1 in sequence (i_1, s_1, d_1), (i_2, s_2, d_2), … , (i_n, s_n, d_n) should also be relocated to (resp. from) stack d'_1. Therefore, we should consider sequence (i_1, s_1, d'_1), (i_2, s'_2, d'_2), … , (i_n, s'_n, d'_n) instead of sequence (i_1, s_1, d'_1), (i_2, s_2, d_2), … , (i_n, s_n, d_n), where, for $k = 2, \dots, n$.

$$s'_k = \begin{cases} d'_1, & s_k = d_1, \\ s_k, & s_k \neq d_1, \end{cases} \quad d'_k = \begin{cases} d'_1, & d_k = d_1, \\ d_k, & d_k \neq d_1. \end{cases} \tag{6}$$

To make this sequence admissible, stack d'_1 should be as slack as stack d_1, which is ensured by $N_{d'_1}(\mathscr{C}) \le N_{d_1}(\mathscr{C})$.

In the example of Fig. 13, two sequences dominate each other as in Fig. 12. Please note that the relocations of block 8 are changed according to the change in the destination stack of block 3.

5. Consistency of dominance properties

The theorems in the preceding section enable us to eliminate unnecessary nodes in the search tree of the branch-and-bound algorithm. However, they should be employed carefully so as not to exclude all optimal sequences. In the example of Fig. 6, sequence (3, 2, 1) dominates sequence (3, 2, 3) and vice versa according to Theorem 7 with $n = 1$, meaning that it is not sufficient to simply

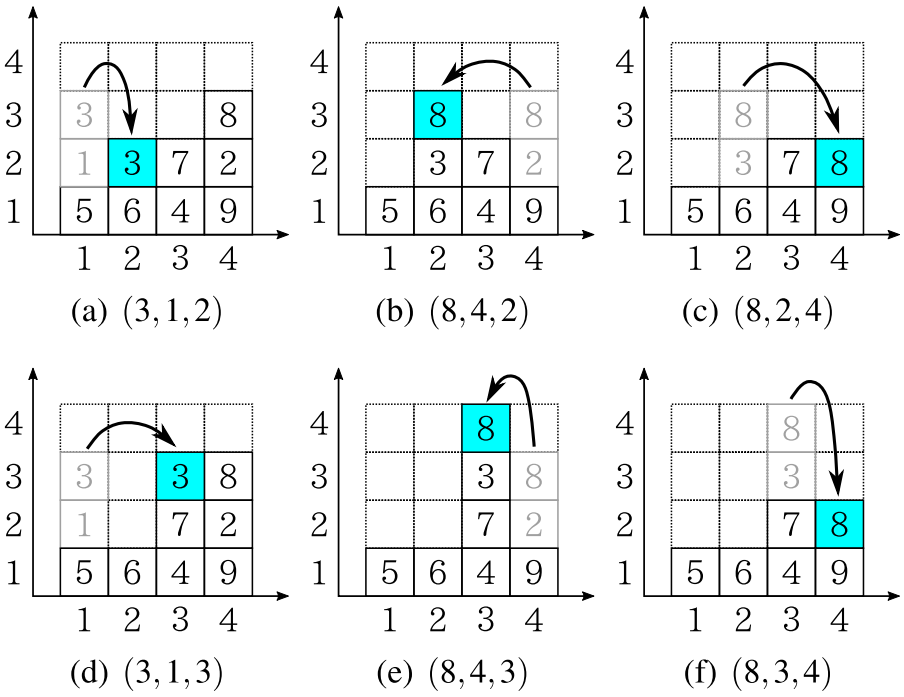


Fig. 13. An example of Theorem 7 for the restricted BRP. (a)–(c): sequence (3, 1, 2), (8, 4, 2), (8, 2, 4), (d)–(f): sequence (3, 1, 3), (8, 4, 3), (8, 3, 4). The two sequences dominate each other

forbid dominated sequences. Two sequences dominate each other also in Figs. 12 and 13. In this section we explain how to break such ties in the conditions to ensure the consistency of the dominance properties.

Theorems 1, 2, and 6 (Transitive Relocation Rules A and B, Retrieval Rule A) give strong dominance properties in the sense that they ensure the existence of a strictly dominating sequence with fewer relocations. On the other hand, the rest of the theorems only give weak dominance properties, and we should break ties consistently so as not to forbid all optimal sequences. The consistency is ensured based on stack priorities. In Theorem 3, ties are broken by adding the following condition:

(TC5) $d_1 \in \{s_2, d_2, \ldots, s_{n-1}, d_{n-1}\} \vee Q_{d_1'}(\mathscr{C}) > Q_{d_1}(\mathscr{C})$.

Condition $d_1 \in \{s_2, d_2, \ldots, s_{n-1}, d_{n-1}\}$ in (TC5) together with (TC3) means that we prefer a stack d_1' such that no block is relocated from or to stack d_1' by sequence $(i_2, s_2, d_2), \ldots, (i_n, s_n, d_n)$. If no block is relocated from or to stack d_1, either, a stack d_1' with the lower (larger) priority is preferred ($Q_{d_1'}(\mathscr{C}) > Q_{d_1}(\mathscr{C})$). If a preferred stack d_1' exists, sequence $(i_1, s_1, d_1), \ldots, (i_n, s_n, d_n)$ is forbidden. The same applies to Theorem 7, and

(**RB5'**) $(d_1 \in \{s_2, d_2, \ldots, s_n, d_n\} \wedge Q_{d_1'}(\mathscr{C}) > i_1) \vee Q_{d_1'}(\mathscr{C}) > Q_{d_1}(\mathscr{C})$

is checked instead of (RB5). Condition $Q_{d_1'}(\mathscr{C}) > Q_{d_1}(\mathscr{C})$ in (RB5') implies (RB5) because from (RB1) and (RB2), block i_1 is retrieved before any block in stack d_1 of $\mathscr{C}$ by sequence $(i_1, s_1, d_1), \ldots, (i_n, s_n, d_n)$, so that $i_1 < Q_{d_1}(\mathscr{C}) < Q_{d_1'}(\mathscr{C})$ should hold.

Ties in Theorems 4 and 5 can be broken by adding the following conditions, respectively:

(**IA3**) $Q_{s_1}(\mathscr{C}) > Q_{s_n}(\mathscr{C}_{n-1})$.
(**IB4**) $Q_{s_1}(\mathscr{C}) > Q_{s_n}(\mathscr{C}_{n-1})$.

However, even if ties are broken appropriately in this manner, Theorems 4 and 5 are possibly inconsistent with each other. Fig. 14 illustrates an example. In this case, $Q_1 = 3 > Q_4 = 2$ and sequence (8, 1, 2), (10, 3, 5), (5, 4, 5) satisfies (IA1)–(IA3). Hence it is dominated by sequence (10, 3, 5), (5, 4, 5), (8, 1, 2), and the former sequence is eliminated. However, $Q_3 = 4 > Q_4 = 2$ and the latter satisfies (IB1)–(IB4), so that it is also eliminated. To avoid this kind of inconsistency, only either Theorem 4 or Theorem 5 should be used. According to results of computational experiments, there was no significant difference in the performance of the two theorems. Thus Theorem 4, which is a little simpler than Theorem 5, was used in our proposed algorithm.

To break ties in Theorem 7 for the restricted BRP, (RB4') and (RB5') are combined into

(**RB4"**) $N_{d_1'}(\mathscr{C}) < N_{d_1}(\mathscr{C}) \vee (N_{d_1'}(\mathscr{C}) = N_{d_1}(\mathscr{C}) \wedge Q_{d_1}(\mathscr{C}) < Q_{d_1'}(\mathscr{C}))$.

In (RB4"), ties are broken by the stack height first, and next the stack priority.

6. New lower bound

One of the primary contributions of this study is the dominance properties in Section 4. The other is a new lower bound of the total number of relocations. It is an extension of the lower bound by Forster and Bortfeldt (2012). Kim and Hong (2006) utilized the total number of blocking blocks as a lower bound in their branch-and-bound algorithm because a blocking block should be relocated at least once in order to retrieve the block with a higher priority under it. Forster and Bortfeldt (2012) showed that it can be increased by one if a BG relocation does not exist for any topmost blocking block. In this section we further improve it by relaxing the condition for the increment. In the following, a minimal configuration is assumed, and the lower bound by Kim and Hong (2006) (the total number of blocking blocks) and that by Forster and Bortfeldt (2012) are denoted by LB-KH and LB-FB, respectively.

