European Journal of Operational Research xxx (xxxx) xxx

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Production, Manufacturing, Transportation and Logistics

An exact approach to the restricted block relocation problem based on a new integer programming formulation

Shunji Tanaka[a,*], Stefan Voß[b]

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

ARTICLE INFO

Article history:
Received 14 August 2020
Accepted 30 March 2021
Available online xxx

Keywords:
OR in maritime industry
Combinatorial optimization
Block relocation problem
Integer programming

ABSTRACT

This study addresses the block(s) relocation problem (BRP), also known as the container relocation problem. This problem considers individually retrieving blocks piled up in tiers according to a predetermined order. When the block to be retrieved next is not at the top, we have to relocate the blocks above it because we can access only the topmost blocks. The objective is to retrieve all the blocks with the smallest number of relocations. In this study, a novel exact algorithm is proposed for the restricted BRP, a class of the problem where relocatable blocks are restricted. The proposed algorithm computes lower and upper bounds iteratively by solving the corresponding integer programming problems until the optimality gap is reduced to zero. The novelty of the algorithm lies in the formulations based on complete and truncated relocation sequences of the individual blocks. We examine the effectiveness of the proposed algorithm through computational experiments for benchmark instances from the literature. In particular, we report that, for the first time, all the instances with up to 100 blocks are solved to proven optimality.

1. Introduction

1.1. Background

The block(s) relocation problem (BRP), also known as the container relocation problem (CRP), is considered in our study. Piling up of blocks or containers vertically is a frequently encountered situation in warehouses, typically in container yards, owing to space limitations. However, this may cause difficulties when retrieving a block, that is, if the required block is not at the top, we have to relocate all blocks above it to other places (temporarily). The total number of such relocations depends on where blocks are relocated. For example, placing them on the block to be retrieved next incurs further relocations. This observation implies that careful and intelligent decisions regarding relocations can reduce unnecessary workload and, as a result, considerably improve the throughput of warehouses and container yards.

The BRP is formally described as follows. Suppose that N blocks of the same size are piled up in tiers, as in Fig. 1. Each column of blocks is called a stack, and S stacks are available. The set of all these S stacks composes a bay. The height of each stack is limited to T: the maximum number of blocks that can be placed in each of the stacks is T. The initial position of a block is specified by a slot (s, t), which denotes that it is placed in the tth tier of stack s. Each block is given an integer retrieval priority between 1 and G $(\leq N)$. The blocks are retrieved from the bay according to these priorities. More specifically, we should first retrieve blocks with priority 1, then those with priority 2, and so on. As only the topmost blocks can be retrieved, relocation of a block is inevitable when no block is retrievable. Thus, we must apply the following two operations:

Relocation. A block on top of a stack is moved to the top of another stack whose height is less than the prescribed limit.
Retrieval. The highest-priority block is removed from the bay if it is on top of a stack.

The objective function to be minimized is the total number of relocations required for retrieving all N blocks from the bay (note that the total number of retrievals is always equal to N).

The BRP can be classified from the following two aspects. The first concerns the block priorities. In the BRP with distinct prior-

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

https://doi.org/10.1016/j.ejor.2021.03.062
0377-2217/© 2021 Elsevier B.V. All rights reserved.

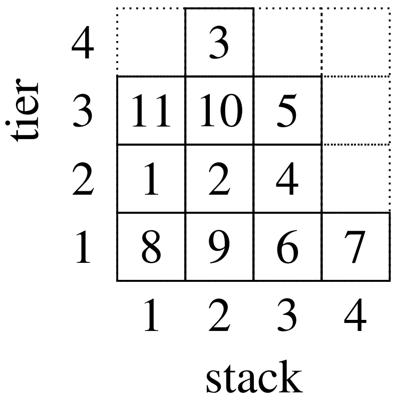


Fig. 1. Example of the BRP with $S = 4$, $T = 4$, and $G = N = 11$. The number in each block denotes the retrieval priority.

ities, each block is assigned a unique priority value, and $G = N$ holds. It follows that the retrieval order is uniquely determined. Fig. 1 illustrates an instance of this type of BRP. In contrast, in the BRP with duplicate priorities (or group priorities), $G \leq N$ is assumed. Therefore, a group of blocks may be assigned the same priority value, and the retrieval order among them is arbitrary. The other classification is with regard to a restriction on relocation. In the restricted BRP (RBRP), we can relocate only the topmost block above the block retrieved next. In the unrestricted BRP (UBRP), such a restriction is not imposed, and hence we can relocate any of the topmost blocks. In Fig. 1, block 1 should be retrieved next, so that we can relocate only block 11 above it in the RBRP. In the UBRP, any of blocks 11, 3, 5, and 7 can be relocated. Among these four types of BRP, this paper addresses the RBRP with distinct priorities. Our primary objective is to construct an efficient exact algorithm based on novel integer programming (IP) formulations.

1.2. Literature review

To the best of the authors' knowledge, the BRP as a combinatorial optimization problem was first studied by Kim and Hong (2006). Based on their optimization model, several researchers have proposed IP and MIP (mixed integer programming) formulations (Bacci, Mattia, & Ventura, 2020; Caserta, Schwarze, & Voß, 2012; Eskandari & Azari, 2015; Expósito-Izquierdo, Melián-Batista, & Moreno-Vega, 2014; Galle, Barnhart, & Jaillet, 2018; Lu, Zeng, & Liu, 2020; Petering & Hussein, 2013; de Melo da Silva, Toulouse, & Wolfler Calvo, 2018b; Wan, Liu, & Tsai, 2009; Zehendner, Caserta, Feillet, Schwarze, & Voß, 2015), exact algorithms (Expósito-Izquierdo et al., 2014; 2015; Ku & Arthanari, 2016; Quispe, Lintzmayer, & Xavier, 2018; de Melo da Silva, Erdoğan, Battarra, & Strusevich, 2018a; Tanaka & Mizuno, 2018; Tanaka & Takii, 2016; Tricoire, Scagnetti, & Beham, 2018; Ünlüyurt & Aydın, 2012; Zehendner & Feillet, 2014; Zhu, Qin, Lim, & Zhang, 2012), (meta)heuristics (Azari, Eskandari, & Nourmohammadi, 2017; Bacci, Mattia, & Ventura, 2019; Caserta, Schwarze, & Voß, 2009; 2011a; Caserta & Voß, 2009; Expósito-Izquierdo et al., 2014; Feillet, Parragh, & Tricoire, 2019; Forster & Bortfeldt, 2012; Jin, Zhu, & Lim, 2014; Jovanovic, Tuba, & Voß, 2019; Jovanovic & Voß, 2014; Lee & Lee, 2010; Lin, Lee, & Lee, 2015; Petering & Hussein, 2013; Ting & Wu, 2017; Tricoire et al., 2018; Ünlüyurt & Aydın, 2012; Zhang, Guan, Yuan, Chen, & Wu, 2020; Zhu et al., 2012), and so on. Among them, we first review studies relevant to dedicated exact algorithms and then review those on IP and MIP formulations because these are more relevant to our study.

Kim and Hong (2006) constructed a branch-and-bound algorithm for the BRP with duplicate priorities, where they proposed a simple lower bound on the total number of relocations. Intuitively, a block which is placed above another block with a higher (i.e., smaller number) priority should be relocated at least once. The researchers used the total number of such blocks (we refer to them as *blocking blocks*) as a lower bound on the total number of relocations. In Fig. 1, blocks 11, 3, 10, and 5 are blocking blocks and, hence, we obtain a lower bound of 4. Ünlüyurt and Aydın (2012) proposed a branch-and-bound algorithm for the RBRP with distinct priorities to minimize total crane working time. They computed a lower bound on the objective value based on the lower bound by Kim and Hong (2006). Expósito-Izquierdo et al. (2014) also utilized the lower bound by Kim and Hong (2006) and proposed an A* algorithm for the RBRP and UBRP with distinct priorities. Additionally, they constructed a branch-and-bound algorithm for the RBRP with distinct priorities (Expósito-Izquierdo, Melián-Batista, & Moreno-Vega, 2015). de Melo da Silva et al. (2018a) studied a particular class of the RBRP with duplicate priorities with only two groups and constructed a branch-and-bound algorithm employing the lower bound by Kim and Hong (2006).

A better lower bound LB3 was proposed by Zhu et al. (2012) for the RBRP with distinct priorities. Using it, they constructed an iterative deepening A* (IDA*) algorithm for the RBRP with distinct priorities. They also constructed an IDA* algorithm for the UBRP, and it employed the lower bound by Kim and Hong (2006). Ku and Arthanari (2016) proposed a bi-directional search method for the RBRP with distinct priorities, which utilizes not only LB3 but also a pre-computed pattern database for pruning a node. Tanaka and Takii (2016) improved and extended LB3 for the RBRP with distinct and duplicate priorities, which are referred to as LB4 and LB4e, respectively. They provided a branch-and-bound algorithm with iterative deepening where LB4 and LB4e are embedded. Quispe et al. (2018) proposed an IDA* algorithm for the RBRP with distinct priorities that utilizes a new lower bound and a pattern database. Their lower bound LB-LIS (where LIS denotes "longest increasing subsequence") does not dominate LB4 but enables faster computation in polynomial time. Bacci et al. (2019) proposed another lower bound, UBALB (where UBA denotes "unordered blocks assignment"), for the RBRP with distinct priorities. The primary difference of UBALB from the aforementioned lower bounds is that it does not ignore the stack height limit. They utilized this lower bound in their beam search algorithm, but did not consider an exact algorithm. Most recently, Zhang et al. (2020) improved LB4 for the RBRP with distinct priorities. However, the authors admit that it is not a good option to compute their lower bound, ELB4, at every node in the branch-and-bound algorithm, as it does not yield a significant improvement over LB4 in proportion to the extra computational effort.

Concerning the UBRP, Forster and Bortfeldt (2012) proposed a better lower bound than that by Kim and Hong (2006) for the UBRP, although they did not provide an exact algorithm. This lower bound is valid regardless of distinct or duplicate priorities. Tricoire et al. (2018) improved it for the UBRP with distinct priorities and incorporated it into a branch-and-bound algorithm. Tanaka and Mizuno (2018) proposed another improvement for the UBRP with distinct priorities. They constructed a branch-and-bound algorithm using the new lower bound and several dominance properties. They also improved the branch-and-bound algorithm for the RBRP with distinct priorities in Tanaka and Takii (2016) using the dominance properties. Lu et al. (2020) proposed a general framework for analyzing lower bounds for several variants of the UBRP, including the UBRP with distinct and duplicate priorities. Based on the framework, they proposed a new lower bound on the total number of relocations for the UBRP. They used it as an initial lower bound in a MIP-based iterative algorithm.

In addition to these dedicated exact algorithms, several IP and MIP formulations have been proposed. Wan et al. (2009) proposed a binary IP formulation for the RBRP with distinct priorities. Caserta et al. (2012) proposed two binary IP formulations for the RBRP and UBRP with distinct priorities. They also proved the NP-hardness of the BRP based on the results of König, Lübbecke, Möhring, Schäfer, and Spenke (2007) for a general stacking problem. Petering and Hussein (2013) proposed a MIP formulation for the UBRP with distinct priorities. Extending a formulation in Caserta et al. (2012), Zehendner and Feillet (2014) proposed a branch-and-price algorithm for the RBRP with distinct priorities. Expósito-Izquierdo et al. (2015) corrected an error in the formulation of the RBRP by Caserta et al. (2012). The error was also corrected by Zehendner et al. (2015), and the formulation was improved. Eskandari and Azari (2015) proposed valid inequalities to improve the corrected formulation. Galle et al. (2018) proposed another binary IP formulation for the RBRP with distinct priorities based on the binary encoding of bay configurations by Caserta et al. (2009). de Melo da Silva et al. (2018b) proposed a unified binary IP model for all four variants of the BRP as well as the premarshalling problem. Bacci et al. (2020) proposed a different binary IP formulation for the BRP with distinct priorities. To alleviate the difficulty in handling an exponential number of constraints in the formulation, they added only violated constraints on-the-fly in a branch-and-cut manner using the call-back function in a commercial IP solver. Lu et al. (2020) proposed MIP formulations for the UBRP and its variants. The underlying idea is similar to the one in Galle et al. (2018). They both introduced binary decision variables to express vertical relations of each pair of blocks in a bay: they become 1 if one is placed above the other in the same stack. The primary difference is that Lu et al. (2020) paired only blocks in adjacent tiers.

Among the existing exact approaches to the RBRP with distinct priorities, the most efficient would be that by Bacci et al. (2020). As in the other IP formulations, they decomposed the planning horizon into time periods according to the retrieval of blocks. Subsequently, they introduced two types of binary decision variables for each block. These variables express whether a block is placed in a stack at the beginning of each period, and whether a block is relocated from a stack during each period. Thus, the decision variables are indexed by stack, block, and period, which implies that their IP formulation has fewer decision variables than the other IP formulations. Indeed, the second-best IP formulation CRP-I (Galle et al., 2018) uses binary decision variables indexed by block, block, and period (note that the number of stacks is smaller than the number of blocks in non-trivial instances). However, the formulation by Bacci et al. (2020) requires an exponential number of constraints to model the relations between blocks in the same stack. Nevertheless, most of them are inactive for an optimal solution, which justifies applying the branch-and-cut approach. Instances with 50 or 60 blocks were successfully solved to optimality.

1.3. Purpose of this study

This study proposes new binary IP formulations of the RBRP with distinct priorities. To this end, we decompose a solution block-wise to express it as a combination of the relocation sequence of each block. Subsequently, the BRP is formulated as a problem to find the best relocation sequence of each block while satisfying coupling constraints. The primary difference from the existing IP formulations is that our formulation requires an exponential number of decision variables associated with relocation sequences. Instead of solving the large-size problem directly, we propose an iterative solution algorithm. This algorithm starts from a reduced IP formulation with a limited number of truncated relocation sequences and then solves it repeatedly with the sequences being extended gradually until the optimality is guaranteed. Some of the coupling constraints are also generated and added to the formulation on-the-fly. In our computational experiments, the proposed algorithm is applied to benchmark instances from the literature, proving to outperform the state-of-the-art branch-and-bound algorithm in Tanaka and Mizuno (2018) and the branch-and-cut algorithm in Bacci et al. (2020). In particular, it succeeds for the first time in solving instances with 100 blocks to proven optimality.

The remainder of the paper is organized as follows. In Section 2, the notation and definitions used in this paper are introduced. Then, possible relocation sequences of each block are described. In Section 3, a complete binary IP formulation is first proposed, then it is relaxed by considering truncated relocation sequences. A binary IP formulation for computing an upper bound is also proposed. Based on these formulations, Section 4 proposes an exact iterative algorithm for the RBRP. In Section 5, we describe the computational experiments that were conducted and examine the effectiveness of the proposed algorithm by comparing it with state-of-the-art exact algorithms. Finally, Section 6 summarizes the contributions of this study and discusses future research directions.

2. Preliminaries

In this section, we first introduce the notation and definitions used throughout this paper, and next investigate the relocation sequences of each block in detail.

2.1. Basic notation and definitions

A bay is composed of S stacks whose heights are limited to T, and N blocks are initially stacked in tiers. The sets of indices of stacks, tiers, and blocks are defined by $\mathcal{S} := \{1, 2, \ldots, S\}$, $\mathcal{T} := \{1, 2, \ldots, T\}$ and $\mathcal{N} := \{1, 2, \ldots, N\}$, respectively. Here, the tiers are numbered from the ground level up. The initial slot of block $i \in \mathcal{N}$ is given by (s_i, t_i) $(s_i \in \mathcal{S},\ t_i \in \mathcal{T})$, which means that block i is placed in the t_ith tier of stack s_i. Therefore, $(s_i, t_i) \neq (s_j, t_j)$ if $i \neq j$. Furthermore, for any block i with $t_i > 1$, there must be a block $j \in \mathcal{N}$ satisfying $(s_j, t_j) = (s_i, t_i - 1)$. In other words, there is no block that is floating in the air. As our problem is the RBRP with distinct priorities and, thus, $G = N$ holds, we assume without loss of generality that blocks are indexed according to their retrieval priorities. That is, block 1 is retrieved first, block 2 follows next, and so on.

The block with the smallest index in the current bay configuration is referred to as the *target block*, and the stack where the target block is placed is referred to as the *target stack*. The target block is the block to be retrieved next, but we can retrieve it only when it is on top of a stack. If this is not the case, we should relocate blocks above it to other stacks. Please note that in the RBRP, only blocks in the target stack can be relocated. It is easy to check that a block is relocated if and only if another block with a smaller index is placed below it. We refer to such a block as a *blocking block*. Fig. 2 depicts an example of a solution for the initial bay configuration in Fig. 1, where gray boxes are blocking blocks and double-line boxes are target blocks.

