MIT Sloan School of Management

MIT Sloan School Working Paper 5199-17

A new binary integer program for the restricted container relocation problem

Virgile Galle, Cynthia Barnhart, Patrick Jaillet

June 2017

A new binary integer program for the restricted container relocation problem

V. Galle * C. Barnhart † P. Jaillet ‡

Abstract

The Container Relocation Problem (CRP), also called Block Relocation Problem (BRP), is concerned with finding a sequence of moves of containers that minimizes the number of relocations needed to retrieve all containers, while respecting a given order of retrieval. The restricted CRP enforces that only containers blocking the target container can be relocated. We improve upon and enhance an existing binary encoding and using it, formulate the restricted CRP as a binary integer programming problem in which we exploit structural properties of the optimal solution. This integer programming formulation reduces significantly the number of variables and constraints compared to existing formulations. Its efficiency is shown through computational results on small and medium sized instances taken from the literature.

1 Introduction

Due to limited space in the storage area in maritime ports, containers are stacked on top of each other. The resulting stacks create rows of containers as shown in Figure 1. If a container that needs to be retrieved (*target* container) is not located at the top and is covered by other containers, *blocking* containers must be relocated. As a result, during the retrieval process, one or more *relocation* moves are performed by the yard cranes. Such relocations (also called reshuffles) are costly for the port operators and result in delays in the retrieval process. The *Container Relocation Problem (CRP)* (also known as the Block Relocation Problem) addresses this challenge by minimizing the number of relocations. The CRP applies to a broad range of two-dimensional storage systems involving containers, boxes, pallets or steel plates.

Mathematical modeling of the CRP usually represents one row with a two dimensional array of size (T, S), where T is the maximum height and S is the number of stacks. Tiers are numbered from bottom (1) to top (T) and stacks from left (1) to right (S). We refer to this array as a *configuration*. The initial configuration has C containers. In order for the problem to always be feasible, we suppose that the triplet (T, S, C) satisfies $0 \leqslant C \leqslant ST - (T-1)$. Indeed, $T-1$ empty slots would be needed in order to relocate a maximum of $T-1$ blocking containers above the target container (see Wan et al. (2009)). Each container is identified by a unique integer indicating its relative retrieval order. At any point in the retrieval process, the current target container is the container currently in the configuration with the smallest number. Container c is said to be *blocking* if there exists Container d in a lower tier of the same stack such that $d < c$.

During the retrieval process, a container can only be retrieved/relocated if it is at the top of its stack, *i.e.*, no other container is blocking it. Most importantly, *a container can only be relocated if it is blocking*

*Operations Research Center, MIT. Email: vgalle@mit.edu, **corresponding author**

†Operations Research Center and Civil & Environmental Engineering, MIT. Email: cbarnhar@mit.edu

‡Operations Research Center and Electrical Engineering & Computer Science, MIT. Email: jaillet@mit.edu

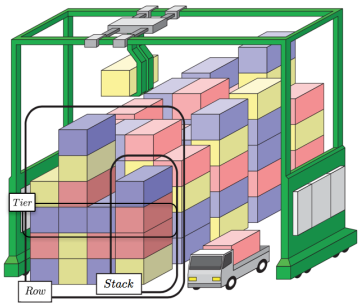

Figure 1: Stacks of containers in a storage yard (Tanaka and Takii (2016))

the target container (Assumption A_1 in Caserta et al. (2012)). The CRP under these assumptions, referred to as *restricted CRP*, involves finding a sequence of moves to retrieve Containers $1, 2, \ldots, C$ (respecting the order) with a minimum number of relocations. Figure 2 provides a simple example of the CRP.

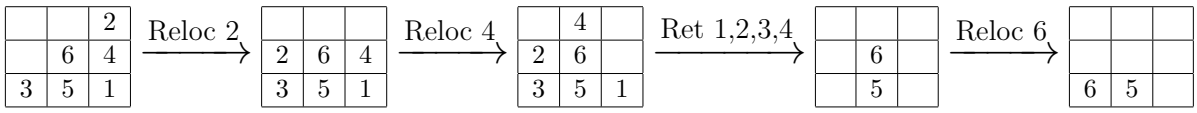

Figure 2: Configuration for the CRP with 3 tiers, 3 stacks and 6 containers. The optimal solution performs 3 relocations: relocate the container labeled 2 from Stack 3 to Stack 1 on the top of the container labeled 3; relocate 4 from 3 to 2 on the top of 6; retrieve 1; retrieve 2; retrieve 3; retrieve 4; relocate 6 from 2 to the empty Stack 1; retrieve 5; finally, retrieve 6.

As the problem with and without Assumption A_1 is NP-hard (Caserta et al. (2012)), many heuristics have been developed to solve this problem. This paper focuses on finding optimal solutions for the CRP. There are several exact methods worth mentioning such as algorithms based on A^* (Zhu et al. (2012), Borjian et al. (2015), Tanaka and Takii (2016)), and branch-and-bound (B&B) approaches (Ünlüyurt and Aydın (2012), Expósito-Izquierdo et al. (2015)).

As the method suggested in this paper is based on solving a binary integer programming (IP) formulation, we review here alternative IP models for the restricted CRP. Wan et al. (2009) formulate one of the first integer programming models for the CRP and develop an IP-based heuristic capable of obtaining near-optimal solutions. Caserta et al. (2012) propose another intuitive formulation of the problem, called BRP-II, as well as an efficient heuristic. Expósito-Izquierdo et al. (2015) correct BRP-II and rename their new formulation BRP-II*. Eskandari and Azari (2015) also correct BRP-II and propose an improved formulation called BRP2ci by adding valid inequalities. Zehendner et al. (2015) correct and improve BRP-II to get formulation BRP-II-A by removing some variables, tightening some constraints, introducing a new upper bound, and applying a pre-processing step to fix several variables. Finally, Tang et al. (2015) propose a very similar formulation with fewer variables than in BRP-II, present heuristics and a worst case analysis.