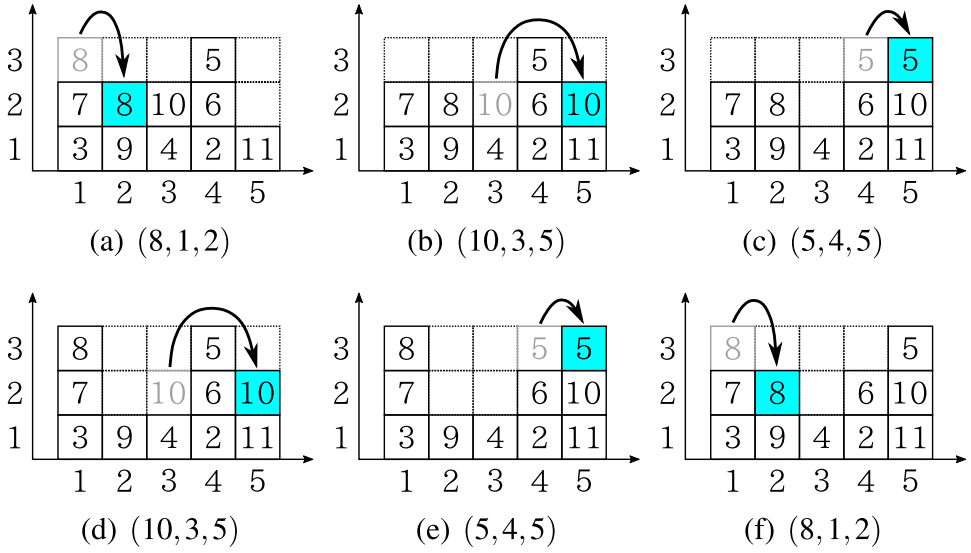


Fig. 14. An example of sequences for which (IA1)–(IA3) and (IB1)–(IB4) become inconsistent. (a)–(c): sequence (8, 1, 2), (10, 3, 5), (5, 4, 5), (d)–(f): sequence (10, 3, 5), (5, 4, 5), (8, 1, 2). The former sequence is eliminated and the latter is adopted according to (IA1)–(IA3), whereas the former is adopted and the latter is eliminated according to (IB1)–(IB4)

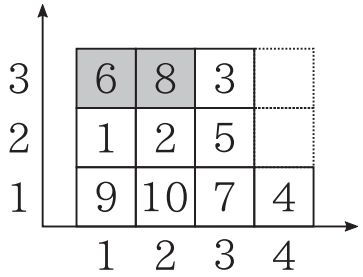


Fig. 15. A configuration for which LB-FB succeeds in improving LB-KH. LB-KH = 2, LB-FB = 3, and gray blocks are blocking blocks.

6.1. LB-FB: Lower bound by Forster and Bortfeldt (2012)

As a preparation, we investigate LB-FB in detail. If LB-KH is equal to the optimal value, every relocation in an optimal sequence should decrease blocking blocks by one. In other words, an optimal sequence should be composed only of BG relocations. Because only topmost blocks are relocatable at first, the first relocation in the sequence is a BG relocation of a topmost blocking block. It follows that the optimal value should be greater than LB-KH if a BG relocation does not exist for any topmost blocking block. In this case, LB-KH + 1 is a valid lower bound and LB-FB is given by LB-KH + 1. Otherwise, LB-FB is equal to LB-KH. In summary, LB-KH is increased by one if

$$\min_{\substack{s\in\mathscr{S}\\ Q_s<P_s^{\mathrm{top}}}} P_s^{\mathrm{top}} > \max_{s\in\mathscr{S}} Q_s \tag{7}$$

is satisfied. The lefthand side of (7) denotes the highest (smallest) priority of the topmost blocking blocks. In the configuration in Fig. 15, the total number of blocking blocks (gray blocks) is 2 and hence LB-KH = 2, whereas LB-FB = LB-KH + 1 = 3 because the highest (smallest) priority of the topmost blocking blocks is 6 and the lowest (largest) stack priority is 4.

6.2. Proposed lower bound LB-N

In the proposed lower bound, a similar idea to LB-FB is exploited to check whether a BG relocation exists or not. The difference from LB-FB is that blocking blocks above the target block are checked. We first identify candidates for their destination stacks. Suppose that the target block is in the t^{T}th tier of stack s^{T}. That is, the target block is block $P_{s^{\mathrm{T}}t^{\mathrm{T}}}$. Then, any stack $j \in \mathscr{S}^{\mathrm{slack}} \setminus \{s^{\mathrm{T}}\}$ is obviously a candidate destination stack. Even if $N_j = T$, stack j can be a candidate, provided that its topmost block is a blocking block and there exists a BG relocation for it. Therefore, the set of indices of candidate destination stacks is given by

$$\mathscr{D} = \left\{ j \in \mathscr{S} \setminus \{s^{\mathrm{T}}\} \mid N_j < T \vee Q_j < P_j^{\mathrm{top}} < \max_{s\in\mathscr{S}^{\mathrm{slack}}} Q_s \right\}. \tag{8}$$

In (8), $Q_j < P_j^{\mathrm{top}}$ means that the topmost block of stack j is a blocking block, and $P_j^{\mathrm{top}} < \max_{s\in\mathscr{S}^{\mathrm{slack}}} Q_s$ ensures that a BG relocation exists for this block. One may think that $\max_{s\in\mathscr{S}^{\mathrm{slack}}} Q_s$ should be replaced by $\max_{s\in\mathscr{S}} Q_s$ because the topmost blocking block of stack j can be relocated to stack s' even when $N_{s'} = T$, if the topmost block of stack s' is relocated beforehand. Suppose that such a stack s' satisfying $Q_{s'} > \max_{s\in\mathscr{S}^{\mathrm{slack}}} Q_s$ exists. Because the topmost block of stack s' should also be relocated by a BG relocation, its priority $P_{s'}^{\mathrm{top}}$ satisfies $Q_{s''} > P_{s'}^{\mathrm{top}} > Q_{s'}$ where stack s'' is the destination stack of this BG relocation. However, $N_{s''} = T$ holds from $Q_{s''} > Q_{s'} > \max_{s\in\mathscr{S}^{\mathrm{slack}}} Q_s$, and hence a BG relocation should exist for the topmost blocking block $P_{s''}^{\mathrm{top}}$ of stack s'' to relocate it beforehand. By repeatedly applying this argument, we can see that no BG relocation exists for the topmost blocking block of stack j. Thus stacks j with $P_j^{\mathrm{top}} > Q_j > \max_{s\in\mathscr{S}^{\mathrm{slack}}} Q_s$ are excluded from the candidate destination stacks.

Next, let us consider the priorities of the candidate destination stacks. We should take into account the effect of relocating blocks other than those above the target block. Because these relocations are BG relocations, they do not affect the priorities of their source stacks, while they always make the priorities of their destination stacks higher (smaller). It follows that the current priorities of the candidate destination stacks yield lowest possible values (upper bounds) of their actual priorities. Since our purpose here is to obtain a lower bound of the total number of relocations, we can safely assume that the priorities of the candidate destination stacks $j \in \mathscr{D}$ are given by their current values Q_j.