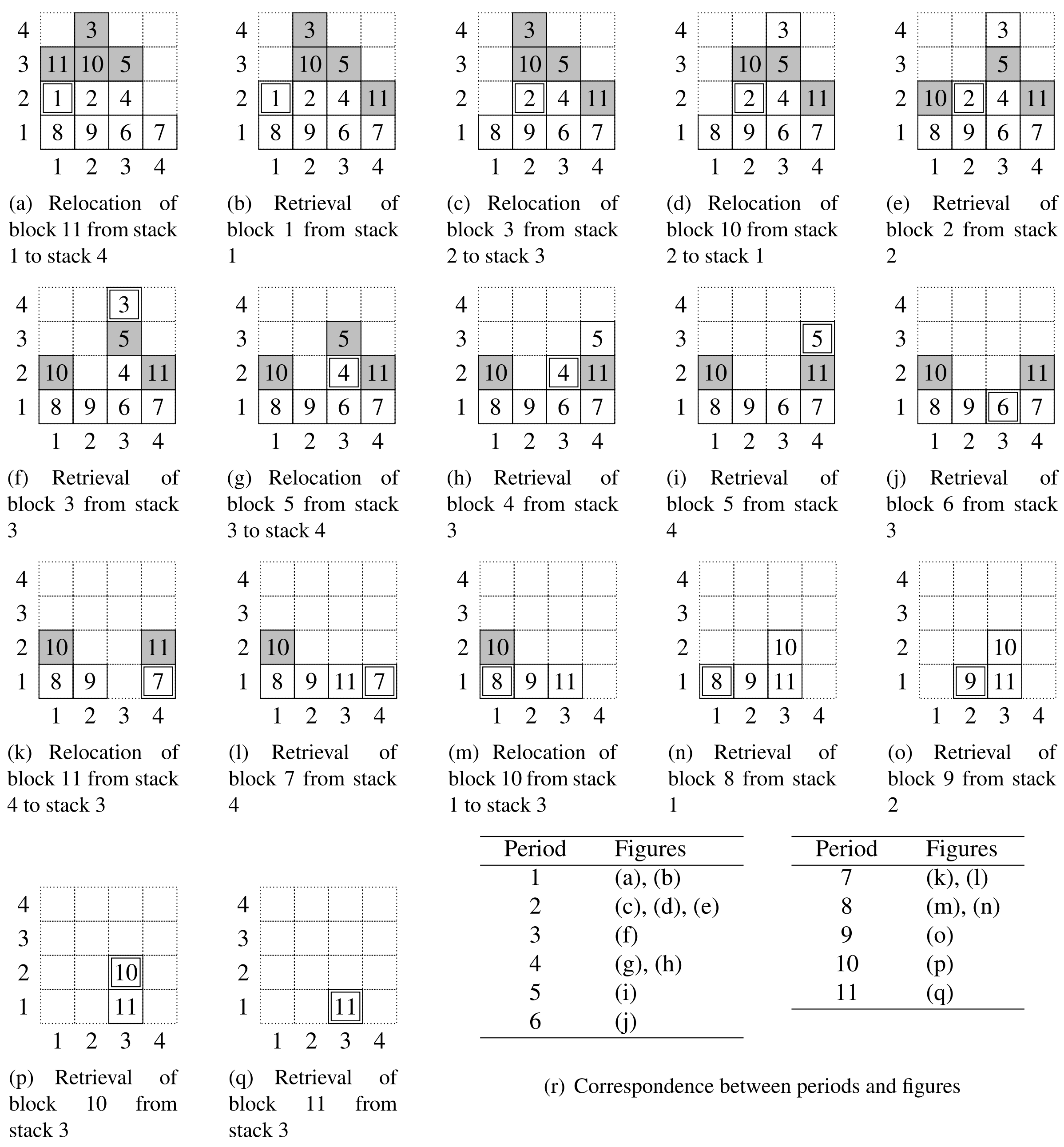
Let us define r_i as the smallest index of blocks below block i, including block i itself, in the initial bay configuration:

$$r_i := \min\left\{j \in \mathcal{N} \mid (s_j = s_i) \wedge (1 \le t_j \le t_i)\right\}. \tag{1}$$

Then, the set of indices of blocking blocks $\mathcal{B}$ in the initial bay configuration is defined as follows:

$$\mathcal{B} := \{j \in \mathcal{N} \mid r_j < j\}. \tag{2}$$

In the example of Fig. 1, $\mathcal{B}$ is given by $\mathcal{B} = \{3, 5, 10, 11\}$ (see also Fig. 2(a)). As only blocks in $\mathcal{B}$ are relocated and the other blocks in $\mathcal{N} \setminus \mathcal{B}$ are never relocated, it implies that the RBRP has to determine when and to which stack each block in $\mathcal{B}$ is to be relocated.



(a) Relocation of block 11 from stack 1 to stack 4
(b) Retrieval of block 1 from stack 1
(c) Relocation of block 3 from stack 2 to stack 3
(d) Relocation of block 10 from stack 2 to stack 1
(e) Retrieval of block 2 from stack 2
(f) Retrieval of block 3 from stack 3
(g) Relocation of block 5 from stack 3 to stack 4
(h) Retrieval of block 4 from stack 3
(i) Retrieval of block 5 from stack 4
(j) Retrieval of block 6 from stack 3
(k) Relocation of block 11 from stack 4 to stack 3
(l) Retrieval of block 7 from stack 4
(m) Relocation of block 10 from stack 1 to stack 3
(n) Retrieval of block 8 from stack 1
(o) Retrieval of block 9 from stack 2
(p) Retrieval of block 10 from stack 3
(q) Retrieval of block 11 from stack 3

Period	Figures
1	(a), (b)
2	(c), (d), (e)
3	(f)
4	(g), (h)
5	(i)
6	(j)
7	(k), (l)
8	(m), (n)
9	(o)
10	(p)
11	(q)

(r) Correspondence between periods and figures

Fig. 2. Example of a solution with six relocations ($S = 4$, $T = 4$, $N = 11$). Gray boxes are blocking blocks and double-line boxes are target blocks.

Let us decompose the planning horizon into N periods according to the target block. In period p, blocking blocks above target block p are relocated, followed by the retrieval of block p. In Fig. 2, period 1 is composed of (a) and (b), period 2 is composed of (c), (d), and (e), and so forth (see Fig. 2(c)). This example implies that a block $i \in \mathcal{B}$ is invariant in periods $1, \dots, (r_i - 1)$. A block $i \in \mathcal{N} \setminus \mathcal{B}$ is also invariant in periods $1, \dots, (r_i - 1)$ because it is retrieved directly from its initial slot (s_i, t_i) in period $i = r_i$. Let us denote the bay configuration composed only of blocks that are invariant in periods $1, \dots, p$ by $\mathcal{C}^{(p)}$. Fig. 3 illustrates these configurations. Clearly, $\mathcal{C}^{(0)}$ is identical to the initial bay configuration. Thus, $i \in \mathcal{C}^{(p)}$ if block i is in $\mathcal{C}^{(p)}$, i.e., $p < r_i$, and $i \notin \mathcal{C}^{(p)}$ otherwise. The number of blocks in stack s in $\mathcal{C}^{(p)}$ is denoted by $h_s^{(p)}$. We define the *priority of a stack* as the minimum index of blocks in that stack (∞ is assumed if the stack is empty). Let $q_s^{(p)}$ denote the priority of stack s in $\mathcal{C}^{(p)}$. As blocks in $\mathcal{C}^{(p)}$ are invariant in periods $1, \dots, p$, the number of blocks in stack s is at least $h_s^{(p)}$ in period p. Similarly, the priority of stack s is at most $q_s^{(p)}$ in period p. In Fig. 2(b), the number of blocks in stack 3 is 4, which is larger than $h_3^{(2)} = 3$. The priority of stack 3 is 3, which is smaller than $q_3^{(2)} = 4$ (see Fig. 3 c).

If $p \in \mathcal{N} \setminus \mathcal{B}$, block p is retrieved from stack s_p in period p. This means that all relocations in period p should be from stack s_p. In contrast, if $p \in \mathcal{B}$, the source stack of the relocations in period p depends on where the blocking block p is placed. Hence, we denote the target stack in period p by $\tau^{(p)}$, which is defined as

$$\tau^{(p)} := \begin{cases} s_p, & p \in \mathcal{N} \setminus \mathcal{B}, \\ 0, & p \in \mathcal{B}. \end{cases} \tag{3}$$

Hence, $\tau^{(p)} = 0$ implies that the target stack in period p is not predetermined. We define the set of indices of potential destination

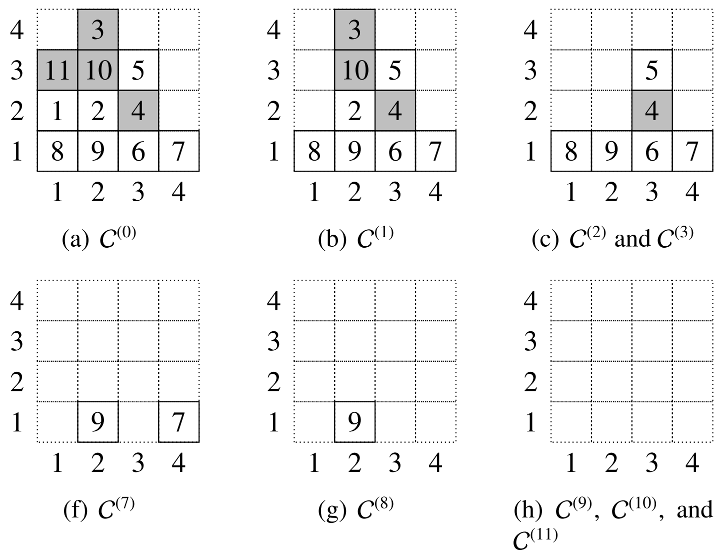

(a) $C^{(0)}$
(b) $C^{(1)}$
(c) $C^{(2)}$ and $C^{(3)}$
(d) $C^{(4)}$ and $C^{(5)}$
(e) $C^{(6)}$
(f) $C^{(7)}$
(g) $C^{(8)}$
(h) $C^{(9)}$, $C^{(10)}$, and $C^{(11)}$

Fig. 3. Bay configurations $C^{(p)}$ composed only of invariant blocks in periods $1, \dots, p$.

stacks of a relocation in period p by

$$\mathcal{L}^{(p)} := \{s \in \mathcal{S} \mid h_s^{(p)} < T\} \setminus \{\tau^{(p)}\}. \tag{4}$$

More specifically, for stack s to become the destination of a relocation, it should not be full or the predetermined target stack.

2.2. Relocation sequence of an individual block

A relocation from stack s to stack d in period p is denoted by a 3-tuple $(p,\ s,\ d)$. For the ease of notation, the retrieval of block p from stack s (in period p) is denoted by $(p,\ s,\ s)$. A solution of the RBRP can be expressed uniquely by a single sequence of relocations, which can be decomposed into block-wise sequences. For example, the relocation sequences of blocking blocks 11, 3, 10, and 5 in Fig. 2 are expressed by:

- block 11, (1, 1, 4), (7, 4, 3), (11, 3, 3);
- block 3, (2, 2, 3), (3, 3, 3);
- block 10, (2, 2, 1), (8, 1, 3), (10, 3, 3);
- block 5, (4, 3, 4), (5, 4, 4).

The source stack s of a relocation $(p,\ s,\ d)$ in a relocation sequence of a block is identical to the destination stack of the immediately preceding relocation. In particular, the first relocation of block i is always from stack $\tau^{(r_i)} = s_i$ in period r_i. The destination stack d of $(p,\ s,\ d)$ is required to belong to $\mathcal{L}^{(p)}$. This relocation can be the final relocation for block i if d satisfies $i < q_d^{(p)}$. Otherwise, block i is relocated again. Even in the case of $i < q_d^{(p)}$, block i may be relocated again because of the blocks below block i in stack d, which are relocated to stack d before the relocation of block i in period p. Noting that only blocks j with $p < j < i$ matter, we consider the set of indices of blocking blocks such that (a) they are already relocated, but not yet retrieved at the end of period p, (b) they can be present below block i after block i is relocated in period p, and (c) their indices are smaller than i. This set $\mathcal{Q}_i^{(p)}$ ($i \in \mathcal{B}$, $r_i \le p < i$) is defined as

$$\mathcal{Q}_i^{(p)} := \begin{cases} \{j \in \mathcal{B} \mid (p < j < i) \wedge (r_j < p)\}, & p > r_i, \\ \{j \in \mathcal{B} \mid (p < j < i) \wedge ((r_j < p) \vee ((r_j = p) \wedge (t_j > t_i)))\}, & p = r_i, \end{cases} \tag{5}$$

In the case of $p > r_i$, block j cannot be placed below block i if $r_j = p$, that is, if the first relocation of block j occurs in period p. As $p > r_i$ implies that block i is relocated before period p, it should be located above block j at the beginning of period p. Therefore, block j is relocated after block i in period p and can never be below block i at the end of this period. We thus consider only blocks j satisfying $r_j < p$ in (5) for $p > r_i$. In the case of $p = r_i$, we should also consider blocks j satisfying both $r_j = p$ and $t_j > t_i$ as in (5). If $t_j > t_i$, block j is above block i at the beginning of period i and can be below block p at the end of the period by relocating them to the same stack.

Noting that only one block can be relocated to stack d in period p if $h_d^{(p)} = T - 1$, we can verify that for a block $i \in \mathcal{B}$, the period p' of the relocation immediately after $(p,\ s,\ d)$ satisfies $p' \in \mathcal{P}_{id}^{(p)}$ ($d \in \mathcal{L}^{(p)}$, $r_i \le p < i$), where

$$\mathcal{P}_{id}^{(p)} := \begin{cases} \emptyset, & (i < q_d^{(p)}) \wedge (h_d^{(p)} = T - 1), \\ \mathcal{Q}_i^{(p)}, & (i < q_d^{(p)}) \wedge (h_d^{(p)} < T - 1), \\ \{q_d^{(p)}\}, & (i > q_d^{(p)}) \wedge (h_d^{(p)} = T - 1), \\ \{j \in \mathcal{Q}_i^{(p)} \mid j < q_d^{(p)}\} \cup \{q_d^{(p)}\}, & (i > q_d^{(p)}) \wedge (h_d^{(p)} < T - 1). \end{cases} \tag{6}$$

Here, the empty set implies that (p, s, d) is the last relocation of block i, and it is retrieved next. Using $\mathcal{P}_{id}^{(p)}$, we can generate all possible relocation sequences from the start to finish. Let us consider block 10 in Fig. 2. Before its first relocation, blocks 11 and 3 have been already relocated. However, only block 3 affects the relocation periods of block 10, and $\mathcal{Q}_{10}^{(2)} = \{3\}$. If the first relocation of block 10 is (2,2,1), the next relocation period is 3 when block 3 is relocated to stack 1, and 8 $(= q_1^{(2)})$ otherwise. Thus, we know that $\mathcal{P}_{10,1}^{(2)} = \{3, 8\}$, which is given by the fourth condition in (6). In the case of (2,2,3), block 10 is always relocated in period 4 $(= q_3^{(2)})$. It is because stack 3 can accept at most one block in period 2 $(T - h_3^{(2)} = 4 - 3 = 1)$, and block 3 cannot be placed below block 10. From $10 > q_3^{(2)} = 4$ and $h_3^{(2)} = 3 = T - 1$, $\mathcal{P}_{10,3}^{(2)}$ is defined as $\mathcal{P}_{10,3}^{(2)} = \{4\}$ by the third condition in (6). We obtain $\mathcal{P}_{10,4}^{(2)} = \{3, 7\}$ for (2,2,4) in a similar manner to the case of (2,2,1). Therefore, all the possible patterns of the first two relocations of block 10 are (A) $(2, 2, 1), (3, 1, d_1)$, (B) $(2, 2, 1), (8, 1, d_2)$, (C) $(2, 2, 3), (4, 3, d_3)$, (D) $(2, 2, 4), (3, 4, d_4)$, and (E) $(2, 2, 4), (7, 4, d_5)$.

The notation and definitions introduced in this section as well as those found in the next section are summarized as follows.

S:	Number of stacks.
T:	Stack height limit (the maximum number of blocks that can be placed in each stack).
N:	Number of blocks.
$\mathcal{N}$:	Set of indices of blocks. $\mathcal{N} := \{1, 2, \ldots, N\}$.
$\mathcal{S}$:	Set of indices of stacks. $\mathcal{S} := \{1, 2, \ldots, S\}$.
$\mathcal{T}$:	Set of indices of tiers. $\mathcal{T} := \{1, 2, \ldots, T\}$.
(s_i, t_i):	Initial slot of block i. Block i is placed in the t_ith tier of stack s_i.
r_i	Minimum index of blocks below block i (including block i itself) in the initial bay configuration. $r_i := \min\{j \in \mathcal{N} \mid (s_j = s_i) \wedge (1 \leq t_j \leq t_i)\}$.
$\mathcal{B}$:	Set of indices of blocking blocks in the initial bay configuration. $\mathcal{B} := \{i \in \mathcal{N} \mid r_i < i\}$.
$(p,\ s,\ d)$:	Relocation. A block is relocated from stack s to stack d in period p.
$(p,\ s,\ s)$:	Retrieval. A block (block p) is retrieved from stack s in period p.
$(p,\ s,\ \bullet)$:	Unfixed relocation. The destination stack is arbitrary.
$\mathcal{C}^{(p)}$:	Bay configuration composed only of blocks that are invariant in periods $1, \ldots, p$. $i \in \mathcal{C}^{(p)} \Leftrightarrow r_i > p$.
$h_s^{(p)}$:	Number of blocks in stack s in configuration $\mathcal{C}^{(p)}$. $h_s^{(p)} := \lvert\{i \in \mathcal{N} \mid (s_i = s) \wedge (r_i > p)\}\rvert$.
$q_s^{(p)}$	Priority of stack s in configuration $\mathcal{C}^{(p)}$. The priority of a stack is the minimum index of blocks placed therein. $q_s^{(p)} := \min\{i \in \mathcal{N} \mid (s_i = s) \wedge (r_i > p)\}$. $q_s^{(p)} := \infty$ if $h_s^{(p)} = 0$.
$\tau^{(p)}$:	Predetermined target stack in period p. $\tau^{(p)} := s_p$ if $p \in \mathcal{N} \setminus \mathcal{B}$, and $\tau^{(p)} := 0$ otherwise.
$\mathcal{L}^{(p)}$:	Set of indices of potential destination stacks of a relocation in period p. Set of indices of slack stacks in $\mathcal{C}^{(p)}$ except $\tau^{(p)}$. $\mathcal{L}^{(p)} := \{s \in \mathcal{S} \mid h_s^{(p)} < T\} \setminus \{\tau^{(p)}\}$.
$\mathcal{Q}_i^{(p)}$:	Set of indices of blocking blocks that may be present below block i at the end of period p and may affect the next relocation of block i, if block i is relocated in period p.
$\mathcal{P}_{id}^{(p)}$:	Set of possible periods of the relocation immediately after $(p,\ s,\ d)$ for block i.
$\mathcal{E}_i$	Set of all possible relocation sequences of block i.
$\mathcal{E}'_i$	Set of all possible relocation sequences of block i including truncated ones.
$\mathcal{G}$	Set of all pairs of relocation sequences that conflict with each other.
$\mathcal{G}'$	Set of all pairs of relocation sequences including truncated ones that conflict with each other.
$R(\pi)$	Number of relocations in the relocation sequence π.
$R'(\pi)$	Lower bound on the number of relocations in relocation sequences represented by truncated relocation sequence π. $R'(\pi) = R(\pi)$ if π is complete.
$P_s^{(p)}(\pi)$	Binary function that becomes 1 if and only if relocation sequence π of block i places block i in stack s at the end of period p.