Carlo et al. (2014) and Lehnfeld and Knust (2014) review and survey the existing literature on the CRP and related problems, some of which are mentioned below. The unrestricted version of the CRP relaxes Assumption A_1. Caserta et al. (2012) and Petering and Hussein (2013) develop formulations for the unrestricted CRP, but both are unable to solve small-sized instances efficiently. Different objective functions have been considered such as the crane travel time, trucks' waiting times or weighted relocations. López-Plata et al. (2017) propose a binary IP to minimize waiting times. Priorities could also be given among groups of blocks, and Zhu et al. (2012) and Tanaka and Takii (2016) consider B&B approaches for this case. Borjian et al. (2013) and Akyüz and Lee (2013) propose two different MIP formulations for the dynamic version of the CRP where incoming containers have to be stacked. We also mention that Tang

et al. (2012) solve the BRP using integer programming in the case of steel plates, for which our approach can also be applied.

In order to evaluate the efficiency of these methods, several sets of instances have been used. The most common one appears in Caserta et al. (2009) and is used by Caserta et al. (2012), Zhu et al. (2012), Petering and Hussein (2013), Borjian et al. (2013), Eskandari and Azari (2015), Expósito-Izquierdo et al. (2015) and Zehendner et al. (2015). In these instances, T and S range from 3 to 10, C is taken to be $(T-2)S$ with $T-2$ containers per stack resulting in 21 classes of problem. With 40 instances per class, this set contains a total of 840 instances. We use these instances in Section 4 to test the efficiency of our new formulation. Instances from Lee and Lee (2010) consider multiple rows and are used to test heuristics. Instances with distinct priorities from Zhu et al. (2012) could be used, but most integer programs are tested on the set of Caserta et al. (2009) hence our preference for this set.

The major contribution of this paper is a new binary IP formulation for the restricted CRP referred to as CRP-I and presented in Section 3. This formulation is novel in several ways. First, CRP-I takes a different modeling approach compared to previous mathematical programming formulations; it builds upon a binary encoding introduced by Caserta et al. (2009) that has never before been used in exact solution methods. We identify and formulate properties of the encoding as linear equalities or inequalities (Section 2.1) in CRP-I and use structural properties of the optimal solution to enhance the tractability of our approach (see Sections 2.2 and 3.1). Finally, we show through extensive computational experiments on small and medium instances that CRP-I outperforms existing mathematical programming formulations and other exact methods (see Section 4).

This paper is organized as follows. Section 2 introduces some preliminary concepts of the binary encoding and structural properties of the optimal solution. Section 3 presents formulation CRP-I and suggests improvements for this new formulation, such as a revisited pre-processing step. Section 4 presents the results of computational experiments and Section 5 concludes this paper.

2 Preliminaries

2.1 Binary encoding

Most integer program formulations mentioned in Section 1 use the matrix representation of a configuration, *i.e.*, they introduce binary variables of the type y_{tscn} which indicates if container c is in tier t of stack s before the n^{th} move is performed.

Our binary IP models a configuration using an enhanced version of the binary encoding introduced by Caserta et al. (2009). First, Caserta et al. (2009) define S *artificial* containers identified by integers from $C+1,\ldots,C+S$, where artificial container c is under all containers in stack $c-C$ (see Figure 3). These artificial containers are used to keep track of which containers are in which stacks. Using these containers, they encode the classical matrix representation of the configuration into a binary matrix denoted by A of size $(C+S)\times(C+S)$. We consider the same encoding but decrease its size to $(C+S)\times C$. Rows indexed from $1,\ldots,C$ and columns of A represent the C *real* containers, while rows indexed from $C+1,\ldots,C+S$ correspond to the S artificial containers.

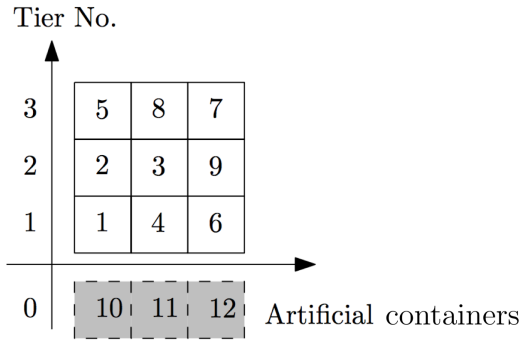

Figure 3: Configuration including artificial containers (example from Caserta et al. (2009))

The binary encoding of the configuration in Figure 3 is

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

By using the matrix A in this example, we can quickly access valuable information. For instance, consider container 1. The first row indicates that container 1 is below containers 2 and 5 and the first column shows that container 1 is above artificial container 10. Because this container represents stack $10 - 9 = 1$, it indicates that container 1 is in the first stack.

Formally, the binary encoding $A \in \{0,1\}^{(C+S)\times C}$ is defined such that such that

$$\forall\ c \in \{1,\ldots,C+S\}, d \in \{1,\ldots,C\},\ A_{c,d} = \begin{cases} 1 & \text{if container } c \text{ is below container } d, \\ 0 & \text{otherwise.} \end{cases}$$

New properties of the binary encoding For the rest of Section 2.1, we describe new properties of this binary encoding as well as novel rules to transform it after each retrieval. Each of these properties and rules are formulated using linear equalities or inequalities and incorporated into our mathematical programming formulation.

The binary encoding A has the following properties. Given a target container $n \in \{1,\ldots,C\}$:

1. Only containers $n,\ldots,C$ remain in the configuration. Thus, we only need to keep track of rows

$c \in \{n, \dots, C+S\}$ and columns $d \in \{n, \dots, C\}$ since:

$$\forall\ c \in \{1, \dots, n-1\},\ \begin{cases} \forall\ d \in \{1, \dots, C\},\ A_{c,d} = 0, \\ \forall\ d \in \{1, \dots, C+S\},\ A_{d,c} = 0. \end{cases}$$

Therefore, we consider that $A \in \{0,1\}^{(C+S-n+1)\times(C-n+1)}$.