Now, let us check whether a BG relocation exists or not for the blocks above the target block, blocks $P_{s^{\mathrm{T}}N_{s^{\mathrm{T}}}}$, $P_{s^{\mathrm{T}},N_{s^{\mathrm{T}}}-1}$, ..., $P_{s^{\mathrm{T}},t^{\mathrm{T}}+1}$ in this order. According to Lemma 1 in Tanaka and Takii (2016), it can

be checked easily if the height limit is ignored. Specifically, a BG relocation exists for any block above the target block if Algorithm 4

Algorithm 4 Procedure for checking whether a BG relocation is possible or not.

```
1:  procedure BGRelocation(s^T, t^T, 𝒞)
2:      set 𝒟 by (8)
3:      for all j ∈ 𝒟 do
4:          q_j ← Q_j(𝒞)
5:      for i = 1 to N_{s^T}(𝒞) − t^T do
6:          p ← P_{s^T, N_{s^T}(𝒞)−i+1}
7:          d ← arg min_{j∈𝒟 ∧ q_j > p} q_j
8:          if d does not exist then
9:              return False
10:         q_d ← p
11:     return True
```

is feasible. If it is infeasible, that is, d is not found on line 7, a BG relocation does not exist for some block. In this case, LB-KH can be increased by one. The time complexity of this algorithm is given by $O((N_{s^T} - t^T)\log S)$ if the candidate destination stacks are sorted in the increasing order of their priorities beforehand. It is because this order is kept unchanged even if q_d is updated on line 10, and d can always be found in $O(\log S)$ time on line 7.

We apply this check in the framework of LB3 by Zhu et al. (2012). If no BG relocation exists for some block above the target block, LB-KH is increased by one. Otherwise, the blocks above the target block are removed from the current configuration together with the target block. Then, the same check is applied to the blocks above the new target block. This procedure is repeated until it is proved that LB-KH can be increased by one, or all the blocks are removed. The algorithm for computing the proposed lower bound LB-N is presented in Algorithm 5.

Algorithm 5 Proposed lower bound LB-N.

```
1:  procedure LB-N(𝒞)
2:      while N(𝒞) > 0 do
3:          s^T ← arg min_{s∈𝒮} Q_s(𝒞)
4:          t^T ← arg min_{1≤t≤N_{s^T}(𝒞)} P_{s^T t}(𝒞)
5:          if BGRelocation(s^T, t^T, 𝒞) = False then
6:              return LB-KH + 1
7:          Remove blocks P_{s^T t}(𝒞) (t^T ≤ t ≤ N_{s^T}(𝒞)) from 𝒞
8:          𝒞 ← R(𝒞)
9:      return LB-KH
```

The validity of LB-N is verified as follows. Let us consider a minimal configuration $\mathscr{C}$ and denote the target block in $\mathscr{C}$ by block x. Let us also denote by $\mathscr{C}'$, the configuration obtained by removing the blocks above block x as well as block x itself, and by block x', the target block in $\mathscr{C}'$. Then, it suffices to show that if $f^*(\mathscr{C}) =$ LB-KH, a BG relocation exists for any block above block x' in $\mathscr{C}'$ because the contraposition holds. From $f^*(\mathscr{C}) =$ LB-KH, all relocations in an optimal sequence should be BG relocations, so that all blocking blocks in $\mathscr{C}$ including those above block x are relocated by BG relocations. Because BG relocations for these blocks always make the priorities of their destination stacks higher (smaller), the actual priorities of the destination stacks of blocks above block x' in $\mathscr{C}$ are not lower (larger) than those in $\mathscr{C}'$. It follows that a BG relocation should exist for any block above block x' also in $\mathscr{C}'$.

It is not difficult to show that LB-N always dominates LB-FB. If (7) is satisfied,

$$P_{s^T}^{\text{top}} \geq \min_{\substack{s\in\mathscr{S}\\ Q_s < P_s^{\text{top}}}} P_s^{\text{top}} > \max_{s\in\mathscr{S}} Q_s \geq \max_{s\in\mathscr{D}} Q_s \tag{9}$$

holds. It follows that the procedure in Algorithm 4 is infeasible for the topmost block $P_{s^T}^{\text{top}} = P_{s^T N_{s^T}}$ of stack s^T, so that LB-N = LB-KH + 1.

Fig. 16 illustrates the procedure of LB-N. We note that LB-FB = LB-KH = 3 for the configuration in (a). First, we check block 3 above the target block 1. The set of indices of candidate destination stacks in (8) is $\mathscr{D} = \{4\}$ because no BG relocation exists for the topmost blocking block 5 of stack 2, and the topmost block 8 of stack 3 is not a blocking block. Because the relocation of block 3 to stack 4 ($\in \mathscr{D}$) is a BG relocation (b), we remove this block and then the target block. Now, block 2 becomes the target block and $\mathscr{D} = \{1, 4\}$ (c). Thus we check block 5, and it can be relocated to stack 1 by a BG relocation (d). We next check block 7, for which no BG relocation is possible (e). Finally, it is proved that LB-KH can be increased by one and LB-N is given by LB-KH + 1 = 4.

6.3. Speed-up of lower bound computation

A lower bound is computed at every node in the search tree of the branch-and-bound algorithm. Compared to LB-KH that can be computed incrementally in $O(1)$ time from that for the parent node, LB-N needs a longer computation time. Here, we show that we need not compute LB-N at some nodes, where we can obtain a lower bound in $O(1)$ time.

Let us denote LB-KH and LB-N for $\mathscr{C}$ by LB-KH($\mathscr{C}$) and LB-N($\mathscr{C}$), respectively. Let us also denote by $\mathscr{C}^{\text{parent}}$ and $\mathscr{C}^{\text{cur}}$ the configurations represented by the parent and current nodes, respectively. Because $\mathscr{C}^{\text{cur}}$ is obtained by relocating one block in $\mathscr{C}^{\text{parent}}$, $f^*(\mathscr{C}^{\text{cur}}) + 1 \geq$ LB-N($\mathscr{C}^{\text{parent}}$) should hold due to the validity of LB-N($\mathscr{C}^{\text{parent}}$) as a lower bound. Therefore, if both LB-N($\mathscr{C}^{\text{parent}}$) = LB-KH($\mathscr{C}^{\text{parent}}$) + 1 and LB-KH($\mathscr{C}^{\text{cur}}$) = LB-KH($\mathscr{C}^{\text{parent}}$) − 1 hold, we obtain

$$f^*(\mathscr{C}^{\text{cur}}) \geq \text{LB-N}(\mathscr{C}^{\text{parent}}) - 1 = \text{LB-KH}(\mathscr{C}^{\text{parent}}) = \text{LB-KH}(\mathscr{C}^{\text{cur}}) + 1. \tag{10}$$

This relation means that LB-KH($\mathscr{C}^{\text{cur}}$) + 1 is a valid lower bound for $\mathscr{C}^{\text{cur}}$ and it always dominates LB-N($\mathscr{C}^{\text{cur}}$) from the definition of LB-N. In this case, LB-KH($\mathscr{C}^{\text{cur}}$) + 1 can be used as a lower bound instead of LB-N($\mathscr{C}^{\text{cur}}$).

7. Computational experiments

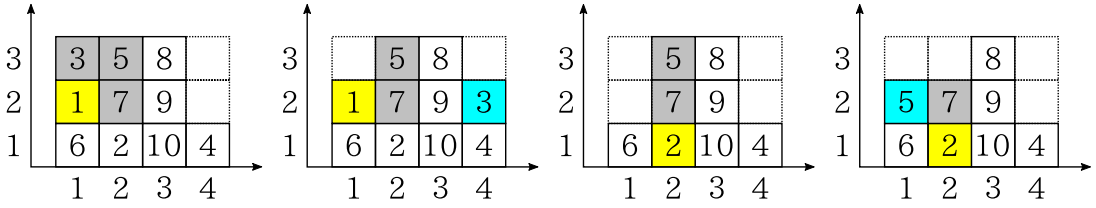
In this section we examine the effectiveness of the proposed exact algorithm by computational experiments.

7.1. Benchmark instances

We used two sets of benchmark instances in the literature. The first one is the dataset in Caserta et al. (2012, 2011).[1]. The instances are characterized by two parameters: the number of stacks S and the number of blocks in every stack H. Thus the total number of blocks is $N = SH$, and both S and H range from 3 to 10. The set contains 40 instances for each combination of S and H. The stack height limit T was set to $T := H + 2$. These instances are referred to as CVS instances. The second set derives from Zhu et al. (2012). Since the web site of the authors no longer exists, it is mirrored in our web page.[2] The instances in this set are characterized by S, T,

[1] Downloadable from https://www.bwl.uni-hamburg.de/en/iwi/forschung/projekte/dataprojekte/brp-instances-caserta-etal-2012.zip.