3. IP formulation based on relocation sequences

It has been observed that relocation sequences of blocks in $\mathcal{B}$ compose a solution in the preceding section. Based on this observation, we first propose a complete binary IP formulation to find an optimal set of relocation sequences. Unfortunately, solving this formulation directly is not easy because of the large number of decision variables and constraints. To alleviate this, we introduce a relaxed IP formulation where truncated relocation sequences are permitted.

3.1. IP formulation of the RBRP

Let $\mathcal{E}_i$ denote the set of all possible relocation sequences of block $i \in \mathcal{B}$ generated in the manner explained in the preceding section, and π_{ik} the kth relocation sequence in $\mathcal{E}_i$. These sequences may depend on each other because of relocation periods and stack capacity. With regard to the relocation periods, the conditions that a pair of relocation sequences π_{ik} and π_{jl} $(j < i)$ should satisfy are summarized as follows.

Proposition 1. *A pair of relocation sequences $\pi_{ik} \in \mathcal{E}_i$ and $\pi_{jl} \in \mathcal{E}_j$ $(j < i)$ in a feasible solution satisfies the following condition:*

(a) *Block i is relocated by π_{ik} in period j if and only if block j is placed below block i by π_{jl} at the beginning of period j.*

Clearly, this condition, which is hereafter referred to as Condition (a), comes from the fact that our problem is the RBRP: block i is relocated when block j is the target block, if and only if block i is placed above block j. If Condition (a) is not satisfied, π_{ik} and π_{jl} will not coexist in a feasible solution. Let us consider the first two relocations of block 10 in the example in the preceding section. In (A), the first relocation of block 3 cannot be (2,2,3) or (2,2,4) because block 3 should be placed below block 10 in stack 1 before the first relocation (2,2,1) of block 10. Similarly, in (D), it cannot be (2,2,1) or (2,2,3). These are enforced by the “only if” part of Condition (a). On the other hand, the “if” part is applied to (B) and (E), where block 10 is not relocated in period 3. In (B), the first relocation of block 3 cannot be (2,2,1) because block 3 should not be placed below block 10 in stack 1. Likewise, it cannot be (2,2,4) in (E).

Unlike these patterns, the dependence of (C) on other sequences is not direct. However, because of the height limit of stack 3, (C) becomes infeasible if the first relocation of block 3 is (2,2,3). This example illustrates the necessity of stack capacity constraints. Noting that the capacity of stack s at the end of period p is given by $(T - h_s^{(p)})$ for non-invariant blocks, we can summarize the stack capacity constraints as follows.

Proposition 2. *For each $p \in \{1, 2, \ldots, N - T - 1\}$ and $s \in \mathcal{L}^{(p)}$, the total number of blocks $i \in \mathcal{B}$ that are placed in stack s at the end of period p does not exceed $(T - h^{(p)})$.*

Here, we do not impose any capacity constraint on stack $\tau^{(p)}$ $(\neq 0)$ because it is the source stack of relocations and can never violate the capacity constraint at the end of period p. Additionally, we exclude periods $(N - T + 1), (N - T + 2), \ldots, N$ because at most T blocks remain in the bay, and the capacity constraints are always satisfied. We can further exclude period $(N - T)$, although $(T + 1)$ blocks remain in the bay, because at most T blocks out of them are present in the stacks other than the target stack at the end of the period and can never violate the capacity constraints (note that the target block $(N - T)$ is placed in the target stack).

To model the problem as an integer program, we introduce binary decision variables x_{ik} $(i \in \mathcal{B},\ 1 \leq k \leq |\mathcal{E}_i|)$ expressing whether π_{ik} is used:

$$x_{ik} = \begin{cases} 1, & \text{if } \pi_{ik} \in \mathcal{E}_i \text{ is used,} \\ 0, & \text{otherwise,} \end{cases} \quad 1 \leq k \leq |\mathcal{E}_i|,\ i \in \mathcal{B}. \tag{7}$$

The fact that exactly one sequence is chosen for each block $i \in \mathcal{B}$ leads us to the following assignment constraints:

$$\sum_{1 \leq k \leq |\mathcal{E}_i|} x_{ik} = 1, \quad i \in \mathcal{B}. \tag{8}$$

Let us define $\mathcal{G}$ as the set of all pairs (π_{jl}, π_{ik}) $(j < i)$ that do not satisfy Condition (a). Then, conflict constraints between relocation sequences are simply formulated as

$$x_{jl} + x_{ik} \leq 1, \quad (\pi_{jl}, \pi_{ik}) \in \mathcal{G}. \tag{9}$$

Using (8), we can reduce the number of constraints by summing (9) up for each combination of i, j, and k:

$$\sum_{\substack{1 \leq l \leq |\mathcal{E}_j| \\ (\pi_{jl}, \pi_{ik}) \in \mathcal{G}}} x_{jl} + x_{ik} \leq 1, \quad 1 \leq k \leq |\mathcal{E}_i|,\ i \in \mathcal{B},\ j \in \mathcal{B},\ j < i. \tag{10}$$

To express the capacity constraints in Proposition 2, let us define $P_s^{(p)}(\pi_{ik})$ ($i \in \mathcal{B}$, $1 \le k \le |\mathcal{E}_i|$, $s \in \mathcal{L}^{(p)}$, $1 \le p \le N - T - 1$) as

$$P_s^{(p)}(\pi_{ik}) := \begin{cases} 1, & \text{if relocation sequence } \pi_{ik} \text{ places block } i \\ & \text{in stack } s \text{ at the end of period } p, \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

Then, the capacity constraints become

$$\sum_{\substack{i \in \mathcal{B} \\ r_i \le p < i}} \sum_{1 \le k \le |\mathcal{E}_i|} P_s^{(p)}(\pi_{ik}) x_{ik} \le T - h_s^{(p)}, \quad s \in \mathcal{L}^{(p)}, \ 1 \le p \le N - T - 1. \tag{12}$$

In (12), only blocks $i \in \mathcal{B}$ satisfying $r_i \le p < i$ are considered because they are invariant in periods $p < r_i$ and already retrieved in periods $p > i$. In period $p = i$, block i is retrieved at the end of this period and does not affect the capacity constraints.

Finally, the objective function is given by

$$\sum_{i \in \mathcal{B}} \sum_{1 \le k \le |\mathcal{E}_i|} R(\pi_{ik}) x_{ik}, \tag{13}$$

where $R(\pi_{ik})$ denotes the number of relocations in the sequence π_{ik}. In summary, the IP formulation (IP) of the RBRP is written as

$$\begin{aligned} &\min (13), \\ &\text{s.t.} (8), (10), (12), \\ &\quad x_{ik} \in \{0, 1\}, \quad 1 \le k \le |\mathcal{E}_i|, \ i \in \mathcal{B}. \end{aligned} \tag{14}$$

3.2. Relaxed formulation with truncated relocation sequences

The major drawback of (IP) is the exponential number of relocation sequences, resulting in an exponential number of decision variables, which also implies an exponential number of conflict constraints. To reduce the number of relocation sequences, we assume that one *truncated* sequence represents a group of sequences. In the example in Section 2.2, all relocation sequences of block 10 that fall in pattern (A) can be represented by a truncated sequence $(2, 2, 1), (3, 1, \bullet)$, where "$\bullet$" denotes that the destination stack of a relocation is arbitrary. This sequence does not specify the location of block 10 from period 3. In other words, we do not care whether block 10 is relocated after the first two relocations. On each truncated relocation sequence, we impose conflict and capacity constraints, provided that all the complete sequences represented by it should satisfy the imposed constraints. This implies that only part of the original constraints are considered. The objective function is also relaxed using its lower bound: the number of relocations $R(\pi_{ik})$ is replaced with its lower bound $R'(\pi_{ik})$ for a truncated sequence π_{ik}. The relaxation of (IP) derived in this manner is denoted by (RP).

Next, let us formulate (RP). We can easily derive the counterparts of the capacity constraints (12) for truncated sequences. We only need to ignore the effect of the corresponding block on the stack capacity after the last relocation in each truncated sequence. Therefore, we assume that $P^{(p)}(\pi_{ik})$ is always zero for $p \ge p'$ if the last relocation of a truncated π_{ik} is in period p'. It does not change the definition of $P^{(p)}(\pi_{ik})$ in (11) and, thus, we use the same notation to express the location of a block in a truncated sequence.

To derive the conflict constraints corresponding to (10), we should detect a conflict between possibly truncated relocation sequences. Hence, we introduce the following proposition in addition to Proposition 1.

Proposition 3. *A pair of relocation sequences π_{ik} and π_{jl} ($j < i$) in a feasible solution satisfies the following conditions:*

(b) *If both blocks i and j are relocated in period p ($p < j$), they are relocated from the same stack.*

(c) *If block i is relocated in period p ($p < j$) and block j is above block i at the beginning of this period, block j is also relocated in period p. Conversely, if block j is relocated in period p ($p < j$) and block i is above block j at the beginning of this period, block i is also relocated in period p.*

Condition (b) claims that if two blocks are relocated in the same period, their source stacks should be the target stack. Condition (c) claims that to relocate a block, those placed above it should also be relocated. These conditions are redundant for complete relocation sequences because they are guaranteed by Condition (a) in Proposition 1 for blocks i and p and for blocks j and p. If blocks i and j are relocated in period p as in Condition (b), block p is required to be below blocks i and j at the beginning of period p from Condition (a), which ensures that blocks i and j are in the same stack. Condition (a) guarantees Condition (c) in a similar manner. However, if relocation sequences of block p are truncated, Condition (a) alone may be insufficient for detecting a conflict between blocks i and j. Therefore, Conditions (b) and (c) are applied together with Condition (a) in (RP).

Let $\mathcal{E}'_i$ be a reduced set of relocation sequences of block i composed of truncated sequences (and possibly complete sequences). The set of all conflicting pairs according to Conditions (a)–(c) is denoted by $\mathcal{G}'$. Furthermore, let us denote a lower bound on the number of relocations in the relocation sequences represented by a truncated π_{ik} by $R'(\pi_{ik})$. For a complete π_{ik}, we assume that $R'(\pi_{ik}) := R(\pi_{ik})$. Then, by replacing $\mathcal{E}_i$, $\mathcal{G}$, and $R(\pi_{ik})$ in (IP) with $\mathcal{E}'_i$, $\mathcal{G}'$, and $R'(\pi_{ik})$, respectively, we obtain the formulation of (RP):

$$\min \sum_{i \in \mathcal{B}} \sum_{1 \le k \le |\mathcal{E}'_i|} R'(\pi_{ik}) x_{ik}, \tag{15}$$

$$\text{s.t.} \sum_{1 \le k \le |\mathcal{E}'_i|} x_{ik} = 1, \quad i \in \mathcal{B}. \tag{16}$$

$$\sum_{\substack{1 \le l \le |\mathcal{E}'_j| \\ (\pi_{jl}, \pi_{ik}) \in \mathcal{G}'}} x_{jl} + x_{ik} \le 1, \quad 1 \le k \le |\mathcal{E}'_i|, \ i \in \mathcal{B}, \ j \in \mathcal{B}, \ j < i. \tag{17}$$

$$\sum_{\substack{i \in \mathcal{B} \\ r_i \le p < i}} \sum_{1 \le k \le |\mathcal{E}'_i|} P_s^{(p)}(\pi_{ik}) x_{ik} \le T - h_s^{(p)}, \quad s \in \mathcal{L}^{(p)}, \ 1 \le p \le N - T - 1, \tag{18}$$

$$x_{ik} \in \{0, 1\}, \quad 1 \le k \le |\mathcal{E}'_i|, \ i \in \mathcal{B}. \tag{19}$$

We explain how to generate $\mathcal{E}'_i$ in Section 4.

3.3. Heuristic using a limited number of relocation sequences

For complete relocation sequences in (RP), the constraints, as well as the coefficients in the objective function, are the same as those in (IP). It follows that a feasible (respectively, optimal) solution of (RP) is also feasible (respectively, optimal) for (IP), provided that it is composed only of complete relocation sequences. This observation enables an upper bound computation method. To obtain a heuristic solution, we derive (UP) from (RP) by removing all truncated sequences from $\mathcal{E}'_i$. Alternatively, we may add constraints $x_{ik} = 0$ for truncated sequences π_{ik} to (RP). If this (UP) is feasible, it yields a heuristic solution of (IP) and, thus, an upper bound on the objective value. This upper bound is utilized in the exact iterative algorithm proposed in Section 4.

4. Exact algorithm based on IP formulations

In this section, an exact iterative algorithm is constructed that employs (RP) and (UP) for lower and upper bounding, respectively.

4.1. Outline of the algorithm

The formulation (RP) yields a relaxation of the original problem (IP). As explained in the previous section, an optimal solution of (RP) is also optimal for (IP) if it is composed only of complete relocation sequences. This fact motivates us to construct an iterative solution algorithm in which the truncated relocation sequences in the current solution are expanded until a solution without the truncated sequences is obtained. An outline of the algorithm is summarized in Algorithm 1. First, the sets of relocation sequences, $\mathcal{E}_i'$ are initialized in line 1. The set of conflicts $\mathcal{G}'$ is initialized accordingly, and (RP) is defined. In line 2, the initial upper bound is computed by the greedy heuristic in Tanaka and Takii (2016), a slightly simplified version of that used in Zhu et al. (2012). In the main loop (lines 3–18), (RP) is solved in line 4. If its solution is composed only of complete relocation sequences, it is optimal for (IP), and the loop is terminated in line 7. If $f^{\mathrm{R}} = f^{\mathrm{U}}$, the optimality gap is zero, and the loop is terminated in line 9. In lines 10–12, truncated relocation sequences composing the solution of (RP) are expanded and $\mathcal{E}_i'$ are updated. Subsequently, the set of conflicts $\mathcal{G}'$ is also updated and, using them, (RP) is updated in line 13. In line 14, (UP) is generated from (RP) by eliminating truncated relocation sequences. If the solution of (UP) is better than the current best solution, the latter is replaced by the former in line 16. The loop is terminated in line 18 if it also closes the optimality gap.

Algorithm 1 Proposed exact algorithm

```
 1: Initialize $\mathcal{E}_i'$ for $i \in \mathcal{B}$, $\mathcal{G}'$, and (RP)
 2: Find a heuristic solution. $\pi_i^{\mathrm{U}} \leftarrow$ (relocation sequence of block $i \in \mathcal{B}$), $f^{\mathrm{U}} \leftarrow$ (objective value)
 3: loop
 4:     Solve (RP). $\pi_i^{\mathrm{R}} \leftarrow$ (optimal relocation sequenceof block $i$), $f^{\mathrm{R}} \leftarrow$ (optimal value)
 5:     if $\pi_i^{\mathrm{R}}$ is complete for all $i \in \mathcal{B}$ then    ▷ (RP) provides a feasible solution for the original RBRP
 6:         $\pi_i^{\mathrm{U}} \leftarrow \pi_i^{\mathrm{R}}$ for $i \in \mathcal{B}$, $f^{\mathrm{U}} \leftarrow f^{\mathrm{R}}$ ▷ $\pi_i^{\mathrm{R}}$ is an optimal sequence of block $i \in \mathcal{B}$
 7:         break
 8:     if $f^{\mathrm{R}} = f^{\mathrm{U}}$ then    ▷ No optimality gap
 9:         break
10:     for $i \in \mathcal{B}$ do
11:         if $\pi_i^{\mathrm{R}}$ is truncated then
12:             Generate relocation sequences by expanding $\pi_i^{\mathrm{R}}$ and add them to $\mathcal{E}_i'$
13:     Update $\mathcal{G}'$ and (RP)
14:     Generate (UP) from (RP) and solve it
15:     if (optimal value of (UP)) $< f^{\mathrm{U}}$ then
16:         Update $\pi_i^{\mathrm{U}}$ for $i \in \mathcal{B}$ and $f^{\mathrm{U}}$
17:         if $f^{\mathrm{R}} = f^{\mathrm{U}}$ then    ▷ No optimality gap
18:             break
19: return $\pi_i^{\mathrm{U}}$ for $i \in \mathcal{B}$ and $f^{\mathrm{U}}$
```

4.2. Expansion of truncated relocation sequences

Each truncated relocation sequence π_i^{R} composing the current solution of (RP) is expanded in line 12 of Algorithm 1. Before going into the details of how to expand it, we first classify truncated relocation sequences.