2. A container is in one stack, and so is blocking exactly one artificial container:

$$\forall\ c \in \{n, \dots, C\},\ \sum_{s=1}^{S} A_{C+s,c} = 1.$$

3. A container cannot be blocking itself so the upper diagonal of A only contains zeros:

$$\forall\ c \in \{n, \dots, C\},\ A_{c,c} = 0.$$

4. Two distinct containers c and d cannot be mutually blocking each other:

$$\forall c, d \in \{n, \dots, C\},\ c \neq d,\ A_{c,d} + A_{d,c} \leqslant 1.$$

5. If two distinct containers c and d are in the same stack s, then either c is below d or d is below c ($A_{c,d} + A_{d,c} = 1$). We formulate this property as:

$$\forall\ s \in \{1, \dots, S\},\ c, d \in \{n, \dots, C\},\ c \neq d,\ A_{c,d} + A_{d,c} \geqslant A_{C+s,c} + A_{C+s,d} - 1.$$

There are three cases to consider:

- If c and d are in stack s, *i.e.*, $A_{C+s,c} = A_{C+s,d} = 1$, then this equation implies $A_{c,d} + A_{d,c} \geqslant 1$. Using the previous property, we have $A_{c,d} + A_{d,c} = 1$, which is the relation we are looking for.
- If either c or d is not in stack s, then this equation implies $A_{c,d} + A_{d,c} \geqslant 0$, which holds in any case since $A_{c,d}, A_{d,c} \in \{0,1\}$.
- If c and d are not in stack s, then this equation implies $A_{c,d} + A_{d,c} \geqslant -1$, which holds in any case since $A_{c,d}, A_{d,c} \in \{0,1\}$.

6. If two distinct containers c and d are in different stacks then neither c can be below d or d below c ($A_{c,d} = 0$ and $A_{d,c} = 0$). This property can be written as:

$$\forall\ s \in \{1, \dots, S\},\ c, d \in \{n, \dots, C\},\ c \neq d,\ A_{c,d} + A_{d,c} \leqslant 2 - A_{C+s,c} - \sum_{\substack{r=1 \\ r \neq s}}^{S} A_{C+r,d}.$$

Similarly to the previous property, we consider three cases :

- If c is in stack s but not d, *i.e.*, $A_{C+s,c} = 1$ and $\sum_{\substack{r=1 \\ r \neq s}}^{S} A_{C+r,d} = 1$, then this equation implies $A_{c,d} + A_{d,c} \leqslant 0$. Since $A_{c,d}, A_{d,c} \in \{0,1\}$, we must have $A_{c,d} = 0$ and $A_{d,c} = 0$ which is the property we are looking for.

- If both or neither c and d are in stack s, then this equation implies $A_{c,d} + A_{d,c} \leqslant 1$ (which holds since it is the fourth property of the binary encoding).
- If c is not in stack s but d is, then this equation implies $A_{c,d} + A_{d,c} \leqslant 2$ which always holds since $A_{c,d}, A_{d,c} \in \{0,1\}$.

7. The number of containers blocking c is given by $\sum_{d=n+1}^{C} A_{c,d}$. In particular, the number of relocations to retrieve the target container n is $\sum_{d=n+1}^{C} A_{n,d}$. Moreover, the height of stack $s \in \{1,\dots,S\}$ is the number of containers blocking artificial container $C+s$, *i.e.*, $\sum_{d=n+1}^{C} A_{C+s,d}$.

8. The total number of blocking containers in A, denoted by $b(A)$, is given by $b(A) = \sum_{d=n+1}^{C} B_d$, where $B_d \in \{0,1\}$ is a binary indicator that container d is blocking which can be computed as:

   $$\forall\ d \in \{n+1,\dots,C\},\ c \in \{n,\dots,d-1\},\ B_d \geqslant A_{c,d}.$$

   Note that the total number of blocking containers is not $\sum_{c=n}^{C-1} \sum_{d=c+1}^{C} A_{c,d}$, as a blocking container can be blocking several containers but should count only once.

Apart from offering a simple representation of a given configuration, this binary encoding presents another great advantage. Tracking the evolution of the configuration with the classical matrix representation is fairly intuitive but requires a significant number of constraints and seems to be one of the bottlenecks of previous formulations. Our representation allows us to express efficiently the transformation of the matrix A when performing the necessary relocations in order to retrieve the target container.

Let $n \in \{1,\dots,C-1\}$, we denote $A \in \{0,1\}^{(C+S-n+1)\times(C-n+1)}$ (respectively, $A' \in \{0,1\}^{(C+S-n)\times(C-n)}$) the binary encoding of the configuration when n is the target container (respectively, when $n+1$ is the target container and relocations to retrieve n have been performed). A' and A should verify the three following rules:

A. The restricted assumption states that if Container c is not blocking the target Container n then c cannot be relocated. Therefore all containers below c cannot be relocated either, hence columns c in A and A' should be identical, *i.e.*, if $A_{n,c} = 0$, then $\forall\ d \in \{n+1,\dots,C\}$, $A'_{d,c} = A_{d,c}$. This can be formulated by the following two inequalities:

   $$\forall\ c,d \in \{n+1,\dots,C\},\ \begin{cases} A'_{d,c} \leqslant A_{d,c} + A_{n,c}, \\ A'_{d,c} \geqslant A_{d,c} - A_{n,c}. \end{cases}$$

   If $A_{n,c} = 0$, both inequalities imply the property we are looking for. If $A_{n,c} = 1$, the inequalities imply $A_{d,c} - 1 \leqslant A'_{d,c} \leqslant A_{d,c} + 1$ which always holds since $A_{d,c} - 1 \leqslant 0$, $A_{d,c} + 1 \geqslant 1$ and $A'_{d,c} \in \{0,1\}$.

B. Any relocated container cannot stay in the same stack, which can be formulated as:

   $$\forall\ c \in \{n+1,\dots,C\},\ s \in \{1,\dots,S\},\ A_{n,c} + A_{C+s,c} + A'_{C+s,c} \leqslant 2.$$

   We consider two cases to justify this formulation:

- If Container c is blocking the target Container n (hence relocated) and is in stack s, *i.e.*, $A_{n,c} = A_{C+s,c} = 1$, then the inequality implies that $A'_{C+s,c} = 0$ which is the property we are aiming for.
- If either Container c is not blocking the target Container n or is not in stack s, then the inequality implies that $A'_{C+s,c} \leqslant 1$ which always holds.