[2] https://sites.google.com/site/shunjitanaka/brp.

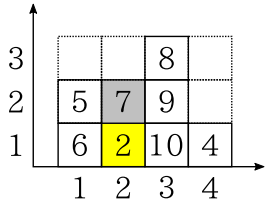


(a) Initial configuration. Block 1 is the target block. $\mathscr{D} = \{4\}$

(b) Relocation of block 3 to stack 4 is a BG relocation

(c) Blocks 3 and 1 are removed. Block 2 is the next target block. $\mathscr{D} = \{1,4\}$

(d) Relocation of block 5 to stack 1 is a BG relocation



(e) No BG relocation exists for block 7

Fig. 16. An example of calculation of LB-N for a configuration for which LB-N = 4 is greater than LB-FB(= LB-KH = 3).

and N. They range from 6 to 10, from 3 to 7, and from $S(T-1)$ to $ST-1$, respectively. The set contains 100 instances for each combination of S, T, and N. Among them, 9000 instances with $T \leq 6$ were used because it is too time-consuming to solve all 3500 instances with $T = 7$. These instances are referred to as ZQLZ instances.

7.2. Algorithm specifications

The proposed exact algorithm was coded in C. The source code is available from our aforementioned web page. The computational experiments were conducted on a desktop computer with an Intel Core i7-6700 3.4 GHz CPU and 8GB RAM. The time limit was set to 1800 seconds for each instance.

To describe the exact algorithm, "T", "I", "R", "FB", and "N" are used, which stand for:

T Transitive Relocation Rules A, B, and C (Theorems 1–3) are used,
I Independent Relocation Rule A (Theorem 4) is used,
R Retrieval Rules A and B (Theorems 6 and 7) are used,
FB LB-FB is used as a lower bound,
N LB-N is used as a lower bound.

For example, "FB" is the naive algorithm that employs LB-FB as a lower bound, and "TIR/N" stands for the algorithm that employs LB-N, Transitive Relocation Rules A–C (T), Independent Relocation Rule A (I), and Retrieval Rules A and B (R).

We also improved the exact algorithm in Tanaka and Takii (2016) for the restricted BRP with distinct priorities by the dominance rules. In this case, "T" and "R" stand for:

T Transitive Relocation Rule A (Theorem 2) is used.
R Retrieval Rule B (Theorem 7) is used.

7.3. Results for CVS instances

First, the unrestricted BRP is investigated through computational results for the CVS instances. They are summarized in Table 3. In this table, "%imp" presents the percentages of instances for which the initial lower bound is given not by LB-KH but by LB-KH + 1. For example, LB-FB is equal to LB-KH + 1 for the initial configurations of $0.425 \times 40 = 17$ instances with $S = 6$ and $H = 6$, whereas LB-N = LB-KH + 1 for all these 40 instances. The difference between LB-FB and LB-N is significant when S is large. LB-FB becomes less efficient as S becomes larger, because the lefthand side of (7) becomes smaller so that (7) is less often satisfied. In contrast, LB-N is not affected by S, and it is almost always better than LB-KH when $H \geq 5$. This result verifies the effectiveness of the proposed lower bound LB-N.

In Table 3, the column "opt" denotes the number of instances solved to optimality within the time limit, and "ave" and "max" the average and maximum computation times in seconds, respectively. In addition, the average objective value by the greedy heuristic and the average optimal (or best) value are provided in the columns "HEUR" and "UB", respectively. The results of the exact algorithms in Expósito-Izquierdo et al. (2014); Tricoire et al. (2018) are also presented in the table, although only the number of instances solved to optimality is provided in Expósito-Izquierdo et al. (2014). In Tricoire et al. (2018), several combinations of lower bounds, upper bounds, and branch-and-bound algorithms were tested. For a fair comparison, we picked up the best result for each group of instances, in terms of the number of instances solved to optimality and the average computation time. Please also note that the average computation time in Tricoire et al. (2018) is over instances solved to optimality, in contrast to ours over 40 instances. It is verified from the table that the dominance rules and the new lower bound bring a significant improvement of the exact algorithm. Indeed, the number of instances solved to optimality increases and the computation time is decreased from the naive algorithm FB. The greedy heuristic yields near-optimal solutions for small-sized instances with $H = 3$. However, the gap from the optimal value increases as H increases, and it reaches an average of 6.3 relocations for instances with $H = S = 6$. It seems difficult to obtain a good solution for such instances by a simple greedy heuristic. With regard to the exact algorithms in the literature, FB already outperforms the A* algorithm in Expósito-Izquierdo et al. (2014) and yields a similar performance to the branch-and-bound algorithms

Table 3
Computational results for CVS instances ($T = H + 2$, unrestricted).

H	S	N	Initial LB		HEUR	UB	Exact Algorithms								
			LB-FB	LB-N			FB			TIR/N			Expósito-Izquierdo et al. (2014)[a]	Tricoire et al. (2018)[b]	
			%imp	%imp			opt	ave	max	opt	ave	max	opt	opt	ave
3	3	9	45.0	80.0	5.275	**4.975**	**40**	0.00	0.00	**40**	0.00	0.00	**40**	**40**	0.00
	4	12	45.0	77.5	6.250	**6.025**	**40**	0.00	0.00	**40**	0.00	0.00	**40**	**40**	0.00
	5	15	20.0	72.5	7.075	**6.850**	**40**	0.00	0.00	**40**	0.00	0.00	**40**	**40**	0.00
	6	18	25.0	72.5	8.450	**8.275**	**40**	0.00	0.00	**40**	0.00	0.00	**40**	**40**	0.00
	7	21	15.0	67.5	9.300	**9.100**	**40**	0.00	0.00	**40**	0.00	0.00	**40**	**40**	0.00
	8	24	12.5	87.5	10.650	**10.300**	**40**	0.00	0.02	**40**	0.00	0.00	**40**	**40**	0.00
4	4	16	50.0	92.5	10.850	**9.725**	**40**	0.00	0.01	**40**	0.00	0.00	**40**	**40**	0.00
	5	20	32.5	87.5	13.250	**12.250**	**40**	0.00	0.03	**40**	0.00	0.00	**40**	**40**	0.00
	6	24	27.5	90.0	14.275	**13.225**	**40**	0.02	0.67	**40**	0.00	0.01	**40**	**40**	0.04
	7	28	10.0	97.5	16.300	**15.375**	**40**	0.06	1.92	**40**	0.00	0.01	5	**40**	0.19
5	4	20	45.0	100.0	16.300	**14.700**	**40**	0.11	2.23	**40**	0.00	0.09	**40**	**40**	0.19
	5	25	47.5	100.0	20.700	**17.425**	**40**	8.39	293.88	**40**	0.08	2.08	25	**40**	19.94
	6	30	25.0	100.0	23.850	**20.800**	**40**	27.78	702.94	**40**	0.10	2.00	1	**40**	50.73
	7	35	12.5	95.0	25.700	**22.575**	38	119.63	1800.00	**40**	0.72	20.20	1	37	10.46
	8	40	17.5	100.0	28.425	**25.600**	37	192.26	1800.00	**40**	3.15	108.39	0	37	89.23
	9	45	15.0	100.0	31.575	**28.350**	37	244.85	1800.00	**40**	2.17	57.10	0	36	169.18
	10	50	7.5	100.0	34.200	**30.825**	30	593.64	1800.00	**40**	2.85	32.82	0	30	162.69
6	6	36	42.5	100.0	34.100	**27.800**	28	732.47	1800.00	**40**	47.95	731.50	0	24	292.13
	10	60	10.0	100.0	48.950	41.925	5	1636.01	1800.00	28	689.04	1800.00	0	5	270.12
10	6	60	50.0	100.0	95.575	76.650	0	1800.00	1800.00	0	1800.00	1800.00	0	0	–
	10	100	35.0	100.0	132.075	115.150	0	1800.00	1800.00	0	1800.00	1800.00	0	0	–

[a] Java SE 7, run on an Intel Core 2 Duo E8500 3.16 GHz CPU with 4GB RAM, time limit of 24 h.
[b] C++, run on two Intel Xeon E5-2650 v2 2.60 GHz CPUs with 64GB RAM in parallel, time limit of 1800 s. The average computation time is over instances solved to optimality.