As explained in Section 3.2, a truncated relocation sequence ends with a relocation whose destination stack is not specified. Suppose that truncated sequence π_i^{R} of block i is composed of $(n-1)$ fixed relocations and one unfixed relocation and is thus expressed as (p_1, d_0, d_1), (p_2, d_1, d_2), $\dots$, $(p_{n-1}, d_{n-2}, d_{n-1})$, $(p_n, d_{n-1}, \bullet)$, where $p_1 = r_i$ and $d_0 = s_i$. Such a truncated sequence is classified into two groups: the first group comprises truncated sequences whose remaining relocations can be zero, that is, $R'(\pi_i^{\mathrm{R}}) = n$. More specifically, π_i^{R} belongs to this group if it represents at least one complete relocation sequence whose last relocation is in period p_n: a complete sequence expressed as (p_1, d_0, d_1), $\dots$, (p_n, d_{n-1}, d_n), (i, d_n, d_n). On the other hand, π_i^{R} falls into the second group if $R'(\pi_i^{\mathrm{R}}) \geq n+1$, that is, if the relocation in period p_n is not the last one in every complete relocation sequence represented by it.

Based on this classification, truncated relocation sequence π^{R} of block i is expanded as follows. If $R'(\pi_i^{\mathrm{R}}) = n$, i.e., π_i^{R} belongs to the first group, we expand only complete sequences with n relocations. Then, we increase $R'(\pi_i^{\mathrm{R}})$ by one. If $R'(\pi_i^{\mathrm{R}}) \geq n+1$ and π_i^{R} belongs to the second group, we expand it by enumerating possible relocations in period p_n and replace π_i^{R} with new truncated sequences composed of $(n+1)$ relocations such as (p_1, d_0, d_1), $\dots$, (p_n, d_{n-1}, d_n), $(p_{n+1}, d_n, \bullet)$. A few of them are further expanded if $R'(\cdot) = n+1$ as well as some additional conditions is satisfied.

The detailed algorithm is presented in Algorithm 2, where lines 3–7 and lines 9–28 correspond to the first and second groups, respectively. For π_i^{R} in the first group, complete sequences with n relocations are expanded in lines 4–6. The set $\mathcal{D}$ in line 3 limits the destination of the nth relocation to slack stacks with priorities larger than i, so that block i can become non-blocking there. Now that no complete sequence with n relocations remains, $R'(\pi_i^{\mathrm{R}})$ is increased by one in line 7. If π_i^{R} belongs to the second group, it is first removed from $\mathcal{E}_i'$ in line 9. Then, the destination stacks d of the nth relocation and the periods p of the $(n+1)$th relocation are enumerated in lines 10 and 11, respectively. For each truncated sequence π with $(n+1)$ relocations, (p_1, d_0, d_1), $\dots$, (p_n, d_{n-1}, d), $(p, d, \bullet)$, we consider the following. If π represents only complete relocation sequences with $(n+1)$ relocations, they are all expanded in lines 14–16. Otherwise, lines 18–28 are processed. First, π is added to $\mathcal{E}_i'$ in lines 18 and 19. Next, we try to expand complete sequences with $(n+1)$ relocations. If no such sequence exists ($|\mathcal{D}'| = 0$), $R'(\pi)$ is calculated in line 21 using RelocationsLowerBound(π). This function computes a lower bound on the number of relocations, which is explained in Section 4.3. If the number of complete relocation sequences with $(n+1)$ relocations is smaller than a parameter W, they are all expanded in lines 23–25, and $R'(\pi)$ is set to $(n+2)$ in line 26. Otherwise, we simply set $R'(\pi) := n+1$ in line 28.

The initial set of $\mathcal{E}_i'$ is generated by Algorithm 3. If all sequences are composed only of one relocation, they are added to $\mathcal{E}_i'$ in lines 3–6. Otherwise, $(r_i, s_i, \bullet)$ is expanded by Algorithm 2 in lines 8–11.

For the example in Section 2, the initial relocation sequences of block 10 are generated by applying Algorithm 2 to $\pi = (2, 2, \bullet)$. As RelocationsLowerBound(π) computes the lower bound on the number of relocations $R'(\pi)$ at 2 (as shown in Section 4.3), π belongs to the second group. Therefore, all the destination stacks of the first relocation are enumerated. When block 10 is relocated to stack 1, the next relocation period can be 3 and 8. For the truncated sequence $\pi_{10,1} = (2, 2, 1)$, $(3, 1, \bullet)$ where block 10 is relocated in period 3, $\mathcal{D}'$ in line 12 becomes $\mathcal{D}' = \emptyset$ from $q_2^{(3)} = 9$, $q_3^{(3)} = 4$, and $q_4^{(3)} = 7$. Since $|\mathcal{D}'| = 0$, no further expansion is performed, and $R'(\pi_{10,1})$ is calculated in line 21 (in this case, $R'(\pi_{10,1})$ becomes 3). For the truncated sequence $\pi_{10,2} = (2, 2, 1)$, $(8, 1, \bullet)$, $\mathcal{D}' = \{3, 4\}$ because these stacks may become empty before period 8. Here, let us assume $W = 1$. Then, the condition $|\mathcal{D}'| \leq W$ in line 22 is not satisfied, and $R'(\pi_{10,2})$ is set to 2 in line 28 without further expansion. When block 10 is relocated to stack 3, the next relocation period is 4. For the truncated sequence $\pi_{10,3} = (2, 2, 3)$, $(4, 3, \bullet)$, $\mathcal{D}' = \emptyset$ holds, so that we just calculate $R'(\pi_{10,3})$ as $\pi_{10,1}$. Finally, when block 10 is relocated to stack 4, the next relocation

Algorithm 2 Expansion of truncated relocation sequence $\pi_i^{\mathrm{R}} = (p_1, d_0, d_1), \ldots, (p_n, d_{n-1}, \bullet)$.

```
1: procedure ExpandTruncatedSequence($\mathcal{E}'_i$, $\pi_i^{\mathrm{R}}$)
2:     if $R'(\pi_i^{\mathrm{R}}) = n$ then        ▷ $\pi_i^{\mathrm{R}}$ belongs to the first group
3:         $\mathcal{D} \leftarrow \{d \in \mathcal{L}^{(p_n)} \mid (d \neq d_{n-1}) \wedge (i < q_d^{(p_n)})\}$
4:         for all $d \in \mathcal{D}$ do        ▷ complete sequences with $n$ relocations
5:             $\pi \leftarrow (p_1, d_0, d_1), \ldots, (p_n, d_{n-1}, d), (i, d, d)$
6:             $\mathcal{E}'_i \leftarrow \mathcal{E}'_i \cup \{\pi\}$
7:         $R'(\pi_i^{\mathrm{R}}) \leftarrow n + 1$ ▷ $\pi_i^{\mathrm{R}}$ is now moved to the second group
8:     else        ▷ $\pi_i^{\mathrm{R}}$ belongs to the second group
9:         $\mathcal{E}'_i \leftarrow \mathcal{E}'_i \setminus \{\pi_i^{\mathrm{R}}\}$
10:        for all $d \in \mathcal{L}^{(p_n)} \setminus \{d_{n-1}\}$ do        ▷ destination stacks of $n$th relocation
11:            for all $p \in \mathcal{P}_{id}^{(p_n)}$ do        ▷ periods of $(n+1)$th relocation
12:                $\mathcal{D}' \leftarrow \{d' \in \mathcal{L}^{(p)} \mid (d' \neq d) \wedge (i < q_{d'}^{(p)})\}$
13:                if $\mathcal{P}_{id''}^{(p)} = \emptyset$ for all $d'' \in \mathcal{L}^{(p)} \setminus \{d\}$ then        ▷ all remaining sequences are with $(n+1)$ relocations
14:                    for all $d' \in \mathcal{D}'$ do        ▷ complete sequences with $(n+1)$ relocations
15:                        $\pi' \leftarrow (p_1, d_0, d_1), \ldots, (p_n, d_{n-1}, d),$
                                $(p, d, d'), (i, d', d')$
16:                        $\mathcal{E}'_i \leftarrow \mathcal{E}'_i \cup \{\pi'\}$
17:                else        ▷ sequences with more than $(n+1)$ relocations exist
18:                    $\pi \leftarrow (p_1, d_0, d_1), \ldots, (p_n, d_{n-1}, d), (p, d, \bullet)$
19:                    $\mathcal{E}'_i \leftarrow \mathcal{E}'_i \cup \{\pi\}$
20:                    if $|\mathcal{D}'| = 0$ then        ▷ no complete sequences with $(n+1)$ relocations
21:                        $R'(\pi) \leftarrow$ RelocationsLowerBound($\pi$)
22:                    else if $|\mathcal{D}'| \leq W$ then        ▷ at most $W$ complete sequences with $(n+1)$ relocations
23:                        for all $d' \in \mathcal{D}'$ do        ▷ complete sequences with $(n+1)$ relocations
24:                            $\pi' \leftarrow (p_1, d_0, d_1), \ldots, (p_n, d_{n-1}, d),$
                                    $(p, d, d'), (i, d', d')$
25:                            $\mathcal{E}'_i \leftarrow \mathcal{E}'_i \cup \{\pi'\}$
26:                        $R'(\pi) \leftarrow n + 2$
27:                    else
28:                        $R'(\pi) \leftarrow n + 1$
```

period can be 3 and 7. For the truncated sequence $\pi_{10,4} = (2, 2, 4), (3, 4, \bullet)$, $\mathcal{D}' = \emptyset$ holds, and we do nothing other than calculating $R'(\pi_{10,4})$. For the truncated sequence $\pi_{10,5} = (2, 2, 4), (7, 4, \bullet)$, $\mathcal{D}' = \{3\}$ holds. In this case, $|\mathcal{D}'| \leq W$ is satisfied. Therefore, a complete relocation sequence $\pi_{10,6} = (2, 2, 4)$, (7,4,4) is expanded in lines 23–25. Since $\pi_{10,5}$ does not represent complete relocation sequences with two relocations anymore, $R'(\pi_{10,5})$ is set to 3 in line 26. In summary, we obtain six initial relocation sequences for block 10:

- $\pi_{10,1} = (2, 2, 1), (3, 1, \bullet)$,
- $\pi_{10,2} = (2, 2, 1), (8, 1, \bullet)$,
- $\pi_{10,3} = (2, 2, 3), (4, 3, \bullet)$,
- $\pi_{10,4} = (2, 2, 4), (3, 4, \bullet)$,
- $\pi_{10,5} = (2, 2, 4), (7, 4, \bullet)$,
- $\pi_{10,6} = (2, 2, 4)$, (7,4,4).

4.3. Lower bound on the number of remaining relocations

In this section, we describe how to compute a lower bound $R'(\pi)$ on the number of relocations for a truncated relocation sequence π of block i. We assume here that π is expressed as $(p_1, d_0, d_1), \ldots, (p_{n-1}, d_{n-2}, d_{n-1}), (p_n, d_{n-1}, \bullet)$. It is not known in advance how many times block i will be relocated after the last relocation in period p_n. The function RelocationsLowerBound(π) in Algorithm 4 returns n plus the minimum number of remaining relocations as $R'(\pi)$. Considering only invariant blocks, the algorithm counts the number of remaining relocations by simply relocating block i to the stack with the maximum priority until it becomes non-blocking. In the example given in Section 2, $R'(\pi) = 2$ for a sequence $\pi = (2, 2, \bullet)$ of block 10, which is computed as follows. First, n' is initialized as $n' = 1$ in line 2. As $q_1^{(2)} = 8$, $q_3^{(2)} = 4$, and $q_4^{(2)} = 7$ (see Fig. 3c), the destination stack with the maximum priority is stack $d_1 = 1$ in line 3. Then, $q_{d_1}^{(2)} = 8 < 10 = i$ holds in line 4, implying that block 10 becomes blocking even when it is relocated to stack d_1 with the maximum priority. We must consider one more relocation so that n' is updated to $n' = 2$ in line 5. The period of the next relocation is set to $p_2 = q_{d_1}^{(1)} = 8$ in line 6. The destination stack of the relocation in period 8 is set to $d_2 = \mathrm{argmax}_{d \in \{2,3,4\}} q_d^{(8)} = 3$ (or 4) in line 7 (see Fig. 3g). Now that $q_3^{(8)} = \infty \geq 10$, the loop is terminated in line 4, and the function returns $n' = 2$ as $R'(\pi)$.

Algorithm 3 Generation of initial set $\mathcal{E}'_i$ of relocation sequences for block i

```
1: procedure InitializeSequences($\mathcal{E}'_i$)
2:     if $\mathcal{P}_{id}^{(r_i)} = \emptyset$ for all $d \in \mathcal{L}^{(r_i)}$ then        ▷ all sequences are with one relocation
3:         $\mathcal{E}'_i \leftarrow \emptyset$
4:         for all $d \in \mathcal{L}^{(r_i)}$ do        ▷ complete sequences with one relocation
5:             $\pi' \leftarrow (r_i, s_i, d), (i, d, d)$
6:             $\mathcal{E}'_i \leftarrow \mathcal{E}'_i \cup \{\pi'\}$
7:     else        ▷ sequences with more than one relocation exist
8:         $\pi \leftarrow (r_i, s_i, \bullet)$
9:         $\mathcal{E}'_i \leftarrow \{\pi\}$
10:        $R'(\pi) \leftarrow$ RelocationsLowerBound($\pi$)
11:        ExpandTruncatedSequence($\mathcal{E}'_i$, $\pi$)
```

Algorithm 4 Lower bound on the number of relocations for truncated relocation sequence π of block i

```
1: function RelocationsLowerBound($\pi$)
2:     $n' \leftarrow n$        ▷ $n$ is the number of relocations in $\pi$
3:     $d_{n'} \leftarrow \mathrm{argmax}_{d \in \mathcal{L}^{(p_{n'})} \setminus \{d_{n'-1}\}} q_d^{(p_{n'})}$
4:     while $q_{d_{n'}}^{(p_{n'})} < i$ do
5:         $n' \leftarrow n' + 1$
6:         $p_{n'} \leftarrow q_{d_{n'-1}}^{(p_{n'-1})}$
7:         $d_{n'} \leftarrow \mathrm{argmax}_{d \in \mathcal{L}^{(p_{n'})}} q_d^{(p_{n'})}$
8:     return $n'$
```

Proposition 4. *Algorithm 4 yields a valid lower bound on the number of relocations for a truncated sequence π of block i expressed as $(p_1, d_0, d_1), \ldots, (p_{n-1}, d_{n-2}, d_{n-1}), (p_n, d_{n-1}, \bullet)$.*

Proof. See Appendix A. □

4.4. Checking a conflict between two relocation sequences

To define conflict constraints (17), it is necessary to check a conflict between every pair of relocation sequences π_i and π_j of blocks i and j ($< i$), respectively. Specifically, we have to check Conditions (a)–(c) for each pair of π_i and π_j. Given that π_i and π_j are expressed as $(p_1, d_0, d_1), \dots, (p_n, d_{n-1}, d_n)$ and $(p'_1, d'_0, d'_1), \dots, (p'_{n'}, d'_{n'-1}, d'_{n'})$, respectively, Algorithm 5 checks the conflict. Here, $d_n = \bullet$ (respectively, $d_{n'} = \bullet$) if π_i (respectively, π_j) is truncated, and $(p_n = i) \wedge (d_{n-1} = d_n)$ (respectively, $(p_{n'} = j) \wedge (d_{n'-1} = d_{n'})$) if π_i (respectively, π_j) is complete.

In this algorithm, X denotes the relation between blocks i and j. Specifically,

$$X = \begin{cases} 1 & \text{block } i \text{ is above block } j \text{ in the same stack,} \\ -1 & \text{block } j \text{ is above block } i \text{ in the same stack,} \\ 0 & \text{blocks } i \text{ and } j \text{ are placed in different stacks.} \end{cases} \tag{20}$$

It is initialized in lines 4–10. The relocation sequences of the blocks are checked from the first to the last in lines 11–39. According to Condition (c), block j (respectively, i) should also be relocated if block i (respectively, j) below it is relocated. A violation of Condition (c) is detected in lines 13–14: block j is above block i ($X = -1$), whereas block j is not relocated together with block i in period p_k. Similarly, lines 21–22 detect a violation of Condition (c) such that block i above block j is not relocated in period $p'_{k'}$. Precisely, Condition (a) is violated if block j is retrieved in period $p'_{k'}$. When blocks i and j are relocated or retrieved in the same period, Conditions (a) and (b) are checked in lines 29–32. In lines 29–30, Condition (a) is violated because block i is relocated in period $p'_{k'} = j$ when block j is retrieved, although block i is not above block j ($X \neq 1$). In lines 31–32, blocks i and j are in different stacks ($X = 0$), which violates Condition (b). After all these checks have been finished, X is updated in lines 15–18, 23–26, and 33–38.