C. If two distinct Containers c and d are blocking Container n (hence relocated) and d is above c when n is the target container, then d cannot remain above c (due to the LIFO policy of stacks), *i.e.* if $A_{n,c} = A_{n,d} = A_{c,d} = 1$, then $A'_{c,d} = 0$. This can be expressed as:

$$\forall\ c, d \in \{n+1, \ldots, C\},\ c \neq d,\ A_{n,c} + A_{n,d} + A_{c,d} + A'_{c,d} \leqslant 3.$$

Again, we justify this formulation by considering two cases:

- if $A_{n,c} = A_{n,d} = A_{c,d} = 1$, the inequality clearly implies $A'_{c,d} = 0$.
- if either $A_{n,c} = 0$, $A_{n,d} = 0$ or $A_{c,d} = 0$, then the inequality implies that $A'_{c,d} \leqslant 1$ which always holds.

2.2 Minimum number of relocations when $C \leqslant S$

We use a result from Galle et al. (2017) which we restate below with our current notations.

Lemma 1 (Galle et al. (2017))**.** *Consider a configuration with T tiers, S stacks, and C containers with binary representation $A \in \{0,1\}^{(C+S)\times C}$. Let $b(A)$ be the number of blocking containers in A which can be computed as*

$$b(A) = \sum_{d=2}^{C} B_d \text{ with } B_d \in \{0,1\}\ ,\ B_d \geqslant A_{c,d}, \forall\ c \in \{1, \ldots, d-1\}.$$

If $C \leqslant S$, then the minimum number of relocations to retrieve all containers from A is equal to $b(A)$.

Lemma 1 states that when the number of containers is less than the number of stacks ($C \leqslant S$), the minimum total number of relocations is simply the number of blocking containers. From now on, we suppose that $C > S$ and we define

$$N = C - S + 1 > 1.$$

Let us explain the use of Lemma 1. Let $A^{(1)}$ (respectively, $A^{(N)}$) be the binary encoding of an initial configuration (respectively, the configuration after $N-1$ retrievals). We define $r\left(A^{(1)}, A^{(N)}\right)$ to be the minimum number of relocations to reach $A^{(N)}$ from $A^{(1)}$. By definition, $A^{(N)} \in \{0,1\}^{(C'+S)\times C'}$, where $C' = C - N + 1 = S \leqslant S$. Hence we can apply Lemma 1 to $A^{(N)}$ and the minimum number of relocations to retrieve all containers from $A^{(N)}$ is equal to $b\left(A^{(N)}\right)$. Therefore, the minimum number of relocations to retrieve all containers from $A^{(1)}$ is $r\left(A^{(1)}, A^{(N)}\right) + b\left(A^{(N)}\right)$. Usually, methods track all relocations until there are no containers left in the configuration. Lemma 1 allows us to restrict our focus on the first $N-1$ retrievals as the problem becomes "trivial" when there are only S containers left.

3 New binary IP formulation

Using Section 2, we can now present our novel formulation that we denote by CRP-I. We suppose that we are given as inputs a feasible triplet (T, S, C) and $A^{(1)}$ the binary encoding of the initial configuration (which verify all the properties of the binary encoding).

CRP-I uses two kinds of variables,

$$\forall\, n \in \{1,\ldots,N\}, c \in \{n,\ldots,C+S\}, d \in \{n,\ldots,C\},$$
$$a_{n,c,d} = \begin{cases} 1 & \text{if container } c \text{ is below container } d \text{ when } n \text{ is the target container,} \\ 0 & \text{otherwise,} \end{cases}$$

and

$$\forall\, d \in \{N+1,\ldots,C\},$$
$$b_d = \begin{cases} 1 & \text{if container } d \text{ is blocking after } N-1 \text{ retrievals ,} \\ 0 & \text{otherwise.} \end{cases}$$

Binary variables a are used to keep track of the binary encoding of the configurations and compute the number of relocations incurred for the first $N-1$ retrievals. Note that we use the first property of binary encoding to limit the number of variables as n increases. Binary variables b are used to compute the number of blocking containers in the configuration after the $N-1$ retrievals. The number of variables is of the order of C^3, where the exact number is

$$\sum_{n=1}^{N-1} (C+S-n+1) \times (C-n+1) + S = \frac{1}{6}(C-S+1)\left(2C^2+5CS+C+S(5S-1)\right) + S.$$

The objective and constraints of CRP-I are presented below

$$Min\left(\sum_{n=1}^{N-1}\sum_{d=n+1}^{C} a_{n,n,d} + \sum_{d=N+1}^{C} b_d\right)$$

s.t.

$$a_{1,c,d} = A^{(1)}_{c,d}, \qquad \forall\, c \in \{1,\ldots,C+S\}, d \in \{1,\ldots,C\} \tag{1}$$

$$\sum_{s=1}^{S} a_{n,C+s,c} = 1, \qquad \forall\, n \in \{2,\ldots,N\}, c \in \{n,\ldots,C\} \tag{2}$$

$$a_{n,c,c} = 0, \qquad \forall\, n \in \{2,\ldots,N\}, c \in \{n,\ldots,C\} \tag{3}$$

$$a_{n,c,d} + a_{n,d,c} \leqslant 1, \qquad \forall\, n \in \{2,\ldots,N\}, c \in \{n,\ldots,C\}, d \in \{n,\ldots,C\} \setminus \{c\} \tag{4}$$

$$a_{n,c,d} + a_{n,d,c} \geqslant a_{n,C+s,c} + a_{n,C+s,d} - 1, \quad \forall\, n \in \{2,\ldots,N\}, s \in \{1,\ldots,S\}, c \in \{n,\ldots,C\}, d \in \{n,\ldots,C\} \setminus \{c\} \tag{5}$$

$$a_{n,c,d} + a_{n,d,c} \leqslant 2 - a_{n,C+s,c} - \sum_{\substack{r=1 \\ r \neq s}}^{S} a_{n,C+r,d}, \quad \forall\, n \in \{2,\ldots,N\}, s \in \{1,\ldots,S\}, c \in \{n,\ldots,C\}, d \in \{n,\ldots,C\} \setminus \{c\} \tag{6}$$

$$\sum_{d=n}^{C} a_{n,C+s,d} \leqslant T, \qquad \forall\, n \in \{2,\ldots,N\}, s \in \{1,\ldots,S\} \tag{7}$$

$$b_d \geqslant a_{N,c,d}, \qquad \forall\, d \in \{N+1,\ldots,C\}, c \in \{N,\ldots,d-1\} \tag{8}$$