Table 4
Effect of dominance rules and new lower bound for CVS instances ($T = H + 2$).

H	S	N	FB		N		T/N		I/N		R/N		TI/N		TIR/N	
			opt	ave	opt	ave	opt	ave	opt	ave	opt	ave	opt	ave	opt	ave
3	3	9	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00
	4	12	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00
	5	15	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00
	6	18	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00
	7	21	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00
	8	24	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00
4	4	16	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00
	5	20	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00
	6	24	**40**	0.02	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00
	7	28	**40**	0.06	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00	**40**	0.00
5	4	20	**40**	0.11	**40**	0.04	**40**	0.00	**40**	0.03	**40**	0.04	**40**	0.00	**40**	0.00
	5	25	**40**	8.39	**40**	2.22	**40**	0.21	**40**	0.83	**40**	2.62	**40**	0.08	**40**	0.08
	6	30	**40**	27.78	**40**	2.66	**40**	0.38	**40**	0.84	**40**	2.97	**40**	0.11	**40**	0.10
	7	35	38	119.63	**40**	42.40	**40**	4.81	**40**	6.16	**40**	50.15	**40**	0.74	**40**	0.72
	8	40	37	192.26	39	62.80	**40**	38.42	**40**	29.15	39	59.22	**40**	3.32	**40**	3.15
	9	45	37	244.85	39	71.51	**40**	37.92	**40**	10.45	39	74.99	**40**	2.29	**40**	2.17
	10	50	30	593.64	39	155.13	**40**	56.04	**40**	12.41	39	152.19	**40**	3.50	**40**	2.85
6	6	36	28	732.47	31	524.94	38	215.90	35	334.01	31	538.18	**40**	51.06	**40**	47.95
	10	60	5	1636.01	14	1297.31	16	1182.65	23	872.74	14	1274.01	27	738.72	28	689.04
10	6	60	0	1800.00	0	1800.00	0	1800.00	0	1800.00	0	1800.00	0	1800.00	0	1800.00
	10	100	0	1800.00	0	1800.00	0	1800.00	0	1800.00	0	1800.00	0	1800.00	0	1800.00

in Tricoire et al. (2018) (note that a time limit of 24 h was imposed in Expósito-Izquierdo et al. (2014), in contrast to 1800 seconds in Tricoire et al. (2018) and in our case). Since our proposed algorithm TIR/N obviously outperforms FB, we can conclude that TIR/N outperforms the existing algorithms in Expósito-Izquierdo et al. (2014); Tricoire et al. (2018).

More detailed results of the proposed exact algorithm are presented in Table 4. This table provides the number of instances solved to optimality and the average computation time in seconds with and without the three types of dominance rules, to examine their effects as well as that of the new lower bound. The results in this table indicate that both the dominance rules and the new lower bound yield significant improvements from the naive algorithm FB. It is observed from the results of T/N and I/N that the independent relocation rule becomes more effective than the transitive relocation rules as the number of stacks increases. This can be explained as follows. The total number of combinations of two relocations is $S^2(S-1)^2$, and the number of two independent relocations among them is $S(S-1)(S-2)(S-3)$. The independent relocation rule eliminates half of them at best. On the other hand, the number of transitive relocations is $S^2(S-1)$. Even if all of them are eliminated by the transitive relocation rules, the latter are less effective than the former when S is large, because $S(S-1)(S-2)(S-3)/2 > S^2(S-1)$. Unlike these two types of rules, the retrieval rules did not greatly affect the efficiency of the algorithm. R/N was even slower than N for some instances

Table 5
Computational results for CVS instances ($T = H + 2$, restricted.)

H	S	N	UB	Tanaka and Takii (2016)			TR			Zehendner and Feillet (2014)[a]	Expósito-Izquierdo et al. (2014)[b]	Zehendner et al. (2015)[c]		Galle et al. (2017)[d]	
				opt	ave	max	opt	ave	max	opt	opt	opt	ave	opt	ave
3	3	9	**5.000**	**40**	0.00	0.00	**40**	0.00	0.00	39	**40**	**40**	0.1	**40**	0.1
	4	12	**6.175**	**40**	0.00	0.00	**40**	0.00	0.00	**40**	**40**	**40**	0.3	**40**	0.1
	5	15	**7.025**	**40**	0.00	0.00	**40**	0.00	0.00	39	**40**	**40**	0.8	**40**	0.1
	6	18	**8.400**	**40**	0.00	0.00	**40**	0.00	0.00	38	**40**	**40**	4.2	**40**	0.4
	7	21	**9.275**	**40**	0.00	0.00	**40**	0.00	0.00	39	**40**	**40**	5.8	**40**	0.6
	8	24	**10.650**	**40**	0.00	0.00	**40**	0.00	0.00	37	**40**	**40**	11.2	**40**	1.3
4	4	16	**10.200**	**40**	0.00	0.00	**40**	0.00	0.00	32	**40**	**40**	1.2	**40**	0.1
	5	20	**12.950**	**40**	0.00	0.00	**40**	0.00	0.00		**40**	**40**	5.8	**40**	0.5
	6	24	**14.025**	**40**	0.00	0.01	**40**	0.00	0.01		**40**	**40**	16.1	**40**	2.3
	7	28	**16.125**	**40**	0.00	0.02	**40**	0.00	0.02	20	**40**	**40**	90.1	**40**	6.4
5	4	20	**15.425**	**40**	0.00	0.00	**40**	0.00	0.00	10	**40**	**40**	19.9	**40**	1.8
	5	25	**18.850**	**40**	0.00	0.04	**40**	0.00	0.04		**40**	39	369.3	**40**	41.9
	6	30	**22.075**	**40**	0.02	0.48	**40**	0.01	0.25	4	**40**	33	524.3	39	68.0
	7	35	**24.250**	**40**	0.45	8.53	**40**	0.10	1.86		37	24	487.7	36	170.9
	8	40	**27.700**	**40**	9.10	211.34	**40**	1.44	26.51		4	9	749.4	33	590.2
	9	45	**30.450**	**40**	10.67	162.25	**40**	1.98	21.35		0	5	126.1	26	658.2
	10	50	33.275	37	185.67	1800.00	38	108.09	1800.00		0	2	–	22	1111.0
6	6	36	**30.875**	**40**	21.73	473.51	**40**	8.96	176.49		0	7	1466.5	19	441.7
	10	60	45.850	17	1172.01	1800.00	24	847.41	1800.00		0	2	–	2	–
10	6	60	78.675	3	1708.49	1800.00	4	1663.64	1800.00		0	0	–	0	–
	10	100	121.325	0	1800.00	1800.00	0	1800.00	1800.00		0	0	–	0	–

[a] IBM ILOG CPLEX 12.1, run on an Intel Xeon 2.67 GHz CPU with 3.48GB RAM, time limit of 1 h.
[b] Java SE 7, run on an Intel Core 2 Duo E8500 3.16 GHz CPU with 4GB RAM, time limit of 24 h.
[c] IBM ILOG CPLEX 12.5, run on an Intel Xeon 3.07 GHz CPU with 12.0GB RAM, time limit of 1 h. The average computation time is over non-trivial instances solved to optimality within the time limit.
[d] Gurobi Optimizer 7.0.1, run on four Intel E5-2690 v4 2.6 GHz CPUs with 8GB RAM, time limit of 1 h. The average computation time is over non-trivial instances solved to optimality within the time limit.

due to computational efforts required for applying the dominance rules. Nevertheless, TIR/N was always a little faster than TI/N. It is because the program codes for checking the dominance rules are mostly composed of common parts. Hence, applying the retrieval rules in addition to the other rules requires only a small computational effort.