If π_j is truncated and block i is relocated in period j, we check Condition (a) twice more in lines 40–47. The first check is in lines 41–42. If block j is placed above block i at the beginning of period $p'_{n'}$, and blocks i and j are both relocated in this period, Y becomes true in line 37. If, on the other hand, Y = false, block i is not relocated in period $p'_{n'}$, or, is relocated before block j in period $p'_{n'}$. In this case, block i is never placed above block j at the end of period $p'_{n'}$. Therefore, block i should be relocated at least once after period $p'_{n'}$ and before period j, to relocate block i in period j by placing it above block j. The inequality $p_{l-1} \leq p'_{n'}$ in line 41 implies that block i is not relocated between periods $p'_{n'}$ and j $(= p_l)$. The second check is in lines 43–47. Granted that Y = true, block i may not be above block j if block j is relocated after $p'_{n'}$ until its retrieval. In this case, block i should be relocated at least once after the last relocation of block j and before period j. To obtain the period of the last relocation of block j, let us consider the number of its remaining relocations: it is at least $R'(\pi_j) - n'$, but even if $R'(\pi_j) = n'$, we should consider one more relocation if block j is retrieved from stack $d'_{n'-1}$ in period $j = p_l$, that is, if $d'_{n'-1} = d_{l-1}$. This is because the relocation of block j in period $p'_{n'}$ is from stack $d'_{n'-1}$, and we need at least one more relocation to move block j back to stack $d'_{n'-1}$. If this is the case, a lower bound on the number of relocations of block j, which is copied to n'' in line 43, is increased to $n'+1$ in lines 44–45. Using this n'', lines 46–47 detect that block i is not relocated after the n''th relocation of block j and before period j. Here, the earliest period of the n''th relocation of block j is computed by EarliestLastRelocationPeriod(π_j, n''), as explained in the following subsection.

4.5. Earliest period of the last relocation in a truncated relocation sequence

The function EarliestLastRelocationPeriod(π, n'') computes the earliest period of the n''th relocation in the relocation sequence $\pi = (p_1, d_0, d_1), \dots, (p_n, d_{n-1}, \bullet)$ of block i. The algorithm is provided in Algorithm 6. In short, it computes the earliest period of the next relocation iteratively until all remaining relocations are considered. The earliest period of the $(k+1)$th relocation is determined from the priorities of the possible destina-

Algorithm 5 Conflict check between two relocation sequences π_i and π_j of blocks i and j $(< i)$, respectively

```
1: function CheckConflict($\pi_i$, $\pi_j$)
2:   $k \leftarrow 1$, $k' \leftarrow 1$
3:   $Y \leftarrow$ true
4:   if $s_i = s_j$ then                ▷ blocks $i$ and $j$ are in the same stack
5:     if $t_i > t_j$ then
6:       $X \leftarrow 1$                ▷ block $i$ is above block $j$
7:     else
8:       $X \leftarrow -1$               ▷ block $j$ is above block $i$
9:   else
10:    $X \leftarrow 0$                  ▷ blocks $i$ and $j$ are in different stacks
11:  while $(k \leq n) \wedge (k' \leq n')$ do
12:    if $p_k < p'_{k'}$ then           ▷ block $i$ is relocated
13:      if $X = -1$ then
14:        return true    ▷ Condition (c): block $j$ above block $i$ is not relocated
15:      if $d_k = d'_{k'-1}$ then
16:        $X \leftarrow 1$             ▷ block $i$ is placed above block $j$
17:      else
18:        $X \leftarrow 0$
19:      $k \leftarrow k + 1$
20:    else if $p_k > p'_{k'}$ then      ▷ block $j$ is relocated or retrieved
21:      if $X = 1$ then
22:        return true    ▷ Condition (c): block $i$ above block $j$ is not relocated
23:      if $d'_{k'} = d_{k-1}$ then
24:        $X \leftarrow -1$            ▷ block $j$ is placed above block $i$
25:      else
26:        $X \leftarrow 0$
27:      $k' \leftarrow k' + 1$
28:    else      ▷ blocks $i$ and $j$ are relocated or retrieved in the same period
29:      if $(p'_{k'} = j) \wedge (X \neq 1)$ then
30:        return true    ▷ Condition (a): when block $j$ is retrieved, block $i$ is not above it, but is relocated
31:      else if $X = 0$ then
32:        return true    ▷ Condition (b): blocks in different stacks are relocated or retrieved
33:      if $d_k = d'_{k'}$ then
34:        $X \leftarrow -X$            ▷ blocks $i$ and $j$ are relocated to the same stack
35:      else
36:        if $(d'_{k'} = \bullet) \wedge (X = -1)$ then
37:          $Y \leftarrow$ true   ▷ block $i$ can be above block $j$ at the end of period $p'_{k'} = p'_{n'}$
38:        $X \leftarrow 0$
39:      $k \leftarrow k + 1$, $k' \leftarrow k' + 1$
40:  if $p_l = j$ for some $l$ $(k \leq l \leq n)$ then   ▷ $\pi_j$ is truncated and block $i$ is relocated in period $j$
41:    if $(Y = \text{false}) \wedge (p_{l-1} \leq p'_{n'})$ then
42:      return true▷ Condition (a): when block $j$ is retrieved, block $i$ not above it is relocated
43:    $n'' \leftarrow R'(\pi_j)$ ▷ minimum number of relocations for block $j$
44:    if $d'_{n'-1} = d_{l-1}$ then
45:      $n'' \leftarrow \max(n'+1, n'')$   ▷ block $j$ should be relocated at least once
46:    if $p_{l-1} <$ EarliestLastRelocationPeriod($\pi_j$, $n''$) then
47:      return true          ▷ Condition (a): when block $j$ is retrieved, block $i$ not above it is relocated
48:  return false
```

Algorithm 6 Computation of the earliest period of the n''th relocation for a truncated sequence π of block i

```
1: function EarliestLastRelocationPeriod($\pi$, $n''$)
2:     $k \leftarrow n$                                  ▷ $n$ is the number of relocations in $\pi$
3:     while $k < n''$ do
4:         $d \leftarrow \mathrm{argmin}_{s\in\mathcal{L}^{(p_k)}\setminus\{d_{k-1}\}} q_s^{(p_k)}$
5:         if $q_d^{(p_k)} < \min \mathcal{Q}_i^{(p_k)}$ then
6:             $(p_{k+1}, d_k) \leftarrow (q_d^{(p_k)}, d)$
7:         else
8:             $(p_{k+1}, d_k) \leftarrow (\min \mathcal{Q}_i^{(p_k)}, 0)$
9:         $k \leftarrow k+1$
10:    return $p_k$
```

tion stacks ($q_s^{(p_k)}$ with $s \in \mathcal{L}^{(p_k)} \setminus \{d_{k-1}\}$) in line 6, or from blocking blocks already relocated, but not retrieved ($\mathcal{Q}_i^{(p_k)}$) in line 8. If the latter yields an earlier period than the former, the destination stack of the kth relocation is assumed to be arbitrary ($d_k = 0$). Please note that $d_{k-1} \notin \mathcal{L}^{(p_k)}$ and $\mathcal{L}^{(p_k)} \setminus \{d_{k-1}\} = \mathcal{L}^{(p_k)}$ hold for $k \geq n+1$. Obviously, $d_{k-1} \notin \mathcal{L}^{(p_k)}$ holds if $d_{k-1} = 0$. If $d_{k-1} = \mathrm{argmin}_{s\in\mathcal{L}^{(p_{k-1})}\setminus\{d_{k-2}\}} q_s^{(p_{k-1})}$, stack d_{k-1} is the target stack in period $p_k = q_{d_{k-1}}^{(p_{k-1})}$, and hence $d_{k-1} \notin \mathcal{L}^{(p_k)}$ holds. The following proposition guarantees the validity of this algorithm.

Proposition 5. *Algorithm 6 yields a valid lower bound on the period of the n''th relocation of a truncated sequence π of block i expressed as $(p_1, d_0, d_1), \ldots, (p_{n-1}, d_{n-2}, d_{n-1}), (p_n, d_{n-1}, \bullet)$.*

Proof. See Appendix B. □

5. Computational experiments

In this section, we verify the effectiveness of the proposed exact algorithm through computational experiments.

5.1. Benchmark instances

Two benchmark datasets are taken from the literature. The first, which is referred to as the CVS dataset, is derived from Caserta, Voß, and Sniedovich (2011b) and Caserta et al. (2012).[1] It provides initial bay configurations characterized by two parameters S and H, where $S \in \{3, 4, 5, 6, 7, 8, 9, 10\}$ is the number of stacks in the bay and $H \in \{3, 4, 5, 6, 10\}$ is the number of blocks in each stack. Thus, a total of $N = SH$ blocks are initially placed in the bay. The bay configuration in each instance is generated randomly. The dataset includes 40 instances for each combination of H and S, and is composed of 840 instances (21 subsets). Following the methods described in the literature, we consider two settings of the stack height limit T for these instances: $T = H+2$ and $T = \infty$ (unlimited stack height).

The second dataset, referred to as the ZQLZ dataset, is derived from Zhu et al. (2012).[2] The instances in this dataset are characterized by three parameters S, T, and N, where $6 \leq S \leq 10$, $3 \leq T \leq 7$, and $(S-1)T \leq N \leq ST-1$. The dataset includes 100 instances for each combination of the parameters and is composed of 12,500 instances (125 subsets).

5.2. Algorithms for comparison

The proposed algorithm was coded in C++. For comparison, the branch-and-cut algorithm BC-RBRP by Bacci et al. (2020) was also applied using the program coded in C++, which has been provided by the authors.[3] In both algorithms, Gurobi Optimizer 9.0.1 was employed to solve IP problems.

BC-RBRP starts from an incomplete IP formulation with part of the constraints and adds the rest of (exponentially many) constraints on-the-fly with the call-back function of the IP solver. An initial upper bound is computed using the beam search algorithm proposed by the same authors (Bacci et al., 2019). Note that our proposed algorithm utilizes only a simple greedy heuristic for an initial upper bound. We did not apply more sophisticated (meta)heuristics for finding initial upper bounds to keep the overall algorithm as simple as possible.

In addition to these IP-based approaches, we apply the state-of-the-art branch-and-bound algorithm from Tanaka and Mizuno (2018), which is an improved version of that in Tanaka and Takii (2016) by dominance rules. Precisely, we use a further improved version in this study. The first improvement utilizes two new lower bounds LB-LIS (Quispe et al., 2018) and UBALB (Bacci et al., 2019) in place of LB4, to compare the effectiveness of the lower bounds. The other improvement is the termination of the lower bound computation. These three lower bounds are computed by iterative algorithms, and the lower bound value increases as the corresponding algorithm proceeds. For pruning the current node in the branch-and-bound algorithm, however, we only need to know whether the lower bound is larger than the current upper bound. Therefore, the lower bound computation is terminated as soon as the lower bound becomes larger than or equal to the current upper bound. This technique for speeding up is applied to all three types of lower bound. Finally, the algorithm is multi-threaded for a fair comparison with the multi-threaded IP solver. This branch-and-bound algorithm is coded in C.

The source codes of the proposed algorithm and the branch-and-bound algorithm are available online[4]. All computations were conducted using desktop computers equipped with an Intel Core i9-9900K CPU (8 cores) at 3.6 gigahertz and 64 gigabytes of RAM. The time limit was set to 1 hour for each instance, and the number of threads was chosen as 1 (single-threading) or 8 (multi-threading). Note that the CPU clock rate accelerates to a maximum of 5.0 gigahertz in single-threading and 4.7 gigahertz in multi-threading.

5.3. Best parameter W

Before examining the detailed results, let us fix the parameter W in Algorithm 2 for expanding truncated relocation sequences. Obviously, a larger number of relocation sequences is generated in one iteration as W increases, making (RP) and (UP) harder to solve. On the other hand, (RP) and (UP) with a larger number of relocation sequences may yield better lower and upper bounds, which can decrease the total number of iterations in Algorithm 1. Therefore, a trade-off relation exists between the increase in computation time for one iteration and the decrease in the number of iterations. To find the best W, we ran the multi-threaded algorithm with W being changed between 0 and 4, and without setting a time limit. The maximum and total computation times for each dataset are summarized in Table 1. In this table, “max” and “total” are the maximum and total computation times in seconds, respectively, for the corresponding

[1] https://www.bwl.uni-hamburg.de/en/iwi/forschung/projekte/dataprojekte/brp-instances-caserta-etal-2012.zip.
[2] Mirrored in https://sites.google.com/site/shunjitanaka/brp.
[3] http://www.iasi.cnr.it/~tbacci/.
[4] https://sites.google.com/site/shunjitanaka/brp.

Table 1
Relation between parameter W and computation time for each dataset (8 threads).

W	CVS dataset				ZQLZ dataset	
	$T = H + 2$		$T = \infty$			
	max	total	max	total	max	total
0	371,691	1,151,653	2547	7029	2981	24,764
1	420,982	1,040,771	2543	6134	3013	**22,590**
2	**93,139**	**583,181**	**826**	**4385**	**2926**	25,765
3	189,083	957,249	1531	5685	6365	41,985
4	1,697,689	2,962,602	2413	6667	14,579	63,128

Table 2
Computational results for the CVS dataset with $T = H + 2$ (single thread).

H	S	N	n	b&b									IP-based					
				LB4			LB-LIS			UBALB			BC-RBRP			Proposed		
				#opt	ave	max	#opt	ave	max	#opt	ave	max	#opt	ave	max	#opt	ave	max
3	3	9	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00
	4	12	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00
	5	15	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.01	**40**	0.00	0.00
	6	18	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.01	0.02	**40**	0.00	0.00
	7	21	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.01	0.02	**40**	0.00	0.00
	8	24	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.02	0.03	**40**	0.00	0.00
4	4	16	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.01	**40**	0.00	0.00
	5	20	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.01	0.02	**40**	0.00	0.01
	6	24	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.02	0.05	**40**	0.00	0.01
	7	28	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.03	0.08	**40**	0.00	0.03
5	4	20	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.01	0.08	**40**	0.01	0.13
	5	25	40	**40**	0.00	0.01	**40**	0.00	0.01	**40**	0.00	0.02	**40**	0.05	0.48	**40**	0.02	0.45
	6	30	40	**40**	0.01	0.15	**40**	0.00	0.12	**40**	0.03	0.69	**40**	0.12	2.43	**40**	0.02	0.10
	7	35	40	**40**	0.05	1.00	**40**	0.04	0.78	**40**	0.38	11.40	**40**	0.11	0.47	**40**	0.02	0.09
	8	40	40	**40**	0.72	13.27	**40**	0.52	9.03	**40**	3.90	70.26	**40**	0.18	0.70	**40**	0.04	0.34
	9	45	40	**40**	1.02	11.25	**40**	0.75	8.98	**40**	6.20	73.01	**40**	0.52	6.54	**40**	0.04	0.32
	10	50	40	39	173.83	3600.00	39	149.31	3600.00	38	242.21	3600.00	**40**	0.59	3.03	**40**	0.10	0.93
6	6	36	40	**40**	5.15	92.49	**40**	3.92	68.60	39	110.21	3600.00	**40**	1.14	11.55	**40**	0.27	2.12
	10	60	40	28	1315.28	3600.00	30	1227.08	3600.00	22	2028.64	3600.00	**40**	22.82	277.46	**40**	0.87	7.05
10	6	60	40	7	3128.77	3600.00	8	3172.63	3600.00	3	3416.77	3600.00	19	2267.74	3600.00	31	1214.43	3600.00
	10	100	40	0	3600.00	3600.00	0	3600.00	3600.00	0	3600.00	3600.00	3	3452.60	3600.00	28	1519.13	3600.00
	Total		840	754			757			742			782			819		

dataset. We can see that $W = 2$ provides the best results except for the total computation time for the ZQLZ dataset. Based on these results, parameter W was fixed to 2 in the following experiments.

5.4. Computational results for the CVS dataset

Now, we examine the results for the CVS dataset in more detail. The computational results for the CVS dataset are summarized in Tables 2–5. In these tables, "n" is the number of instances in each subset and "#opt" is the number of instances solved to optimality among them. The average and maximum computation times in seconds over all instances in each subset are provided in columns "ave" and "max," respectively. Note that the computation time of BC-RBRP does not include the time to obtain an initial upper bound. It applies the beam search algorithm of Bacci et al. (2019) with a time limit of 1 second.

5.4.1. Comparison with other exact approaches

Let us examine the results by the branch-and-bound algorithm "b&b." Comparing Tables 2 and 3 or Tables 4 and 5, we can observe that multi-threading is effective: the algorithm is four or five times faster in multi-threading than in single-threading, even though it did not reach the ideal improvement of a factor of eight. Using LB-LIS in place of LB4 results in a slight increase in speed, except for instances with $(H, S) = (10, 6)$. In contrast, UBALB rather makes the algorithm slower, so that fewer instances are solved to optimality than with LB4 and LB-LIS. From Tables 2 and 4, the instances with $T = \infty$ appear to be harder to solve than those with $T = H + 2$ for the branch-and-bound algorithm. As LB4 and LB-LIS do not consider the stack height limit, they become tighter for the instances with $T = \infty$ as the optimal values become smaller by relaxing the stack height limit. Therefore, we may well expect that the performance of the algorithm improves when $T = \infty$. In reality, the negative effect of the increase in the search tree size dominated the improvement, making the algorithm slower. The branch-and-bound algorithm enumerates feasible relocations of blocks from first to last to search for an optimal sequence of relocations. When $T = \infty$, $(S - 1)$ stacks should always be considered as possible destination stacks of a relocation, whereas some of them may be full and, thus, excluded when $T = H + 2$. This is more likely to occur in the first few relocations when many blocks remain in the bay, enabling many branches and nodes in the search tree to be cut off.

Next, let us examine the IP-based approaches, BC-RBRP and the newly proposed algorithm. BC-RBRP is more scalable than the branch-and-bound algorithm and can solve a larger number of instances to optimality in the time limit. For example, the branch-and-bound algorithm failed in solving one or two instances to optimality with $(H, S) = (5, 10)$ and $T = H + 2$ in 1 hour, whereas BC-RBRP took at most 3 seconds for each. The difference is more evident for the instances with $(H, S) = (6, 10)$ and $T = H + 2$: BC-RBRP solved all 40 instances to optimality, but the branch-and-bound algorithm solved at most 80%. The proposed algorithm is

Table 3
Computational results for the CVS dataset with $T = H + 2$ (8 threads).