$$a_{n+1,d,c} \leqslant a_{n,d,c} + a_{n,n,c}, \quad \forall\, n \in \{1,\ldots,N-1\}, c \in \{n+1,\ldots,C\}, d \in \{n+1,\ldots,C+S\} \setminus \{c\} \tag{9}$$

$$a_{n+1,d,c} \geqslant a_{n,d,c} - a_{n,n,c}, \quad \forall\, n \in \{1,\ldots,N-1\}, c \in \{n+1,\ldots,C\}, d \in \{n+1,\ldots,C+S\} \setminus \{c\} \tag{10}$$

$$a_{n,n,c} + a_{n,C+s,c} + a_{n+1,C+s,c} \leqslant 2, \quad \forall\, n \in \{1,\ldots,N-1\}, c \in \{n+1,\ldots,C\}, s \in \{1,\ldots,S\} \tag{11}$$

$$a_{n,n,c} + a_{n,n,d} + a_{n,c,d} + a_{n+1,c,d} \leqslant 3, \quad \forall\, n \in \{1,\ldots,N-1\}, c \in \{n+1,\ldots,C\}, d \in \{n+1,\ldots,C\} \setminus \{c\}. \tag{12}$$

$$a_{n,c,d} \in \{0,1\}, \qquad \forall\, n \in \{1,\ldots,N\}, c \in \{n,\ldots,C+S\}, d \in \{n,\ldots,C\} \tag{13}$$

$$b_d \in \{0,1\}, \qquad \forall\, d \in \{N+1,\ldots,C\} \tag{14}$$

The objective function of CRP-I minimizes the total number of relocations. As suggested in Section 2, the objective function has two main components. The first term counts the number of relocations incurred for the first $N-1$ retrievals, *i.e.*, $r\left(A^{(1)}, A^{(N)}\right)$. When the target container is $n \in \{1,\ldots,N-1\}$, the seventh property of the binary encoding states how to compute the number of containers blocking the target container n, hence the number of relocations to retrieve n. The second term counts the number of blocking containers in the configuration after the first $N-1$ retrievals, *i.e.*, $b\left(A^{(N)}\right)$, as suggested by Lemma 1.

Constraint (1) initializes the variable a for $n = 1$ with the input binary encoding $A^{(1)}$. Constraints (2), (3), (4), (5) and (6) correspond to the second, third, fourth, fifth and sixth properties of the binary encoding. Constraint (7) enforces that each stack can be of height at most T, where the height of stack s is computed using the seventh property of the binary encoding. Constraint (8) enforces the relation between variables b and a, which is explained in the eighth property of the binary encoding. The four

next constraints enforce the rules to update the binary encoding between retrievals. Constraints (9) and (10) correspond to rule A, Constraint (11) corresponds to rule B and Constraint (12) corresponds to rule C. Finally, constraints (13) and (14) ensure that variables are binary variables.

We point out that variables of CRP-I only keep track of the configuration states through the binary encoding, and it could seem that no decisions are made. Actually, decisions are implicitly taken when a state of configuration is chosen after each retrieval and the cost of these decisions (*i.e.*, the number of relocations) can easily be induced. Note also that the feasible set of CRP-I includes all sequences of binary configurations to reach a configuration with S containers from $A^{(1)}$, hence a feasibility problem still provides a feasible sequence of relocations.

Finally, we believe that this modeling approach can be easily generalized to different objective functions. We provide three extensions in the Appendix.

3.1 Minor improvements

We suggest here some features that we use together with CRP-I when calling commercial solvers. First, since the optimal objective value is an integer, we can set *the absolute gap of the optimization solver to* $1-\epsilon$ *for* $\epsilon > 0$. Indeed, if we can guarantee that the best feasible solution found so far with objective z_{best} is such that $z_{best} - z_{opt} \leqslant 1-\epsilon$, then by integrality of z_{best} and z_{opt}, this ensures that $z_{best} = z_{opt}$. Second, (Caserta et al., 2012) suggest a Minmax heuristic which is commonly used as a good upper bound. *We include this solution as a first incumbent to the solver.*

The flexibility of CRP-I allows us to take into account some existing improvements suggested in the literature to enhance the efficiency of integer programs for the CRP. For instance, *we can simply adapt one of the pre-processing steps suggested by Zehendner et al. (2015).* For each container $c \in \{1,\dots,C\}$, one can easily compute the first target container $\pi_c \in \{1,\dots,c\}$ for which c is moved. Therefore, all containers below c, including c, stay in their initial position until container π_c is the target container. Thus the c^{th} column of the binary encoding does not change until π_c is the current target container, *i.e.*,

$$a_{n,d,c} = A^{(1)}_{d,c} \qquad \forall\, n \in \{1,\dots,\min\{\pi_c - 1, N\}\}, c \in \{n,\dots,C\}, d \in \{n,\dots,C+S\}.$$

Although we do not use it in our experiments, we mention here that we can also easily *adapt constraint* (B) *from Zehendner et al. (2015)* as

$$\sum_{d=n+1}^{C} a_{n,n,d} \leqslant UB_n, \qquad \forall\, n \in \{1,\dots,N-1\},$$

where the computation of UB_n is detailed in Zehendner et al. (2015).

4 Computational Experiments

As mentioned in Section 1, we test our binary model on the instances introduced by Caserta et al. (2009). These instances are defined by two values (T', S), where S is the number of stacks and T' the initial number of containers per stack (identical for all stacks). It is assumed that the maximum number of tiers T is taken to be $T = T' + 2$. Each class associated with a pair (T', S) contains 40 instances, each representing different initial configurations with $C = S \times T'$ containers.

Experiments are carried out on a Dell C6300 with an Intel E5-2690 v4 2.6 GHz processor, 4 CPUs, 8.00 GB of RAM and the programming language is Julia 0.5.0. We use Gurobi 7.0.1. to solve the binary programming model CRP-I enhanced by the gap (with $\epsilon = 0.01$), and the incumbent and pre-processing improvements described in the previous section. We set a time limit of *3600 seconds per instance* for the solver. We also define the memory limit to be when *the product of the number of variables with the*

number of constraints is greater than 8×10^{10}. This definition is different than in Zehendner et al. (2015), but it has the merit of not depending on the computer system, allowing for an easier comparison with future studies. All instances and scripts are available at `https://github.com/vgalle/binaryIP_CRP`. We present our results similarly to Zehendner et al. (2015) in order to ease the comparison with this state-of-the art integer programming solution. Finally, we also compare our results with the integer program and the branch and bound algorithm presented by Expósito-Izquierdo et al. (2015) and Eskandari and Azari (2015) on the first 5 instances of several classes, as these are the only instances for which they provide results in their papers.