Next, the restricted BRP is investigated through computational results for the CVS instances. In Table 5, our previous algorithm in Tanaka and Takii (2016) and an improved one TR using the dominance rules are compared. We also present the results of four existing exact approaches: the branch-and-price algorithm (Zehendner and Feillet, 2014), the A* algorithm (Expósito-Izquierdo et al., 2014), and the IP approaches (Galle et al., 2017; Zehendner et al., 2015). It should be noted that the average computation time in Galle et al. (2017); Zehendner et al. (2015) is over non-trivial instances solved to optimality: Lower and upper bounds are computed for each instance in pre-processing, and it is considered non-trivial when the LB/UB gap is not zero. With regard to Expósito-Izquierdo et al. (2014); Zehendner and Feillet (2014), only the number of instances solved to optimality is shown in the table. It is because the average computation time in Zehendner and Feillet (2014) is provided separately for the root node and inner nodes of the search tree, which makes it difficult to evaluate the overall computation time. Moreover, computation time is not reported in Expósito-Izquierdo et al. (2014). Among the four approaches, the IP approach by Galle et al. (2017) succeeded in solving more instances to optimality than Expósito-Izquierdo et al. (2014); Zehendner et al. (2015); Zehendner and Feillet (2014). Our previous algorithm (Tanaka and Takii, 2016) solved even more instances to optimality in spite of a shorter time limit of 1800 s (the time limit in Galle et al., 2017 was set to 1 h). The dominance rules improved it and TR solved nine more instances to optimality. Although the improvement was not so significant as that for the unrestricted BRP, we can see that TR yields the best performance among these approaches.

There are several studies on exact approaches to the restricted BRP whose results are not presented in Table 5. Expósito-Izquierdo et al. (2015) corrected a mistake in the ILP formulation in Caserta et al. (2012). They also proposed a branch-and-bound algorithm that is considerably faster than their previous A* algorithm (Expósito-Izquierdo et al., 2014). Eskandari and Azari (2015) corrected the mistake in the ILP formulation in Caserta et al. (2012) as well, and introduced some cuts to further improve it. Ku and Arthanari (2016b) proposed another branch-and-bound algorithm. However, these four approaches were applied to only 5 out of 40 instances in each instance group, so that their results are not included in Table 5. To examine them, the computation times by these approaches are summarized in Tables 6 and 7. We also present, in the same tables, computation times by our previous algorithm (Tanaka and Takii, 2016), TR in this study, and the IP approach (Galle et al., 2017). Among the approaches in Eskandari and Azari (2015); Expósito-Izquierdo et al. (2015); Galle et al. (2017); Ku and Arthanari (2016b), the branch-and-bound algorithm (Expósito-Izquierdo et al., 2015) seems to be the fastest for most of the instances in Table 6, although computation times by the IP approach (Galle et al., 2017) are available only for non-trivial instances (not marked with an asterisk). For the instances in Table 7, only results by the two IP approaches (Eskandari and Azari, 2015; Galle et al., 2017) are available, and we can see that Galle et al. (2017) is faster than Eskandari and Azari (2015). On the other hand, our previous algorithm and TR took less than 0.01 s for every instance in these tables, thus outperforming the other approaches in terms of computation time.

7.4. Results for ZQLZ instances

In Table 8, results for the ZQLZ instances are summarized. The number of instances is shown in the column "n". We can again verify that the dominance rules and the new lower bound considerably improve the efficiency of the exact algorithms. Unlike the CVS instances, the ZQLZ instances seem to be easier to solve

Table 6
Comparison of computation time (in seconds) for a subset of CVS instances: Our previous branch-and-bound algorithm in Tanaka and Takii (2016), TR in this study, IP approach and b&b algorithm in Expósito-Izquierdo et al. (2015), IP approach in Eskandari and Azari (2015), b&b algorithm in Ku and Arthanari (2016b), and IP approach in Galle et al. (2017) ($T = H + 2$, restricted).

H	S	Instance No.	b&b (Tanaka and Takii, 2016)	TR	IP (Expósito-Izquierdo et al., 2015)[a]	b&b (Expósito-Izquierdo et al., 2015)[a]	IP (Eskandari and Azari, 2015)[b]	b&b (Ku and Arthanari, 2016b)[c]	IP (Galle et al., 2017)[d]
3	3	1	0.00	0.00	1.18	0.007	0.11		*
		2	0.00	0.00	1.39	0.007	0.11		*
		3	0.00	0.00	1.00	0.006	0.08		*
		4	0.00	0.00	1.09	0.007	0.09		*
		5	0.00	0.00	0.68	0.007	0.06		*
	4	1	0.00	0.00	4.76	0.008	0.28		0.014
		2	0.00	0.00	18.39	0.007	0.22		*
		3	0.00	0.00	11.71	0.008	0.34		*
		4	0.00	0.00	16.06	0.007	0.26		*
		5	0.00	0.00	18.04	0.007	0.27		*
	5	1	0.00	0.00	83.09	0.010	0.72		*
		2	0.00	0.00	75.95	0.007	1.20		*
		3	0.00	0.00	100.71	0.013	0.91		*
		4	0.00	0.00	95.31	0.008	0.80		*
		5	0.00	0.00	65.32	0.026	1.59		0.097
	6	1	0.00	0.00	124.11	0.086	5.35	5.26	0.397
		2	0.00	0.00	113.29	0.008	1.59	0.00	*
		3	0.00	0.00	89.06	0.019	3.09	2.01	*
		4	0.00	0.00	93.12	0.016	1.78	0.03	*
		5	0.00	0.00	96.50	0.007	1.23	0.00	*
	7	1	0.00	0.00	182.14	0.009	2.89	0.08	*
		2	0.00	0.00	284.29	0.011	3.29	0.60	*
		3	0.00	0.00	119.20	0.011	3.34	1.98	*
		4	0.00	0.00	472.32	0.010	3.35	0.30	*
		5	0.00	0.00	277.97	0.038	7.97	191.60	*
	8	1	0.00	0.00	84.66	0.021	7.25	0.75	0.508
		2	0.00	0.00	14,403.33	0.055	9.31	8.65	*
		3	0.00	0.00	8298.85	0.012	6.57	0.66	*
		4	0.00	0.00	250.33	0.013	11.03	7.24	*
		5	0.00	0.00	6384.65	0.022	11.06	13.53	*
4	4	1	0.00	0.00	71.96	0.017	1.25	0.08	*
		2	0.00	0.00	228.91	0.025	0.93	0.00	0.036
		3	0.00	0.00	71.99	0.009	1.56	0.03	0.077
		4	0.00	0.00	65.25	0.008	0.78	0.00	*
		5	0.00	0.00	89.36	0.013	1.15	0.04	0.081
	5	1	0.00	0.00	3544.86	0.540	61.62	21.08	0.543
		2	0.00	0.00	326.54	0.043	5.27	0.15	0.229
		3	0.00	0.00	1023.98	0.018	8.28	2.60	0.261
		4	0.00	0.00	119.86	0.014	2.96	0.02	*
		5	0.00	0.00	2656.67	0.579	81.26	93.40	0.763
	6	1	0.00	0.00	4077.08	17.476	–	64.73	8.679
		2	0.00	0.00	18985.03	0.030	5.23	0.02	0.541
		3	0.00	0.00	1706.75	0.069	12.86	31.90	*
		4	0.00	0.00	2376.86	0.315	23.40	59.53	0.828
		5	0.00	0.00	8564.20	0.076	45.22	107.15	0.641

[a] Java SE 7, run on an Intel Core 2 Duo E8500 3.16GHz CPU with 4GB RAM.
[b] IBM ILOG CPLEX 12.1.0, run on an Intel Core i3 CPU with 4GB RAM, time limit of 900 s.
[c] Java, run on an Intel Core i5-2500 3.3GHz CPU with 8GB RAM.
[d] Gurobi Optimizer 7.0.1, run on four Intel E5-2690 v4 2.6GHz CPUs with 8GB RAM, time limit of 1 h. * means a trivial instance.

as restricted BRPs than as unrestricted BRPs, regardless of S. The reason for it is not clear, but the two sets are different in the fill rate: $2S$ slots are not filled with blocks in the CVS instances ($T = H + 2$), whereas at most S slots are not filled in the ZQLZ instances. This difference could have affected the efficiency of the exact algorithms.