H	*S*	*N*	*n*	b&b									IP-based					
				LB4			LB-LIS			UBALB			BC-RBRP			Proposed		
				#opt	ave	max	#opt	ave	max	#opt	ave	max	#opt	ave	max	#opt	ave	max
3	3	9	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00
	4	12	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00
	5	15	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.01	**40**	0.00	0.00
	6	18	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.01	0.02	**40**	0.00	0.00
	7	21	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.01	0.02	**40**	0.00	0.00
	8	24	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.02	0.03	**40**	0.00	0.01
4	4	16	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.01	**40**	0.00	0.00
	5	20	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.01	0.02	**40**	0.00	0.01
	6	24	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.02	0.05	**40**	0.00	0.02
	7	28	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.03	0.08	**40**	0.00	0.03
5	4	20	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.01	0.08	**40**	0.01	0.23
	5	25	40	**40**	0.00	0.01	**40**	0.00	0.01	**40**	0.00	0.01	**40**	0.04	0.33	**40**	0.02	0.22
	6	30	40	**40**	0.00	0.03	**40**	0.00	0.02	**40**	0.01	0.13	**40**	0.08	0.74	**40**	0.02	0.11
	7	35	40	**40**	0.01	0.15	**40**	0.01	0.13	**40**	0.07	2.24	**40**	0.11	0.47	**40**	0.02	0.10
	8	40	40	**40**	0.19	3.48	**40**	0.12	2.03	**40**	0.84	15.36	**40**	0.18	0.71	**40**	0.04	0.35
	9	45	40	**40**	0.20	1.96	**40**	0.16	1.68	**40**	1.37	16.39	**40**	0.48	4.97	**40**	0.04	0.32
	10	50	40	**40**	99.44	3398.24	**40**	83.00	2859.04	39	184.97	3600.00	**40**	0.59	2.97	**40**	0.10	0.96
6	6	36	40	**40**	1.04	17.65	**40**	0.89	15.05	**40**	71.56	2665.66	**40**	0.88	8.67	**40**	0.24	1.45
	10	60	40	31	926.34	3600.00	31	906.56	3600.00	27	1419.62	3600.00	**40**	15.92	233.66	**40**	0.80	6.32
10	6	60	40	13	2673.79	3600.00	12	2748.22	3600.00	4	3335.42	3600.00	20	2049.60	3600.00	35	937.06	3600.00
	10	100	40	0	3600.00	3600.00	0	3600.00	3600.00	0	3600.00	3600.00	2	3488.05	3600.00	29	1412.74	3600.00
	Total		840	764			763			750			782			824		

Table 4
Computational results for the CVS dataset with $T = \infty$ (single thread).

H	*S*	*N*	*n*	b&b									IP-based					
				LB4			LB-LIS			UBALB			BC-RBRP			Proposed		
				#opt	ave	max	#opt	ave	max	#opt	ave	max	#opt	ave	max	#opt	ave	max
3	3	9	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00
	4	12	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00
	5	15	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.01	**40**	0.00	0.00
	6	18	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.01	0.02	**40**	0.00	0.00
	7	21	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.01	0.02	**40**	0.00	0.00
	8	24	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.02	0.03	**40**	0.00	0.00
4	4	16	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.01	**40**	0.00	0.01
	5	20	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.01	0.02	**40**	0.00	0.01
	6	24	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.01	**40**	0.02	0.06	**40**	0.00	0.01
	7	28	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.01	**40**	0.03	0.06	**40**	0.00	0.01
5	4	20	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.01	0.03	**40**	0.00	0.05
	5	25	40	**40**	0.00	0.01	**40**	0.00	0.01	**40**	0.00	0.01	**40**	0.03	0.16	**40**	0.01	0.12
	6	30	40	**40**	0.01	0.27	**40**	0.01	0.21	**40**	0.07	1.06	**40**	0.06	0.30	**40**	0.01	0.04
	7	35	40	**40**	0.14	2.87	**40**	0.12	2.28	**40**	0.95	26.39	**40**	0.08	0.30	**40**	0.01	0.09
	8	40	40	**40**	1.90	27.66	**40**	1.39	20.11	**40**	10.84	176.86	**40**	0.12	0.52	**40**	0.02	0.11
	9	45	40	**40**	2.08	28.83	**40**	1.58	23.02	**40**	16.08	276.53	**40**	0.28	1.52	**40**	0.02	0.20
	10	50	40	38	200.74	3600.00	38	195.45	3600.00	38	265.88	3600.00	**40**	0.40	3.74	**40**	0.03	0.34
6	6	36	40	**40**	82.69	3043.53	**40**	70.72	2611.43	39	178.08	3600.00	**40**	0.65	10.53	**40**	0.12	0.87
	10	60	40	22	1914.78	3600.00	23	1820.38	3600.00	14	2459.81	3600.00	**40**	4.00	49.14	**40**	0.34	3.61
10	6	60	40	1	3511.31	3600.00	1	3511.02	3600.00	0	3600.00	3600.00	33	973.28	3600.00	**40**	16.57	141.31
	10	100	40	0	3600.00	3600.00	0	3600.00	3600.00	0	3600.00	3600.00	16	2428.80	3600.00	**40**	123.91	1612.98
	Total		840	741			742			731			809			**840**		

more efficient than BC-RBRP and, thus, the branch-and-bound algorithm. It solved two-thirds of the hardest instances to optimality with $(H, S) = (10, 10)$ and $T = H + 2$. In contrast, BC-RBRP could solve only a few instances in this subset. Notably, our newly proposed algorithm succeeded in solving all 840 instances with $T = \infty$ in both single-threading and multi-threading.

Unlike the branch-and-bound algorithm, the IP-based approaches are faster for the instances with $T = \infty$ than for those with $T = H + 2$. Setting $T = \infty$ reduces the number of constraints in the IP problems, which makes them easier to solve. The multi-threading performance was less significant for the IP-based approaches than for the branch-and-bound algorithm. This may be due to the heavy memory usage of the IP solver when solving hard IP problems. Indeed, the multi-threading performance of BC-RBRP seems better for easier instances. Nevertheless, the proposed algorithm solves a few more instances to optimality in multi-threading than in single-threading. As is investigated in the following, the proposed IP formulations (RP) and (UP) are considerably smaller than the formulation in BC-RBRP. This would be the reason why the proposed algorithm exhibited a better multi-threading performance.

Table 5
Computational results for the CVS dataset with $T = \infty$ (8 threads).

H	S	N	n	b&b									IP-based					
				LB4			LB-LIS			UBALB			BC-RBRP			Proposed		
				#opt	ave	max	#opt	ave	max	#opt	ave	max	#opt	ave	max	#opt	ave	max
3	3	9	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00
3	4	12	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00
	5	15	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.01	**40**	0.00	0.00
	6	18	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.01	0.02	**40**	0.00	0.00
	7	21	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.01	0.02	**40**	0.00	0.00
	8	24	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.02	0.03	**40**	0.00	0.00
4	4	16	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.01	**40**	0.00	0.01
	5	20	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.01	0.02	**40**	0.00	0.01
	6	24	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.02	0.06	**40**	0.00	0.01
	7	28	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.03	0.06	**40**	0.00	0.01
5	4	20	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.01	0.03	**40**	0.00	0.05
	5	25	40	**40**	0.00	0.00	**40**	0.00	0.00	**40**	0.00	0.01	**40**	0.03	0.16	**40**	0.01	0.13
	6	30	40	**40**	0.00	0.05	**40**	0.00	0.04	**40**	0.01	0.21	**40**	0.06	0.29	**40**	0.01	0.04
	7	35	40	**40**	0.02	0.32	**40**	0.02	0.28	**40**	0.24	6.89	**40**	0.08	0.30	**40**	0.01	0.09
	8	40	40	**40**	0.33	4.70	**40**	0.28	4.09	**40**	2.64	45.04	**40**	0.12	0.53	**40**	0.02	0.11
	9	45	40	**40**	0.41	5.00	**40**	0.35	4.35	**40**	3.90	84.77	**40**	0.28	1.43	**40**	0.02	0.20
	10	50	40	39	134.29	3600.00	39	126.99	3600.00	38	195.13	3600.00	**40**	0.39	3.04	**40**	0.03	0.35
6	6	36	40	**40**	15.10	554.55	**40**	13.99	515.78	39	114.22	3600.00	**40**	0.43	6.36	**40**	0.13	0.89
	10	60	40	26	1438.07	3600.00	26	1407.57	3600.00	20	2048.13	3600.00	**40**	4.62	67.06	**40**	0.37	4.83
10	6	60	40	2	3508.09	3600.00	2	3502.79	3600.00	0	3600.00	3600.00	35	736.35	3600.00	**40**	15.19	65.35
	10	100	40	0	3600.00	3600.00	0	3600.00	3600.00	0	3600.00	3600.00	19	2458.40	3600.00	**40**	93.84	825.69
	Total		840	747			747			737			814			**840**		

Table 6
Detailed results for the CVS dataset with $T = H + 2$ (8 threads).

H	S	N	#opt	#iter	Initial gap		Initial size		Final size		BC-RBRP		Computation time [seconds]			
					ave	max	#var	#cnstr	#var	#cnstr	#var	#cnstr	(RP)	(UP)	other	total
3	3	9	40	1.0	0.000	0	10.7	13.4	10.8	13.6	243	561.0	0.00	0.00	0.00	0.00
	4	12	40	1.0	0.000	0	18.3	27.2	18.9	28.4	576	1297.4	0.00	0.00	0.00	0.00
	5	15	40	1.0	0.000	0	26.0	47.5	26.6	48.9	1125	2498.6	0.00	0.00	0.00	0.00
	6	18	40	1.0	0.000	0	35.5	73.7	36.6	76.3	1944	4276.8	0.00	0.00	0.00	0.00
	7	21	40	1.0	0.000	0	43.6	96.0	44.5	98.0	3087	6724.5	0.00	0.00	0.00	0.00
	8	24	40	1.0	0.000	0	58.3	145.6	59.5	147.8	4608	10020.0	0.00	0.00	0.00	0.00
4	4	16	40	1.1	0.025	1	30.0	50.9	37.3	72.2	1024	2329.7	0.00	0.00	0.00	0.00
	5	20	40	1.2	0.025	1	48.0	97.3	60.0	133.5	2000	4474.6	0.00	0.00	0.00	0.00
	6	24	40	1.2	0.025	1	59.9	135.1	67.1	155.7	3456	7655.7	0.00	0.00	0.00	0.00
	7	28	40	1.2	0.000	0	74.7	194.6	92.2	256.1	5488	12087.7	0.00	0.00	0.00	0.00
5	4	20	40	1.8	0.125	2	46.9	100.4	83.0	253.4	1600	3648.8	0.01	0.00	0.00	0.01
	5	25	40	1.9	0.175	2	74.6	194.2	131.4	439.5	3125	7039.7	0.01	0.00	0.00	0.02
	6	30	40	1.8	0.100	1	99.6	290.2	165.2	607.4	5400	12064.1	0.01	0.00	0.00	0.02
	7	35	40	1.7	0.000	0	116.7	368.0	195.4	773.4	8575	18986.0	0.01	0.00	0.00	0.02
	8	40	40	1.9	0.025	1	152.3	518.4	239.3	1017.3	12,800	28216.9	0.03	0.01	0.00	0.04
	9	45	40	2.0	0.000	0	178.1	642.2	295.0	1360.7	18,225	40016.2	0.03	0.01	0.00	0.04
	10	50	40	2.2	0.000	0	216.3	855.0	344.6	1762.2	25,000	54712.5	0.06	0.03	0.01	0.10
6	6	36	40	3.4	0.350	2	154.6	558.4	384.3	2224.9	7776	17403.0	0.18	0.05	0.01	0.24
	10	60	40	4.0	0.100	1	309.0	1552.5	797.5	6173.9	36,000	79082.3	0.56	0.22	0.02	0.80
10	6	60	35	15.5	3.743	10	364.8	2365.0	2615.1	37237.8	21,600	48341.0	473.02	83.25	0.37	556.64
	10	100	29	12.4	1.241	4	891.7	9699.4	5268.3	110214.4	100,000	220091.1	489.73	92.23	1.13	583.10

5.4.2. *Detailed results of the proposed algorithm*

More detailed results of the proposed algorithm in multi-threading are provided in Tables 6 and 7. In these tables, “#opt” is the number of instances solved to optimality in each subset, “#iter” is the average number of iterations executed, and “#var” and “#constr” are the average numbers of decision variables and constraints in the IP problem, respectively. Furthermore, “initial” and “final” indicate the results at the first and last iterations, respectively, and “gap” implies the average (“ave”) and maximum (“max”) *absolute* gaps between the lower bound and the optimal value. For comparison, the number of decision variables and the average number of constraints in the initial (possibly incomplete) formulation in BC-RBRP are also provided. Note that the number of decision variables is given by N^2S. The average computation time in each component of the proposed algorithm is summarized in the column “computation time”, where “(RP)” is the time spent for solving (RP), “(UP)” for solving (UP), “others” for the other part of the algorithm, and “total” is the total computation time. The observations from these tables are summarized as follows.

Initial lower bound The initial lower bound is very tight. In particular, it is equal to the optimal value for the instances with $H = 3$. For the instances with $H = 4$, 5 and 6, the initial absolute gap from the optimal value is at most 2. It becomes large for the instances with $(H, S) = (10, 6)$ because the H/S ratio is high. For such instances, the number of relocations of each block in an optimal solution tends to be large. As the initial relocation sequences gener-

Table 7
Detailed results for the CVS dataset with $T = \infty$ (8 threads).

H	S	N	#opt	#iter	Initial gap		Initial size		Final size		BC-RBRP		Computation time [seconds]			
					ave	max	#var	#cnstr	#var	#cnstr	#var	#cnstr	(RP)	(UP)	other	total
3	3	9	40	1.0	0.000	0	10.7	8.7	10.9	8.9	243	534.0	0.00	0.00	0.00	0.00
	4	12	40	1.0	0.000	0	18.3	13.8	18.8	14.6	576	1249.4	0.00	0.00	0.00	0.00
	5	15	40	1.0	0.000	0	26.0	20.4	26.1	20.8	1125	2423.6	0.00	0.00	0.00	0.00
	6	18	40	1.0	0.000	0	35.5	28.0	36.8	30.9	1944	4168.8	0.00	0.00	0.00	0.00
	7	21	40	1.0	0.000	0	43.6	30.7	44.4	32.0	3087	6577.5	0.00	0.00	0.00	0.00
	8	24	40	1.0	0.000	0	58.3	47.0	59.5	48.6	4608	9828.0	0.00	0.00	0.00	0.00
4	4	16	40	1.2	0.000	0	30.0	29.1	38.5	51.3	1024	2265.7	0.00	0.00	0.00	0.00
	5	20	40	1.2	0.025	1	48.0	54.5	56.5	77.7	2000	4374.6	0.00	0.00	0.00	0.00
	6	24	40	1.1	0.000	0	59.9	67.7	64.8	78.1	3456	7511.7	0.00	0.00	0.00	0.00
	7	28	40	1.2	0.000	0	74.7	93.5	90.0	142.3	5488	11891.7	0.00	0.00	0.00	0.00
5	4	20	40	1.5	0.050	1	46.9	69.0	80.0	198.8	1600	3568.8	0.00	0.00	0.00	0.00
	5	25	40	1.7	0.100	2	74.6	137.5	125.1	358.7	3125	6914.7	0.00	0.00	0.00	0.01
	6	30	40	1.6	0.050	1	99.6	195.5	152.0	440.2	5400	11884.1	0.00	0.00	0.00	0.01
	7	35	40	1.5	0.000	0	116.7	234.8	181.3	575.1	8575	18741.0	0.01	0.00	0.00	0.01
	8	40	40	1.8	0.025	1	152.3	330.0	225.1	751.8	12,800	27896.9	0.01	0.00	0.00	0.02
	9	45	40	2.0	0.000	0	178.1	394.7	281.5	1014.3	18,225	39611.2	0.02	0.01	0.00	0.02
	10	50	40	1.6	0.000	0	216.3	534.2	304.8	1161.2	25,000	54212.5	0.02	0.01	0.00	0.03
6	6	36	40	3.2	0.275	2	154.6	444.8	350.3	1850.5	7776	17187.0	0.09	0.03	0.00	0.13
	10	60	40	3.4	0.025	1	309.0	1170.7	742.9	5215.3	36,000	78482.3	0.22	0.13	0.02	0.37
10	6	60	40	8.4	1.325	5	353.7	2102.6	1817.9	23922.5	21,600	47996.0	11.94	3.11	0.13	15.19
	10	100	40	8.6	0.500	2	809.1	7814.4	4182.3	85504.5	100,000	219163.5	75.51	17.82	0.52	93.84

Table 8
Computation time for hard instances in the CVS dataset with $(H, S) = (10, 6)$, (10,10) and $T = H + 2$ (8 threads).

H	S	N	n	Computation time									
				$\le$ 1 hour	$\le$ 3 hours	$\le$ 6 hours	$\le$ 9 hours	$\le$ 12 hours	$\le$ 15 hours	$\le$ 18 hours	$\le$ 21 hours	$\le$ 24 hours	$\le$ 27 hours
10	6	60	40	35	37	38	**40**						
10	10	100	40	29	33	35	35	35	35	35	37	38	**40**

ated by Algorithm 3 are composed of at most two relocations, the initial (RP) does not yield a good lower bound for this subset. Concerning the stack height limit, the initial gap is smaller for $T = \infty$ than for $T = H + 2$. This is one of the reasons why the instances with $T = \infty$ are easier to solve.