4.1 Difficulty of instances

(T', S)	C	Avg. LB	Avg. UB	Avg. gap	Nb. trivial
$(3, 3)$	9	4.725	5.075	0.35	27
$(3, 4)$	12	5.85	6.30	0.45	26
$(3, 5)$	15	6.75	7.05	0.30	28
$(3, 6)$	18	8.275	8.45	0.175	36
$(3, 7)$	21	9.10	9.325	0.225	33
$(3, 8)$	24	10.45	10.725	0.275	32
$(4, 4)$	16	9.40	10.975	1.575	8
$(4, 5)$	20	12.15	13.55	1.40	11
$(4, 6)$	24	13.225	14.675	1.45	11
$(4, 7)$	28	15.20	16.90	1.70	9
$(5, 4)$	20	13.575	16.75	3.175	4
$(5, 5)$	25	17.00	21.225	4.225	1
$(5, 6)$	30	20.05	24.25	4.20	2
$(5, 7)$	35	22.425	26.325	3.90	5
$(5, 8)$	40	25.60	29.60	4.00	4
$(5, 9)$	45	28.525	32.35	3.825	3
$(5, 10)$	50	31.35	35.50	4.15	2
$(6, 6)$	36	26.95	35.90	8.95	0
$(6, 10)$	60	41.525	49.85	8.325	2
$(10, 6)$	60	56.60	101.25	44.65	0
$(10, 10)$	100	84.125	139.275	55.15	0

Table 1: Difficulty of instances of Caserta et al. (2009)

Table 1 shows several key values indicating the difficulty of the instances. It displays the problem class (T', S), the number of containers in each class C, the average lower bound from Zhu et al. (2012) (Avg. LB), the average upper bound from Caserta et al. (2012) (Avg. UB) and the average gap between these two values (Avg. gap). Most importantly, it gives how many instances can be considered as trivial (Nb. trivial), *i.e.*, when the upper bound is equal to the lower bound, hence the integer program is not required to solve these instances. We mention that Table 1 confirms exactly the results of Zehendner et al. (2015). We mainly provide this table for the sake of clarity associated with the next results.

4.2 Results of CRP-I and comparison with BRP-II-A

We provide the results of CRP-I aggregated by class of instances in Table 2. This table recalls the number of trivial instances per class (Nb. trivial) and displays the number of instances that were solved to optimality (Nb. solved non-trivial), the average and standard deviation of CPU times in seconds (Avg.

(T', S)	Nb. trivial	Nb. solved non-trivial	Avg. CPU time (sec.)	Std. CPU time (sec.)	Nb. time limit	Nb. memory limit
$(3,3)$	27	13 (13)	0.1	0.1	0 (0)	0 (0)
$(3,4)$	26	14 (14)	0.1	0.1	0 (0)	0 (0)
$(3,5)$	28	12 (12)	0.1	0.1	0 (0)	0 (0)
$(3,6)$	36	4 (4)	0.4	0.2	0 (0)	0 (0)
$(3,7)$	33	7 (7)	0.6	0.4	0 (0)	0 (0)
$(3,8)$	32	8 (8)	1.3	0.7	0 (0)	0 (0)
$(4,4)$	8	32 (32)	0.1	0.1	0 (0)	0 (0)
$(4,5)$	11	29 (29)	0.5	0.3	0 (0)	0 (0)
$(4,6)$	11	29 (29)	2.3	5.4	0 (0)	0 (0)
$(4,7)$	9	31 (31)	6.4	12	0 (0)	0 (0)
$(5,4)$	4	36 (36)	1.8	4.5	0 (0)	0 (0)
$(5,5)$	1	**39** (38)	41.9	177.6	0 (1)	0 (0)
$(5,6)$	2	**37** (31)	68.0	166.2	1 (6)	0 (0)
$(5,7)$	5	**31** (19)	170.9	367.4	4 (2)	0 (14)
$(5,8)$	4	**29** (5)	590.2	820.6	7 (0)	0 (31)
$(5,9)$	3	**23** (2)	658.2	799.1	14 (0)	0 (35)
$(5,10)$	2	**20** (0)	1111.0	1056.7	18 (0)	0 (38)
$(6,6)$	0	**19** (7)	441.7	723.8	21 (10)	0 (23)
$(6,10)$	2	0 (0)	n.a.	n.a.	0 (0)	38 (38)
$(10,6)$	0	0 (0)	n.a.	n.a.	0 (0)	40 (40)
$(10,10)$	0	0 (0)	n.a.	n.a.	0 (0)	40 (40)

Table 2: Computational results of CRP-I on non-trivial instances (in parenthesis the results from Zehendner et al. (2015)). In bold, the classes for which CRP-I solves more instances optimally than BRP-II-A.

CPU time and Std. CPU time). Finally, it differentiates instances not solved optimally either due to time or memory limit. We provide the results of Zehendner et al. (2015) to make a comparison only in terms of the number of instances solved or not solved optimally, as CPU times cannot be fairly compared since both experiments were not run with same computer power.

Our experiment shows that we can now solve almost all instances with sizes lower than $(5,7)$ within minutes on average. We can also solve a majority of instances for classes $(5,8$ to $10)$ and some in $(6,6)$. However, all the other classes reach the memory limit threshold. Our method clearly outperforms BRP-II-A, specially for the classes $(5,7$ to $10)$ and $(6,6)$.

Table 3 presents alternative indicators which only depend on the binary formulation properties and not on the computer features. Compared to those in Table 2, these indicators provide a more unbiased estimation of the performance of CRP-I. We provide the number of binary variables (Nb. binary variables), the average number of constraints (Avg. nb. constraints) over all 40 instances of each class. Note that the number of binary variables is constant in a given class of instances. In addition, we show the average number of node iterations done by the solver (Avg. node iterations) for instances solved optimally. We can observe that the number of variables is very reasonable for all classes (expect $(10,10)$) and that CRP-I decreases this number substantially compared to BRP-II-A (by a factor varying from 1.34 to 22), even with their proposed pre-processing step. This can partly explain the performances reported in Table 2. However, the bottleneck of our model seems to be the substantial growth of the number of constraints as instance size grows. Future work could be to study a row-generation approach for the CRP-I model.