Zhu et al. (2012) proposed IDA* algorithms for the restricted and unrestricted BRPs. With regard to the restricted BRP, our previous algorithm (Tanaka and Takii, 2016) outperforms their algorithm, as already reported in Tanaka and Takii (2016). However, the comparison is not easy on the unrestricted BRP only from limited information in Zhu et al. (2012). Nevertheless, we try to draw as fair a comparison as possible. According to the detailed results obtained from the authors' web page before it was closed, their algorithm IDA*-U solved 5307 instances out of 9000 to optimality, when the maximum number of nodes generated in the search tree was limited to 2^{30}. They also proposed a better algorithm named IDA*-UM that employs a memory dominance technique to eliminate the same configurations in the search tree. Unfortunately, they imposed a very short time limit of 1 s on it in order to use this algorithm not as an exact algorithm but as a metaheuristic. According to the results presented in Zhu et al. (2012), IDA*-UM solved 4868 instances to optimality within this time limit. On the other hand, TIR/N solved to optimality 8135 instances when the maximum number of generated nodes was limited to 2^{30}, and 5472 instances within 0.01 s. Although the Pentium4 3.0 GHz CPU in Zhu et al. (2012) is considerably slower than ours, we can conclude from these facts that our proposed algorithm outperforms IDA*-U and IDA*-UM in Zhu et al. (2012).

Table 7
Comparison of computation time (in seconds) for a subset of CVS instances: Our previous branch-and-bound algorithm in Tanaka and Takii (2016), TR in this study, and IP approaches in Eskandari and Azari (2015); Galle et al. (2017) ($T = H + 2$, restricted).

H	S	Instance No.	b&b (Tanaka and Takii, 2016)	TR	IP (Eskandari and Azari, 2015)[a]	IP (Galle et al., 2017)[b]
4	7	1	0.00	0.00	44.10	2.995
		2	0.00	0.00	20.47	1.245
		3	0.00	0.00	18.95	*
		4	0.00	0.00	49.78	2.310
		5	0.00	0.00	85.75	2.160
5	4	1	0.00	0.00	118.20	0.295
		2	0.00	0.00	185.36	1.060
		3	0.00	0.00	5.90	0.320
		4	0.00	0.00	5.01	0.170
		5	0.00	0.00	118.16	0.652
	5	1	0.00	0.00	–	6.079
		2	0.00	0.00	22.06	0.422
		3	0.00	0.00	–	25.542
		4	0.00	0.00	–	5.538
		5	0.00	0.00	46.30	0.837

[a] IBM ILOG CPLEX 12.1.0, run on an Intel Core i3 CPU with 4GB RAM, time limit of 900 s.
[b] Gurobi Optimizer 7.0.1, run on four Intel E5-2690 v4 2.6 GHz CPUs with 8GB RAM, time limit of 1 h. * means a trivial instance.

Table 8
Computational results for ZQLZ instances.

T	S	n	Unrestricted BRP						Restricted BRP					
			FB			TIR/N			Tanaka and Takii (2016)			TR		
			opt	ave	max	opt	ave	max	opt	ave	max	opt	ave	max
3	6	300	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.00	0.00
	7	300	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.00	0.00
	8	300	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.00	0.00
	9	300	**300**	0.00	0.03	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.00	0.00
	10	300	**300**	0.00	0.39	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.00	0.00
4	6	400	**400**	0.00	1.02	**400**	0.00	0.11	**400**	0.00	0.00	**400**	0.00	0.00
	7	400	**400**	0.03	7.17	**400**	0.00	0.11	**400**	0.00	0.00	**400**	0.00	0.00
	8	400	**400**	0.04	5.20	**400**	0.00	0.24	**400**	0.00	0.01	**400**	0.00	0.00
	9	400	**400**	1.56	575.30	**400**	0.02	7.31	**400**	0.00	0.10	**400**	0.00	0.04
	10	400	399	5.05	1800.00	**400**	0.01	2.12	**400**	0.00	0.57	**400**	0.00	0.30
5	6	500	499	5.66	1800.00	**500**	0.18	70.56	**500**	0.00	0.01	**500**	0.00	0.02
	7	500	495	29.01	1800.00	**500**	0.93	125.31	**500**	0.00	0.04	**500**	0.00	0.04
	8	500	486	79.80	1800.00	499	10.47	1800.00	**500**	0.00	1.03	**500**	0.00	0.59
	9	500	480	115.88	1800.00	499	9.05	1800.00	**500**	0.04	12.85	**500**	0.01	2.75
	10	500	459	224.76	1800.00	495	29.01	1800.00	**500**	0.05	13.72	**500**	0.01	0.62
6	6	600	519	304.32	1800.00	590	59.63	1800.00	**600**	0.00	0.50	**600**	0.00	0.16
	7	600	468	508.31	1800.00	572	122.31	1800.00	**600**	0.03	2.63	**600**	0.01	1.09
	8	600	372	813.26	1800.00	529	283.44	1800.00	**600**	0.38	85.53	**600**	0.20	38.78
	9	600	254	1131.59	1800.00	474	491.56	1800.00	599	6.26	1800.00	**600**	2.82	849.57
	10	600	208	1292.73	1800.00	432	611.54	1800.00	598	19.77	1800.00	599	8.84	1800.00

8. Conclusion

In this paper we proposed an exact algorithm for the unrestricted BRP with distinct priorities. For this purpose, we derived three types of dominance properties to eliminate unnecessary nodes in the search tree. We also improved the lower bound by Forster and Bortfeldt (2012). These enabled us to solve a larger number of benchmark instances to optimality than the existing exact algorithms in the literature. Some of the dominance properties are applicable also to the restricted BRP, and they contributed to improving our previous exact algorithm in Tanaka and Takii (2016) for this class of problem. Now, we can efficiently solve to optimality the unrestricted and restricted BRPs with distinct priorities. However, to the best of the authors' knowledge, no good exact algorithm exists for the unrestricted BRP with duplicate priorities. To handle duplicate priorities, the proposed dominance properties should be modified. It might be possible to derive specific properties to this class of problem. The proposed lower bound should also be extended. These topics are left for future research.

Acknowledgement

This work was supported by JSPS KAKENHI grant number JP15K01187.

References

Akyüz, M.H., Lee, C.Y., 2014. A mathematical formulation and efficient heuristics for the dynamic container relocation problem. Nav. Res. Logist. 61, 101–118.
Avriel, M., Penn, M., 1993. Exact and approximate solutions of the container ship stowage problem. Comput. Ind. Eng. 25, 271–274.
Avriel, M., Penn, M., Shpirer, N., 2000. Container ship stowage problem: complexity and connection to the coloring of circle graphs. Discrete Appl. Math. 103, 271–279.
Avriel, M., Penn, M., Shpirer, N., Witteboon, S., 1998. Stowage planning for container ships to reduce the number of shifts. Ann. Oper. Res. 76, 55–71.
Blasum, U., Bussieck, M.R., Hochstättler, W., Moll, C., Scheel, H.H., Winter, T., 1999. Scheduling trams in the morning. Math. Methods Oper. Res. 49, 137–148.
Bortfeldt, A., Forster, F., 2012. A tree search procedure for the container pre-marshalling problem. Eur. J. Oper. Res. 217, 531–540.
van Brink, M., 2014. A branch and price procedure for the container premarshalling problem. Lect. Notes Comput. Sci. 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. Lect. Notes Comput. Sci. 5482, 37–48.