Number of iterations The proposed algorithm converges in a small number of iterations, which appears natural because the initial gap is already small. This also explains why the instances with $T = \infty$, for which the initial gap is smaller, were solved in a smaller number of iterations than those with $T = H + 2$. Given that the initial lower bound is optimal, the algorithm terminates in one iteration only when at least one of the following conditions is satisfied: the initial upper bound by the greedy heuristic is optimal, the solution of the initial (RP) is feasible for (IP), or the solution of the initial (UP) is optimal for (IP). For a few instances with $H = 4$ and 5 where the initial lower bound is optimal, none of these conditions is satisfied and, hence, the average number of iterations is not always equal to one for the subsets.

Problem size The size of (RP) in the proposed algorithm is considerably smaller than the IP formulation in BC-RBRP. For the instances with $H = 3$, the final (RP) has no more than 60 decision variables and 150 constraints on average. For the same instances, BC-RBRP requires approximately 4600 decision variables and 10,000 constraints at the maximum (on average). The number of decision variables in (RP) is around 5000 on average, even for the hardest instances with $H = 10$, although only instances solved to optimality in the time limit are considered for $T = H + 2$. This compactness is a notable advantage of the proposed formulation over that in BC-RBRP.

Computation time in each component The computation time for (RP) dominates that for (UP). This is not surprising because (UP) is composed only of complete relocation sequences, so it has fewer decision variables than (RP). In addition, it is often the case that (UP) is infeasible in early iterations, resulting in shorter computation time for (UP). In general, it is easier for the IP solver to detect the infeasibility of a problem than to actually find its optimal solution. The computation time spent in the other part, primarily for expanding relocation sequences and detecting their conflicts, is almost negligible.

5.4.3. Results for hard instances

As listed in Table 3, the proposed algorithm failed in solving 16 instances with $T = H + 2$ to optimality in the time limit of 1 hour. We managed to solve these hard instances to optimality using the multi-threaded algorithm without setting a time limit. The results are provided in Table 8, and indicate that 35 out of the 40 instances with $(H, S) = (10, 6)$ were solved to optimality in 1 hour, 37 in 3 hours, 38 in 6 hours, and 40 in 9 hours. On the other hand, it took 24 hours for 38 out of the 40 instances with $(H, S) = (10, 10)$, and 27 hours for the remaining two instances. We can conclude that our algorithm can solve instances with 100 blocks, although hard instances are still time-consuming.

Now that all instances have been solved to optimality. For future reference, optimal objective values are presented in Table C.2 in Appendix C.

5.5. Computational results for the ZQLZ dataset

Next, we briefly review the results for the ZQLZ dataset, which are summarized in Tables 9 and 10. As we can observe from the tables, the results are similar to those for the CVS dataset: BC-RBRP is more scalable than the branch-and-bound algorithm, and

Table 9
Computational results for the ZQLZ dataset (single thread).

T	*S*	*N*	*n*	b&b									IP-based					
				LB4			LB-LIS			UBALB			BC-RBRP			Proposed		
				#opt	ave	max	#opt	ave	max	#opt	ave	max	#opt	ave	max	#opt	ave	max
3	6	15–17	300	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.01	0.02	**300**	0.00	0.00
	7	18–20	300	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.01	0.04	**300**	0.00	0.00
	8	21–23	300	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.02	0.07	**300**	0.00	0.00
	9	24–26	300	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.03	0.08	**300**	0.00	0.00
	10	27–29	300	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.04	0.16	**300**	0.00	0.01
4	6	20–23	400	**400**	0.00	0.00	**400**	0.00	0.00	**400**	0.00	0.00	**400**	0.02	0.10	**400**	0.00	0.09
	7	24–27	400	**400**	0.00	0.00	**400**	0.00	0.00	**400**	0.00	0.00	**400**	0.04	0.55	**400**	0.00	0.11
	8	28–31	400	**400**	0.00	0.00	**400**	0.00	0.00	**400**	0.00	0.00	**400**	0.06	0.52	**400**	0.00	0.07
	9	32–35	400	**400**	0.00	0.01	**400**	0.00	0.01	**400**	0.00	0.02	**400**	0.09	0.96	**400**	0.01	1.03
	10	36–39	400	**400**	0.00	0.16	**400**	0.00	0.12	**400**	0.00	0.36	**400**	0.16	2.55	**400**	0.01	0.48
5	6	25–29	500	**500**	0.00	0.00	**500**	0.00	0.00	**500**	0.00	0.02	**500**	0.05	0.69	**500**	0.03	3.19
	7	30–34	500	**500**	0.00	0.02	**500**	0.00	0.01	**500**	0.00	0.02	**500**	0.13	2.78	**500**	0.05	1.68
	8	35–39	500	**500**	0.00	0.34	**500**	0.00	0.26	**500**	0.01	0.75	**500**	0.38	12.38	**500**	0.07	5.69
	9	40–44	500	**500**	0.01	1.27	**500**	0.00	0.98	**500**	0.01	4.58	**500**	0.59	55.22	**500**	0.10	3.85
	10	45–49	500	**500**	0.01	0.35	**500**	0.01	0.29	**500**	0.03	1.60	**500**	1.51	92.83	**500**	0.16	4.53
6	6	30–35	600	**600**	0.00	0.10	**600**	0.00	0.08	**600**	0.00	0.47	**600**	0.30	7.87	**600**	0.30	14.40
	7	36–41	600	**600**	0.01	0.58	**600**	0.00	0.46	**600**	0.06	8.13	**600**	1.21	63.59	**600**	0.49	23.75
	8	42–47	600	**600**	0.11	25.64	**600**	0.08	20.05	**600**	1.32	350.41	**600**	2.41	110.07	**600**	0.62	13.23
	9	48–53	600	**600**	1.41	444.22	**600**	1.15	358.11	597	23.16	3600.00	**600**	8.63	465.38	**600**	1.24	76.41
	10	54–59	600	599	8.57	3600.00	599	7.84	3600.00	597	36.91	3600.00	**600**	18.55	1455.19	**600**	1.35	31.51
7	6	35–41	700	**700**	0.07	20.79	**700**	0.06	18.26	**700**	1.63	584.64	**700**	3.67	182.91	**700**	1.98	112.12
	7	42–48	700	**700**	0.85	178.20	**700**	0.68	141.57	698	22.87	3600.00	**700**	18.24	2015.02	**700**	4.22	207.27
	8	49–55	700	698	22.51	3600.00	698	18.91	3600.00	690	89.13	3600.00	**700**	50.83	2670.23	**700**	5.52	337.62
	9	56–62	700	689	101.52	3600.00	692	88.02	3600.00	647	411.89	3600.00	690	152.38	3600.00	**700**	15.33	692.58
	10	63–69	700	666	311.98	3600.00	671	271.72	3600.00	590	832.63	3600.00	684	275.42	3600.00	**700**	18.50	3104.31
	Total		12,500	12,452			12,460			12,319			12,474			**12,500**		

Table 10
Computational results for the ZQLZ dataset (8 threads).

T	*S*	*N*	*n*	b&b									IP-based					
				LB4			LB-LIS			UBALB			BC-RBRP			Proposed		
				#opt	ave	max	#opt	ave	max	#opt	ave	max	#opt	ave	max	#opt	ave	max
3	6	15–17	300	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.01	0.02	**300**	0.00	0.00
	7	18–20	300	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.01	0.03	**300**	0.00	0.00
	8	21–23	300	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.02	0.08	**300**	0.00	0.01
	9	24–26	300	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.03	0.08	**300**	0.00	0.00
	10	27–29	300	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.00	0.00	**300**	0.04	0.15	**300**	0.00	0.00
4	6	20–23	400	**400**	0.00	0.00	**400**	0.00	0.00	**400**	0.00	0.00	**400**	0.02	0.11	**400**	0.00	0.09
	7	24–27	400	**400**	0.00	0.00	**400**	0.00	0.00	**400**	0.00	0.00	**400**	0.04	0.42	**400**	0.00	0.12
	8	28–31	400	**400**	0.00	0.00	**400**	0.00	0.01	**400**	0.00	0.01	**400**	0.06	0.49	**400**	0.00	0.08
	9	32–35	400	**400**	0.00	0.01	**400**	0.00	0.00	**400**	0.00	0.02	**400**	0.09	0.86	**400**	0.01	0.65
	10	36–39	400	**400**	0.00	0.04	**400**	0.00	0.03	**400**	0.00	0.07	**400**	0.15	1.96	**400**	0.01	0.49
5	6	25–29	500	**500**	0.00	0.00	**500**	0.00	0.00	**500**	0.00	0.00	**500**	0.05	0.45	**500**	0.02	1.09
	7	30–34	500	**500**	0.00	0.01	**500**	0.00	0.01	**500**	0.00	0.01	**500**	0.12	1.45	**500**	0.05	1.70
	8	35–39	500	**500**	0.00	0.07	**500**	0.00	0.06	**500**	0.00	0.12	**500**	0.28	5.48	**500**	0.07	3.39
	9	40–44	500	**500**	0.00	0.30	**500**	0.00	0.22	**500**	0.00	0.68	**500**	0.42	15.80	**500**	0.11	9.21
	10	45–49	500	**500**	0.00	0.08	**500**	0.00	0.08	**500**	0.01	0.36	**500**	1.11	28.90	**500**	0.17	4.71
6	6	30–35	600	**600**	0.00	0.02	**600**	0.00	0.02	**600**	0.00	0.09	**600**	0.23	6.03	**600**	0.24	9.01
	7	36–41	600	**600**	0.00	0.12	**600**	0.00	0.10	**600**	0.01	1.37	**600**	0.68	20.47	**600**	0.46	29.68
	8	42–47	600	**600**	0.03	7.67	**600**	0.02	6.69	**600**	0.30	98.51	**600**	1.50	30.24	**600**	0.59	11.07
	9	48–53	600	**600**	0.24	75.60	**600**	0.22	65.93	**600**	8.34	2877.82	**600**	5.25	291.40	**600**	0.95	46.62
	10	54–59	600	**600**	2.29	1040.86	**600**	2.16	1025.25	599	13.43	3600.00	**600**	9.71	430.17	**600**	1.19	24.04
7	6	35–41	700	**700**	0.01	3.44	**700**	0.01	3.36	**700**	0.36	111.20	**700**	1.69	96.09	**700**	1.52	57.89
	7	42–48	700	**700**	0.18	32.20	**700**	0.16	29.72	699	9.73	3600.00	**700**	7.07	404.46	**700**	2.72	114.85
	8	49–55	700	699	12.26	3600.00	698	11.92	3600.00	696	43.14	3600.00	**700**	22.37	1966.06	**700**	3.93	127.38
	9	56–62	700	696	41.79	3600.00	696	37.17	3600.00	675	218.51	3600.00	**700**	69.48	2042.54	**700**	9.81	449.88
	10	63–69	700	685	144.32	3600.00	685	132.44	3600.00	642	462.02	3600.00	695	142.65	3600.00	**700**	15.56	2926.28
	Total		12,500	12,480			12,479			12,411			12,495			**12,500**		

the proposed algorithm outperforms BC-RBRP. In particular, the proposed algorithm solved all 12,500 instances to proven optimality regardless of multi-threading or single-threading, whereas BC-RBRP could not solve 26 and 5 instances in single-threading and multi-threading, respectively. The optimal values are summarized in Table C.2 in Appendix C.

6. Conclusion

In this study, we have proposed new binary IP formulations of the RBRP with distinct priorities. We first formulated the problem of choosing optimal relocation sequences for individual blocks that satisfy conflict and capacity constraints. To avoid issues in handling an exponential number of relocation sequences, we have introduced a relaxed formulation, considering truncated relocation sequences. To obtain a feasible solution of the original problem, we derived another formulation from the relaxed formulation by excluding truncated relocation sequences. Subsequently, we constructed an exact approach that iteratively improves the relaxed formulation by expanding truncated relocation sequences until an optimal solution is obtained. Computational experiments for two benchmark datasets from the literature have proved the effectiveness and scalability of the proposed approach, which outperformed the state-of-the-art branch-and-bound algorithm and branch-and-cut algorithm. To the best of the authors' knowledge, our algorithm is the first to solve all instances of the datasets to proven optimality. However, it might be necessary to construct some solution method dedicated to the formulation if we want to further speed up the proposed approach for hard instances with many blocks. Another important research direction is to extend the formulation for the UBRP. In the case of the restricted problem, we decompose the planning horizon into periods, assuming that exactly one block is retrieved in each period. Such a decomposition is possible because each block is relocated at most once in one period. The derivation of conflicts between relocation sequences in Algorithm 5 also relies on this property. Unfortunately, this is not applicable for the unrestricted problem, and the same block may be relocated more than once between two successive retrievals. Therefore, the existing formulations of the unrestricted problem use relocation-based periods such that at most one relocation occurs in each period. In this case, however, the number of possible relocation sequences of blocks would increase, making the formulation less efficient. Alleviating this challenge is left for future research.

Acknowledgment

This work is supported by JSPS (Japan Society for the Promotion of Science) KAKENHI Grant Number JP18K04607.

Appendix A. Proof of Proposition 4

The proof is in two steps. First, it is shown that considering only invariant blocks does not increase the shortest length of complete relocation sequences. Let us denote by π^{A} the shortest complete sequence of block i when all blocks are considered, which is expressed as $\dots,\ (p_n, d_{n-1}, e_n),\ (o_{n+1}, e_n, e_{n+1}),\ \dots,\ (o_{n^{\mathrm{A}}}, e_{n^{\mathrm{A}}-1}, e_{n^{\mathrm{A}}})$, $(i, e_{n^{\mathrm{A}}}, e_{n^{\mathrm{A}}})$. Here, n^{A} is the number of relocations in π^{A}. From π^{A}, we construct a partial sequence π^{B} of length n^{A} expressed as $\dots,$ $(p_n, d_{n-1}, e_n),\ (o'_{n+1}, e_n, e_{n+1}),\ \dots,\ (o'_{n^{\mathrm{A}}}, e_{n^{\mathrm{A}}-1}, e_{n^{\mathrm{A}}})$, where only invariant blocks are considered after period p_n. In π^{B}, the period o'_k of the kth relocation $(k \geq n+1)$ is determined by the minimum priority of the destination stack in bay configuration $\mathcal{C}^{(p'_{k-1})}$. Therefore, $o'_k = q_{e_{k-1}}^{(o'_{k-1})}$ holds for any $k \geq n+1$ ($o'_n = p_n$ is assumed for ease of notation). On the other hand, $o_k \leq q_{e_{k-1}}^{(o_{k-1})}$ holds for any $k \geq n+1$ in π^{A}, because blocking blocks that have already been relocated and placed below block i in stack e_{k-1} may make the stack priority smaller than $q_{e_{k-1}}^{(o_{k-1})}$. Suppose that $o'_{k-1} \geq o_{k-1}$ holds. Then, $o'_k = q_{e_{k-1}}^{(o'_{k-1})} \geq q_{e_{k-1}}^{(o_{k-1})} \geq o_k$ holds from $q_s^{(p')} \geq q_s^{(p)}$ for any $p' > p$ and any $s \in \mathcal{S}$. It follows that $o'_k \geq o_k$ holds for any $k \geq n$ by induction, and $q_{e_{n^{\mathrm{A}}}}^{(o'_{n^{\mathrm{A}}})} \geq q_{e_{n^{\mathrm{A}}}}^{(o_{n^{\mathrm{A}}})} > i$ should hold. If we denote by n^{B} the smallest k that satisfies $q_{e_k}^{(o'_k)} > i$, n^{B} satisfies $n^{\mathrm{B}} \leq n^{\mathrm{A}}$. Moreover, a complete sequence expressed by $\dots,\ (p_n, d_{n-1}, e_n)$, $(o'_{n+1}, e_n, e_{n+1}),\ \dots,\ (o'_{n^{\mathrm{B}}}, e_{n^{\mathrm{B}}-1}, e_{n^{\mathrm{B}}}),\ (i, e_{n^{\mathrm{B}}}, e_{n^{\mathrm{B}}})$ is feasible when only invariant blocks are considered. This completes the first step of the proof.