4.3 Efficiency of the upper bound and hardness of the CRP

Table 4 provides the number of non-trivial instances solved optimally by CRP-I and for these instances,the average optimal number of relocations (Avg. Opt.), the average gap between the upper bound and the optimal solution (Avg. Gap UB-Opt) and the number of instances for which the upper bound is optimal (Nb. UB optimal). These key factors can help estimate the impact of using an upper bound as first incumbent. It can also help to get insight whether the hard part of the CRP is to find

(T', S)	Nb. binary variables	Avg. nb. constraints	Avg. node iterations
$(3, 3)$	408 (546.8)	2,326.4	0
$(3, 4)$	927 (2,140.1)	6,730.7	0
$(3, 5)$	1,764 (5,236.0)	15,389.8	0
$(3, 6)$	2,995 (16,170.8)	30,303.2	0
$(3, 7)$	4,696 (25,576.0)	54,364.3	0
$(3, 8)$	6,943 (37,248.0)	90,340.4	0
$(4, 4)$	2,005 (5,755.5)	16,383.5	0
$(4, 5)$	3,844 (19,380.2)	37,413.0	0
$(4, 6)$	6,560 (32,718.8)	74,130.1	3.5
$(4, 7)$	10,324 (72,055.6)	132,182.1	0
$(5, 4)$	3,675 (15,161.6)	32,361.9	22.8
$(5, 5)$	7,074 (42,530.7)	73,823.8	420.7
$(5, 6)$	12,105 (112,706.7)	145421.6	77.5
$(5, 7)$	19,088 (198,902.1)	260,202.2	25.6
$(5, 8)$	28,343 (352,122.7)	431,277.6	18.2
$(5, 9)$	40,190 (583,790.6)	675,140.5	10.7
$(5, 10)$	54,949 (884,834.6)	1,009,631.0	12.2
$(6, 6)$	20,062 (243,620.9)	253,368.7	466.4
$(6, 10)$	91,384 (2,058,673.2)	1,750,000.0	n.a.
$(10, 6)$	84,650 (-)	1,170,272.8	n.a.
$(10, 10)$	388,124 (-)	8,090,000.0	n.a.

Table 3: Efficiency indicators of CRP-I on non-trivial instances (in paranthesis the number of variables from Zehendner et al. (2015) with pre-processing).

an optimal solution or to prove optimality of a given optimal solution. We can see that for instances with $T' \leqslant 4$, a majority of instances are solved optimally by the upper bound and the average gap is always less than 1. Table 2 shows CRP-I solves all these instances optimally within a few seconds. This suggests that providing the upper bound as an incumbent is beneficial for the integer program and the proof of optimality of this incumbent seems to be relatively fast. For larger classes, there are many fewer instances for which the upper bound is optimal, and hence, the solution time for these instances increases dramatically. This suggests that the hard part for larger proglem is to be find an optimal solution. Once found, CRP-I seems to be quite efficient in proving the optimality of the solution.

4.4 Comparison of CRP-I with solutions from Expósito-Izquierdo et al. (2015) and Eskandari and Azari (2015)

We present Table 5 in a manner similar to Expósito-Izquierdo et al. (2015) and Eskandari and Azari (2015) in order to ease the comparison between our solution and theirs. First, the optimal values are all identical, confirming the accuracy of our model. Secondly, CRP-I clearly outperforms the corrected integer programs BRP-II* and BRP2ci in terms of computation time. In Table 6, we compare BRP2ci and CRP-I on three other problem classes provided by Eskandari and Azari (2015).

Most importantly, we compare CRP-I with the branch-and-bound approach. For all instances in classes other than $(4, 6)$, even though the branch-and-bound approach is faster, CRP-I still solves instances in less than a second, hence we claim that the difference between these two solution times is minor. However for the hardest problem class in this table, *i.e.*, $(4, 6)$, CRP-I solves all instances in less than 10 seconds, while

(T', S)	Nb. solved non-trivial	Avg. Opt.	Avg. Gap UB-Opt.	Nb. UB optimal
$(3, 3)$	13	6.77	0.23	10
$(3, 4)$	14	7.5	0.36	10
$(3, 5)$	12	8.75	0.08	11
$(3, 6)$	4	11.5	0.50	2
$(3, 7)$	7	11.71	0.29	5
$(3, 8)$	8	12.25	0.38	5
$(4, 4)$	32	10.69	0.97	12
$(4, 5)$	29	13.59	0.83	14
$(4, 6)$	29	14.55	0.90	15
$(4, 7)$	31	16.9	1.00	12
$(5, 4)$	36	15.61	1.47	9
$(5, 5)$	39	18.79	2.44	7
$(5, 6)$	37	22.24	2.22	5
$(5, 7)$	31	24.23	2.13	4
$(5, 8)$	29	27.48	1.86	9
$(5, 9)$	23	29.52	2.09	5
$(5, 10)$	20	32.00	1.8	5
$(6, 6)$	19	28.05	3.58	1

Table 4: Efficiency of the upper bound and hardness of the CRP on non-trivial instances

branch-and-bound requires more than 17 seconds to solve one instance. Expósito-Izquierdo et al. (2015) mention that the exact branch-and-bound approach could start to be intractable for this problem size and propose faster heuristic solutions by varying a parameter in their approach. However, we see in Table 2 that CRP-I scales well for larger problems, thus should be preferred when looking for optimal solutions in larger problems. Finally, we point out that this comparison can be generalized only to a certain extent since it is based on few instances for small problem classes (only these results were available in Expósito-Izquierdo et al. (2015) and Eskandari and Azari (2015)).

5 Conclusion

This paper deals with the restricted CRP (Assumption A_1). We present a new binary integer program (CRP-I) using an enhanced binary encoding of the problem. This formulation is novel, efficient for the CRP, and could easily be used to solve related problems. We show through a comparative computational study that this new model outperforms the state-of-the-art approaches for small and medium-sized instances of the CRP. However, the number of constraints still becomes too large to solve optimally for very large instances of the problem.