Caserta, M., Schwarze, S., Voß, S., 2012. A mathematical formulation and complexity considerations for the blocks relocation problem. Eur. J. Oper. Res. 219, 96–104.
Caserta, M., Voß, S., 2009a. A corridor method-based algorithm for the pre-marshalling problem. Lect. Notes Comput. Sci. 5484, 788–797.
Caserta, M., Voß, S., 2009b. Corridor selection and fine tuning for the corridor method. Lect. Notes Comput. Sci. 5851, 163–175.
Caserta, M., Voß, S., Sniedovich, M., 2011. Applying the corridor method to a blocks relocation problem. OR Spectr. 33, 915–929.
Cheng, X., Tang, L., 2010. A scatter search algorithm for the slab stack shuffling problem. Lect. Notes Comput. Sci. 6145, 382–389.
Choe, R., Park, T., Oh, M.S., Kang, J., Ryu, K.R., 2011. Generating a rehandling-free intra-block remarshaling plan for an automated container yard. J. Intell. Manuf. 22, 201–217.
Dubrovsky, O., Levitin, G., Penn, M., 2002. A genetic algorithm with a compact solution encoding for the container ship stowage problem. J. Heuristics 8, 585–599.
Eskandari, H., Azari, E., 2015. Notes on mathematical formulation and complexity considerations for blocks relocation problem. Scientia Iranica Trans. E. (6) 2722–2728.
Expósito-Izquierdo, C., Melián-Batista, B., Moreno-Vega, J.M., 2014. A domain-specific knowledge-based heuristic for the blocks relocation problem. Adv. Eng. Infor. 28, 327–343.
Expósito-Izquierdo, C., Melián-Batista, B., Moreno-Vega, J.M., 2015. An exact approach for the blocks relocation problem. Expert Syst. Appl. 42, 6408–6422.
Expósito-Izquierdo, C., Melián-Batista, B., Moreno-Vega, M., 2012. Pre-marshalling problem: heuristic solution method and instances generator. Expert Syst. Appl. 39, 8337–8349.
Forster, F., Bortfeldt, A., 2012. A tree search procedure for the container relocation problem. Comput. Oper. Res. 39, 299–309.
Galle, V., Barnhart, C., Jaillet, P., 2017. A new binary formulation of the restricted container relocation problem based on a binary encoding of configurations. Eur. J. Oper. Res. doi:10.1016/j.ejor.2017.11.053. Available online.
Gupta, N., Nau, D.S., 1992. On the complexity of blocks-world planning. Artif. Intell. 56, 223–254.
Hottung, A., Tierney, K., 2016. A biased random-key genetic algorithm for the container pre-marshalling problem. Comput. Oper. Res. 75, 83–102.
Huang, S.H., Lin, T.H., 2012. Heuristic algorithms for container pre-marshalling problems. Comput. Ind. Eng. 62, 13–20.
Jin, B., Zhu, W., Lim, A., 2015. Solving the container relocation problem by an improved greedy look-ahead heuristic. Eur. J. Oper. Res. 240, 837–847.
Jovanovic, R., Tuba, M., Voß, S., 2017. A multi-heuristic approach for solving the pre–marshalling problem. Cent. Eur. J. Oper. Res. 25, 1–28.
Jovanovic, R., Voß, S., 2014. A chain heuristic for the blocks relocation problem. Comput. Ind. Eng. 75, 79–86.
Kim, B.I., Koo, J., Sambhajirao, H.P., 2011. A simplified steel plate stacking problem. Int. J. Prod. Res. 49, 5133–5151.
Kim, K.H., Hong, G.-P., 2006. A heuristic rule for relocating blocks. Comput. Oper. Res. 33, 940–954.
König, F.G., Lübbecke, M., Möhring, R., Schäfer, G., Spenke, I., 2007. Solutions to real-world instances of PSPACE-complete stacking. Lect. Notes Comput. Sci. 4698, 729–740.
Ku, D., Arthanari, T.S., 2016a. Container relocation problem with time windows for container departure. Eur. J. Oper. Res. 252, 1031–1039.
Ku, D., Arthanari, T.S., 2016b. On the abstraction method for the container relocation problem. Comput. Oper. Res. 68, 110–122.
Lee, Y., Chao, S.L., 2009. A neighborhood search heuristic for pre-marshalling export containers. Eur. J. Oper. Res. 196, 468–475.
Lee, Y., Hsu, N.Y., 2007. An optimization model for the container pre-marshalling problem. Comput. Oper. Res. 34, 3295–3313.
Lee, Y., Lee, Y.J., 2010. A heuristic for retrieving containers from a yard. Comput. Oper. Res. 37, 1139–1147.
Lehnfeld, J., Knust, S., 2014. Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. Eur. J. Oper. Res. 239, 297–312.
Lin, D.Y., Lee, Y.J., Lee, Y., 2015. The container retrieval problem with respect to relocation. Transp. Res. Part C 52, 132–143.
de Melo, d.S.M., Erdoğan, G., Battarra, M., Strusevich, V., 2017. The block retrieval problem. Eur. J. Oper. Res. doi:10.1016/j.ejor.2017.08.048. Available online.
Petering, M.E.H., Hussein, M.I., 2013. A new mixed integer program and extended look-ahead heuristic algorithm for the block relocation problem. Eur. J. Oper. Res. 231, 120–130.
Singh, K.A., Srinivas, Tiwari, S.M.K., 2004. Modelling the slab stack shuffling problem in developing steel rolling schedules and its solution using improved parallel genetic algorithms. Int. J. Prod. Econ. 91, 135–147.
Slaney, J., Thiébaux, S., 2001. Blocks world revisited. Artif. Intell. 125, 119–153.
Tanaka, S., Takii, K., 2016. A faster branch-and-bound algorithm for the block relocation problem. IEEE Trans. Autom. Sci. Eng. 13, 181–190.
Tang, L., Liu, J., Rong, A., Yang, Z., 2001. An effective heuristic algorithm to minimise stack shuffles in selecting steel slabs from the slab yard for heating and rolling. J. Oper. Res. Soc. 52, 1091–1097.
Tang, L., Liu, J., Rong, A., Yang, Z., 2002. Modelling and a genetic algorithm solution for the slab stack shuffling problem when implementing steel rolling schedules. Int. J. Prod. Res. 40, 1583–1595.
Tang, L., Ren, H., 2010. Modelling and a segmented dynamic programming-based heuristic approach for the slab stack shuffling problem. Comput. Oper. Res. 37, 368–375.
Tang, L., Zhao, R., Liu, J., 2012. Models and algorithms for shuffling problems in steel plants. Nav. Res. Logist. 59, 502–524.
Tierney, K., Pacino, D., Jensen, R.M., 2014. On the complexity of container stowage planning problems. Discrete Appl. Math. 169, 225–230.
Tierney, K., Pacino, D., Voß, S., 2017. Solving the pre-marshalling problem to optimality with a* and IDA*. Flex. Serv. Manuf. J. 29, 223–259.
Tricoire, F., Scagnetti, J., Beham, A., 2018. New insights on the block relocation problem. Comput. Oper. Res. 89, 127–139.
Ünlüyurt, T., Aydın, C., 2012. Improved rehandling strategies for the container retrieval process. J. Adv. Transp. 46, 378–393.
Voß, S., 2012. Extended mis-overlay calculation for pre-marshalling containers. Lect. Notes Comput. Sci. 7555, 86–91.
Wan, Y.w., Liu, J., Tsai, P.C., 2009. The assignment of storage locations to containers for a container stack. Nav. Res. Logist. 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. Eur. J. Oper. Res. 256, 90–101.
Wang, N., Zhang, Z., Lim, A., 2014. The stowage stack minimization problem with zero rehandle constraint. Lect. Notes Comput. Sci. 8482, 456–465.
Zehendner, E., Caserta, M., Feillet, D., Schwarze, S., Voß, S., 2015. An improved mathematical formulation for the blocks relocation problem. Eur. J. Oper. Res. 245, 415–422.
Zehendner, E., Feillet, D., 2014. A branch and price approach for the container relocation problem. Int. J. Prod. Res. 52, 7159–7176.
Zehendner, E., Feillet, D., Jaillet, P., 2017. An algorithm with performance guarantee for the online container relocation problem. Eur. J. Oper. Res. 259, 48–62.
Zhang, R., Jiang, Z.Z., Yun, W.Y., 2015. Stack pre-marshalling problem: a heuristic-guided branch-and-bound algorithm. Int. J. Ind. Eng. 22, 509–523.
Zhang, R., Liu, S., Kopfer, H., 2016. Tree search procedures for the blocks relocation problem with batch moves. Flex. Serv. Manuf. J. 28, 397–424.
Zhu, W., Qin, H., Lim, A., Zhang, H., 2012. Iterative deepening a* algorithms for the container relocation problem. IEEE Trans. Autom. Sci. Eng. 9, 710–722.