Next, considering only invariant blocks, we show that the complete sequence π^{C} obtained by the algorithm is the shortest. Suppose that π^{C} is expressed by $\dots,\ (p_n, d_{n-1}, d_n),\ (p_{n+1}, d_n, d_{n+1})$, $\dots,\ (p_{n^{\mathrm{C}}}, d_{n^{\mathrm{C}}-1}, d_{n^{\mathrm{C}}}),\ (i, d_{n^{\mathrm{C}}}, d_{n^{\mathrm{C}}})$. Suppose further that there exists a shorter complete sequence π^{D} expressed by $\dots,\ (p_n, d_{n-1}, d'_n)$, $(p'_{n+1}, d'_n, d'_{n+1}),\ \dots,\ (p'_{n^{\mathrm{D}}}, d'_{n^{\mathrm{D}}-1}, d'_{n^{\mathrm{D}}}),\ (i, d'_{n^{\mathrm{D}}}, d'_{n^{\mathrm{D}}})$. where $n^{\mathrm{D}} < n^{\mathrm{C}}$. Let l be the minimum integer satisfying $d_l \neq d'_l$. That is, the first $(l-1)$ relocations in π^{D} are the same as those in π^{C}. Since $d_l \in \mathcal{L}^{(p_l)}$ is chosen to maximize the stack priority in period p_l in line 3 of Algorithm 4, we obtain $p'_{l+1} = q_{d'_l}^{(p_l)} \leq q_{d_l}^{(p_l)} = p_{l+1}$. If $p'_{l+1} = p_{l+1}$, $q_{d'_l}^{(p_l)} = q_{d_l}^{(p_l)}$ implies that both stacks d'_l and d_l should be empty in period p_l, leading to a contradiction because we do not need to relocate block i anymore, so that $n^{\mathrm{D}} = n^{\mathrm{C}} = l+1$ should hold. In the following, we assume $p'_{l+1} < p_{l+1}$. First, we note that $q_{d_l}^{(p'_{l+1})} \leq q_{d_l}^{(p_{l+1}-1)} = q_{d_l}^{(p_l)} = p_{l+1}$ holds because $p'_{l+1} \leq p_{l+1} - 1$, and invariant blocks in stack d_l do not change in periods $p_l,\ p_l + 1, \dots, p_{l+1} - 1$. As the destination stack d'_{l+1} of the $(l+1)$th relocation in π^{D} is chosen from $\mathcal{L}^{(p'_{l+1})}$, p'_{l+2} is smaller than or equal to the maximum stack priority among them. Therefore,

$$
\begin{aligned}
p'_{l+2} &\leq \max_{d \in \mathcal{L}^{(p'_{l+1})}} q_d^{(p'_{l+1})} \\
&= \max\left\{ \max_{d \in \mathcal{L}^{(p'_{l+1})} \setminus \{d_l\}} q_d^{(p'_{l+1})},\ q_{d_l}^{(p'_{l+1})} \right\} \\
&\leq \max\left\{ \max_{d \in \mathcal{L}^{(p'_{l+1})} \setminus \{d_l\}} q_d^{(p'_{l+1})},\ p_{l+1} \right\} \\
&\leq \max\left\{ \max_{d \in \mathcal{L}^{(p_{l+1})}} q_d^{(p_{l+1})},\ p_{l+1} \right\} \\
&= \max_{d \in \mathcal{L}^{(p_{l+1})}} q_d^{(p_{l+1})} = p_{l+2},
\end{aligned}
\tag{A.1}
$$

and we obtain $p'_{l+2} \leq p_{l+2}$. Here, we also applied the relations $\mathcal{L}^{(p'_{l+1})} \setminus \{d_l\} \subset \mathcal{L}^{(p_{l+1})}$ and $p_{l+1} < q_d^{(p_{l+1})}$. The first relation is derived from the fact that d_l is the target stack in period p_{l+1} and $d_l \notin \mathcal{L}^{(p_{l+1})}$. The second relation is trivial because p_{l+1} is the priority of the target stack in period p_{l+1}: p_{l+1} should be smaller than those of the other stacks. If $p'_{l+2} = p_{l+2}$ and, hence, $d'_{l+1} = d_{l+1}$ in (A.1), we can replace the relocations $(p_l, d_{l-1}, d'_l),\ (p'_{l+1}, d'_l, d'_{l+1})$ in π^{D} with $(p_l, d_{l-1}, d_l),\ (p_{l+1}, d_l, d'_{l+1})$. Now that we obtain a sequence where the first l relocations are the same as those in π^{C}, we search for a new l, the minimum index satisfying $d_l \neq d'_l$, which is larger than the original value of l. If $p'_{l+2} < p_{l+2}$, we can derive $p'_{l+3} \leq p_{l+3}$ in a similar manner. By repeatedly applying this argument, we know that π^{D} is at least as long as π^{C}, which is a contradiction.

Appendix B. Proof of Proposition 5

Let $\pi'' = (p_1, d_0, d_1), \ldots, (p_n, d_{n-1}, d_n), (p_{n+1}, d_n, d_{n+1}), \ldots, (p_{n''}, d_{n''-1}, d_{n''})$ be the relocation sequence obtained by Algorithm 6 for a partial relocation sequence $\pi = (p_1, d_0, d_1), \ldots, (p_n, d_{n-1}, \bullet)$. Here, $d_{n''}$ is used for the sake of convenience, although the algorithm does not compute it. Suppose that there exists a feasible relocation sequence $\pi' = (p_1, d_0, d_1), \ldots, (p_n, d_{n-1}, d'_n), (p'_{n+1}, d'_n, d'_{n+1}), \ldots, (p'_{n''}, d'_{n''-1}, d'_{n''})$ that satisfies $p'_{n''} < p_{n''}$. Let l be the minimum index satisfying $p_l \neq p'_l$. As Algorithm 6 minimizes the period of the lth relocation by construction, $p_l < p'_l$ holds. If $p'_l = q^{(p_{l-1})}_{d'_{l-1}}$, $d'_{l-1} \notin \mathcal{L}^{(p'_l)}$ holds because stack d'_{l-1} is the predetermined target stack in period p'_l. In addition, it does not become the predetermined target stack in periods $p_{l-1}, p_{l-1}+1, \ldots, p_l, \ldots, p'_l - 1$, thus $d'_{l-1} \in \mathcal{L}^{(p_l)}$ holds. It follows that stack d'_{l-1} remains a candidate for the destination stack in period p_l. Furthermore, $p'_l = q^{(p_{l-1})}_{d'_{l-1}} = q^{(p_{l-1}+1)}_{d'_{l-1}} = \cdots = q^{(p_l)}_{d'_{l-1}} = \cdots = q^{(p'_l-1)}_{d'_{l-1}}$ holds. Therefore, p'_l is a candidate for p_{l+1} in π''. If, on the other hand, $p'_l \in \mathcal{Q}^{(p_{l-1})}_i$, block i is relocated on top of a variant block p'_l. As this block is not yet retrieved in period $p_l < p'_l$, it remains in $\mathcal{Q}^{(p_l)}_i$ and is considered a candidate for p_{l+1}. In summary, $p_{l+1} \leq p'_l$ holds in both cases. If $p_{l+1} < p'_l$, we can show that $p_{l+2} \leq p'_l$ holds in a similar manner. Hence, there exists some $l' > l$ satisfying $p_{l'} = p'_l$. By repeatedly applying this argument to the remaining relocation sequences after period $p_{l'} = p'_l$, we can see that for any $k \geq n+1$, there exists $k' \geq k$ satisfying $p_{k'} = p'_k$. This implies $p_k \leq p'_k$ for any $k \geq n+1$, which contradicts the assumption $p'_{n''} < p_{n''}$.

Appendix C. Optimal values of datasets

The optimal values for the CVS and ZQLZ datasets are presented in Tables C.1 and C.2, respectively. In each table, "obj" is the average optimal value over each subset.

Table C1
Average optimal values for the CVS dataset. The number of instances in each subset is 40.

H	S	N	$T = H + 2$ obj	$T = \infty$ obj
3	3	9	5.000	5.000
	4	12	6.175	6.175
	5	15	7.025	7.025
	6	18	8.400	8.400
	7	21	9.275	9.275
	8	24	10.650	10.650
4	4	16	10.200	10.150
	5	20	12.950	12.950
	6	24	14.025	14.000
	7	28	16.125	16.125
5	4	20	15.425	15.275
	5	25	18.850	18.650
	6	30	22.075	21.925
	7	35	24.250	24.225
	8	40	27.700	27.650
	9	45	30.450	30.450
	10	50	33.275	33.150
6	6	36	30.875	30.475
	10	60	45.500	45.400
10	6	60	74.375	69.700
	10	100	104.750	102.050

Table C2
Average optimal values for the ZQLZ dataset. The number of instances in each subset is 100.

T	S	N	obj	T	S	N	obj	T	S	N	obj	T	S	N	obj
3	6	15	5.61	5	6	25	16.74	6	6	30	22.95	7	6	35	29.87
		16	7.12			26	18.59			31	24.76			36	33.04
		17	7.20			27	18.88			32	26.82			37	34.92
	7	18	6.90			28	19.69			33	28.73			38	35.87
		19	7.99			29	21.13			34	29.20			39	36.78
		20	8.43		7	30	20.11			35	29.29			40	38.74
	8	21	8.12			31	21.86		7	36	27.61			41	38.91
		22	9.08			32	22.68			37	29.12		7	42	35.68
		23	9.59			33	23.40			38	31.36			43	37.36
	9	24	9.76			34	24.54			39	31.76			44	40.94
		25	10.03		8	35	23.30			40	32.54			45	41.77
		26	11.33			36	24.73			41	33.62			46	42.16
	10	27	10.74			37	26.42		8	42	31.38			47	44.94
		28	11.74			38	26.82			43	34.02			48	44.86
		29	12.28			39	27.36			44	35.06		8	49	41.07
4	6	20	10.55		9	40	25.68			45	36.04			50	44.59
		21	12.31			41	26.98			46	36.73			51	44.96
		22	13.64			42	29.05			47	38.64			52	47.43
		23	13.53			43	30.11		9	48	35.05			53	48.02
	7	24	12.88			44	29.74			49	36.82			54	47.49
		25	14.31		10	45	28.86			50	39.77			55	50.19
		26	14.78			46	30.58			51	41.22		9	56	46.19
		27	16.04			47	31.55			52	42.02			57	48.29
	8	28	15.78			48	32.36			53	42.21			58	50.82
		29	16.30			49	33.91		10	54	39.44			59	53.18
		30	17.17							55	40.75			60	54.11
		31	17.64							56	43.13			61	55.25
	9	32	16.64							57	43.63			62	55.52
		33	18.47							58	45.48		10	63	51.86
		34	19.42							59	47.41			64	53.88
		35	19.33											65	56.54
	10	36	19.02											66	56.51
		37	20.06											67	59.04
		38	21.27											68	59.50
		39	21.81											69	61.78

References

Azari, E., Eskandari, H., & Nourmohammadi, A. (2017). Decreasing the crane working time in retrieving the containers from a bay. *Scientia Iranica, 24*, 309–318. https://doi.org/10.24200/sci.2017.4035.

Bacci, T., Mattia, S., & Ventura, P. (2019). The bounded beam search algorithm for the block relocation problem. *Computers & Operations Research, 103*, 252–264. https://doi.org/10.1016/j.cor.2018.11.008.

Bacci, T., Mattia, S., & Ventura, P. (2020). A branch-and-cut algorithm for the restricted block relocation problem. *European Journal of Operational Research, 287*, 452–459. https://doi.org/10.1016/j.ejor.2020.05.029.

Caserta, M., Schwarze, S., & Voß, S. (2009). A new binary description of the blocks relocation problem and benefits in a look ahead heuristic. *Lecture Notes in Computer Science, 5482*, 37–48. https://doi.org/10.1007/978-3-642-01009-5_4.

Caserta, M., Schwarze, S., & Voß, S. (2011a). Container rehandling at maritime container terminals. In *Handbook of terminal planning, operations research/computer science interfaces series 49* (pp. 247–269). New York: Springer. https://doi.org/10.1007/978-1-4419-8408-1_13.

Caserta, M., Schwarze, S., & Voß, S. (2012). A mathematical formulation and complexity considerations for the blocks relocation problem. *European Journal of Operational Research, 219*, 96–104. https://doi.org/10.1016/j.ejor.2011.12.039.

Caserta, M., & Voß, S. (2009). Corridor selection and fine tuning for the corridor method. *Lecture Notes in Computer Science, 5851*, 163–175. https://doi.org/10.1007/978-3-642-11169-3_12.

Caserta, M., Voß, S., & Sniedovich, M. (2011b). Applying the corridor method to a blocks relocation problem. *OR Spectrum, 33*, 915–929. https://doi.org/10.1007/s00291-009-0176-5.

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

Expósito-Izquierdo, C., Melián-Batista, B., & Moreno-Vega, J. (2014). A domain-specific knowledge-based heuristic for the blocks relocation problem. *Advanced Engineering Informatics, 28*, 327–343. https://doi.org/10.1016/j.aei.2014.03.003.

Expósito-Izquierdo, C., Melián-Batista, B., & Moreno-Vega, J. (2015). An exact approach for the blocks relocation problem. *Expert Systems with Applications, 42*, 6408–6422. https://doi.org/10.1016/j.eswa.2015.04.021.

Feillet, D., Parragh, S., & Tricoire, F. (2019). A local-search based heuristic for the unrestricted block relocation problem. *Computers & Operations Research, 108*, 44–56. https://doi.org/10.1016/j.cor.2019.04.006.

Forster, F., & Bortfeldt, A. (2012). A tree search procedure for the container relocation problem. *Computers & Operations Research, 39*, 299–309. https://doi.org/10.1016/j.cor.2011.04.004.

Galle, V., Barnhart, C., & Jaillet, P. (2018). A new binary formulation of the restricted container relocation problem based on a binary encoding of configurations. *European Journal of Operational Research, 267*, 467–477. https://doi.org/10.1016/j.ejor.2017.11.053.

Jin, B., Zhu, W., & Lim, A. (2014). Solving the container relocation problem by an improved greedy look-ahead heuristic. *European Journal of Operational Research, 240*, 837–847. https://doi.org/10.1016/j.ejor.2014.07.038.

Jovanovic, R., Tuba, M., & Voß, S. (2019). An efficient ant colony optimization algorithm for the blocks relocation problem. *European Journal of Operational Research, 274*, 78–90. https://doi.org/10.1016/j.ejor.2018.09.038.

Jovanovic, R., & Voß, S. (2014). A chain heuristic for the blocks relocation problem. *Computers & Industrial Engineering, 75*, 79–86. https://doi.org/10.1016/j.cie.2014.06.010.

Kim, K., & Hong, G. P. (2006). A heuristic rule for relocating blocks. *Computers & Operations Research, 33*, 940–954. https://doi.org/10.1016/j.cor.2004.08.005.

König, F., Lübbecke, M., Möhring, R., Schäfer, G., & Spenke, I. (2007). Solutions to real-world instances of PSPACE-complete stacking. *Lecture Notes in Computer Science, 4698*, 729–740. https://doi.org/10.1007/978-3-540-75520-3_64.

Ku, D., & Arthanari, T. (2016). On the abstraction method for the container relocation problem. *Computers & Operations Research, 68*, 110–122. https://doi.org/10.1016/j.cor.2015.11.006.

Lee, Y., & Lee, Y. J. (2010). A heuristic for retrieving containers from a yard. *Computers & Operations Research, 37*, 1139–1147. https://doi.org/10.1016/j.cor.2009.10.005.

Lin, D. Y., Lee, Y. J., & Lee, Y. (2015). The container retrieval problem with respect to relocation. *Transportation Research Part C: Emerging Technologies, 52*, 132–143. https://doi.org/10.1016/j.trc.2015.01.024.

Lu, C., Zeng, B., & Liu, S. (2020). A study on the block relocation problem: Lower bound derivations and strong formulations. *IEEE Transactions on Automation Science and Engineering, 17*, 1829–1853. https://doi.org/10.1109/TASE.2020.2979868.

Petering, M., & Hussein, M. (2013). A new mixed integer program and extended look-ahead heuristic algorithm for the block relocation problem. *European Journal of Operational Research, 231*, 120–130. https://doi.org/10.1016/j.ejor.2013.05.037.

Quispe, K., Lintzmayer, C., & Xavier, E. (2018). An exact algorithm for the blocks relocation problem with new lower bounds. *Computers & Operations Research, 99*, 206–217. https://doi.org/10.1016/j.cor.2018.06.021.

de Melo da Silva, M., Erdoğan, G., Battarra, M., & Strusevich, V. (2018a). The block retrieval problem. *European Journal of Operational Research, 265*, 931–950. https://doi.org/10.1016/j.ejor.2017.08.048.

de Melo da Silva, M., Toulouse, S., & Wolfler Calvo, R. (2018b). A new effective unified model for solving the pre-marshalling and block relocation problems. *European Journal of Operational Research, 271*, 40–56. https://doi.org/10.1016/j.ejor.2018.05.004.

Tanaka, S., & Mizuno, F. (2018). An exact algorithm for the unrestricted block relocation problem. *Computers & Operations Research, 95*, 12–31. https://doi.org/10.1016/j.cor.2018.02.019.

Tanaka, S., & Takii, K. (2016). A faster branch-and-bound algorithm for the block relocation problem. *IEEE Transactions on Automation Science and Engineering, 13*, 181–190. https://doi.org/10.1109/TASE.2015.2434417.

Ting, C. J., & Wu, K. C. (2017). Optimizing container relocation operations at container yards with beam search. *Transportation Research Part E: Logistics and Transportation Review, 103*, 17–31. https://doi.org/10.1016/j.tre.2017.04.010.

Tricoire, F., Scagnetti, J., & Beham, A. (2018). New insights on the block relocation problem. *Computers & Operations Research, 89*, 127–139. https://doi.org/10.1016/j.cor.2017.08.010.

Ünlüyurt, T., & Aydın, C. (2012). Improved rehandling strategies for the container retrieval process. *Journal of Advanced Transportation, 46*, 378–393. https://doi.org/10.1002/atr.1193.

Wan, Y. W., Liu, J., & Tsai, P. C. (2009). The assignment of storage locations to containers for a container stack. *Naval Research Logistics, 56*, 699–713. https://doi.org/10.1002/nav.20373.

Zehendner, E., Caserta, M., Feillet, D., Schwarze, S., & Voß, S. (2015). An improved mathematical formulation for the blocks relocation problem. *European Journal of Operational Research, 245*, 415–422. https://doi.org/10.1016/j.ejor.2015.03.032.

Zehendner, E., & Feillet, D. (2014). A branch and price approach for the container relocation problem. *International Journal of Production Research, 52*, 7159–7176. https://doi.org/10.1080/00207543.2014.965358.

Zhang, C., Guan, H., Yuan, Y., Chen, W., & Wu, T. (2020). Machine learning-driven algorithms for the container relocation problem. *Transportation Research Part B: Methodological, 139*, 102–131. https://doi.org/10.1016/j.trb.2020.05.017.

Zhu, W., Qin, H., Lim, A., & Zhang, H. (2012). Iterative deepening A* algorithms for the container relocation problem. *IEEE Transactions on Automation Science and Engineering, 9*, 710–722. https://doi.org/10.1109/TASE.2012.2198642.