References

Akyüz, M. H. and Lee, C.-Y. (2013). A mathematical formulation and efficient heuristics for the dynamic container relocation problem. *Naval Research Logistics*, 61(2):101–118. `http://doi.org/10.1002/nav.21569`.

Borjian, S., Galle, V., Manshadi, V. H., Barnhart, C., and Jaillet, P. (2015). Container relocation problem: Approximation, asymptotic, and incomplete information. *CoRR*, abs/1505.04229. `http://arxiv.org/abs/1505.04229`.

(T', S)	Instance	Optimal	time (sec.)			
			CRP-I	BRP-II*	B&B	BRP2ci
$(3, 3)$	1	6	$*$	1.18	0.007	0.11
	2	5	$*$	1.39	0.007	0.11
	3	2	$*$	1.00	0.006	0.08
	4	4	$*$	1.09	0.007	0.09
	5	1	$*$	1.06	0.007	0.06
	Avg	3.6	$*$	3.6	0.007	0.09
$(3, 4)$	1	5	0.014	4.76	0.008	0.28
	2	3	$*$	18.39	0.007	0.22
	3	7	$*$	11.71	0.006	0.34
	4	5	$*$	16.06	0.007	0.26
	5	6	$*$	18.04	0.007	0.27
	Avg	5.2	0.014	13.79	0.008	0.274
$(3, 5)$	1	6	$*$	83.09	0.010	0.72
	2	7	$*$	75.95	0.007	1.2
	3	8	$*$	100.71	0.013	0.91
	4	6	$*$	95.31	0.008	0.8
	5	9	0.097	65.32	0.026	1.59
	Avg	7.2	0.097	84.11	0.013	1.044
$(3, 6)$	1	11	0.397	124.11	0.086	5.35
	2	7	$*$	113.29	0.008	1.59
	3	11	$*$	89.06	0.019	3.09
	4	7	$*$	93.12	0.016	1.78
	5	4	$*$	96.50	0.007	1.23
	Avg	8.0	0.397	103.22	0.027	2.608
$(3, 7)$	1	7	$*$	182.14	0.009	2.89
	2	10	$*$	284.29	0.011	3.29
	3	9	$*$	119.20	0.011	3.34
	4	8	$*$	472.32	0.010	3.35
	5	12	$*$	277.97	0.038	7.97
	Avg	9.2	$*$	267.26	0.016	4.168
$(3, 8)$	1	8	0.508	84.66	0.021	7.25
	2	10	$*$	14403.33	0.055	9.31
	3	9	$*$	8298.85	0.012	6.57
	4	10	$*$	250.33	0.013	11.03
	5	13	$*$	6384.65	0.022	11.06
	Avg	10.0	0.508	5884.36	0.025	9.044
$(4, 4)$	1	10	$*$	71.96	0.017	1.25
	2	10	0.036	228.91	0.025	0.95
	3	10	0.077	71.99	0.009	1.56
	4	7	$*$	65.25	0.008	0.78
	5	9	0.081	89.36	0.013	1.15
	Avg	9.2	0.065	105.49	0.014	1.138
$(4, 5)$	1	16	0.543	3544.86	0.540	61.62
	2	10	0.229	326.54	0.043	5.27
	3	13	0.261	1023.98	0.018	8.28
	4	8	$*$	119.86	0.014	2.96
	5	16	0.763	2656.67	0.579	81.26
	Avg	12.6	0.449	1534.38	0.239	31.878
$(4, 6)$	1	17	8.679	4077.08	17.476	n.a.
	2	8	0.541	18985.03	0.030	5.23
	3	13	$*$	1706.75	0.069	12.86
	4	14	0.828	2376.86	0.315	23.4
	5	15	0.641	8564.20	0.076	45.22
	Avg	13.4	2.672	7141.98	3.593	n.a.

Table 5: Comparison between our formulation CRP-I, BRP-II* and branch-and-bound (B&B) algorithm both proposed by Expósito-Izquierdo et al. (2015) and BRP2ci introduced by Eskandari and Azari (2015) on a subset of instances from Caserta et al. (2009). Time limit is one day for BRP-II* and 900 seconds for BRP2ci (instances not solved optimally are noted by n.a.). Times are given in seconds. $*$ indicates that the instance is trivial as defined in Section 4 and there is no need to run CRP-I

(T', S)	Instance	Optimal	time (sec.)	
			CRP-I	BRP2ci
$(4, 7)$	1	17	2.995	44.1
	2	18	1.245	20.47
	3	13	$*$	18.95
	4	16	2.31	49.78
	5	16	2.16	85.75
	Avg	16	2.178	43.81
$(5, 4)$	1	15	0.295	118.2
	2	19	1.06	185.36
	3	15	0.32	5.9
	4	12	0.17	5.01
	5	17	0.652	118.16
	Avg	15.6	0.499	86.526
$(5, 5)$	1	22	6.079	n.a.
	2	16	0.422	22.06
	3	22	25.542	n.a.
	4	21	5.538	n.a.
	5	16	0.837	46.3
	Avg	19.4	7.684	n.a.

Table 6: Further comparison between our formulation CRP-I and BRP2ci introduced by Eskandari and Azari (2015) on an extended subset of instances from Caserta et al. (2009). Time limit is 900 seconds for BRP2ci (instances not solved optimally are noted by n.a.). Times are given in seconds. $*$ indicates that the instance is trivial as defined in Section 4 and there is no need to run CRP-I

Borjian, S., Manshadi, V., Barnhart, C., and Jaillet, P. (2013). Dynamic stochastic optimization of relocations in container terminals. *working paper*. https://pdfs.semanticscholar.org/c53e/da06de3fda5597f7ee0ad82b830e4d1abb7c.pdf.

Carlo, H. J., Vis, I. F., and Roodbergen, K. J. (2014). Storage yard operations in container terminals: Literature overview, trends, and research directions. *European Journal of Operational Research*, 235(2):412–430. https://doi.org/10.1016/j.ejor.2013.10.054.

Caserta, M., Schwarze, S., and Voß, S. (2009). *A New Binary Description of the Blocks Relocation Problem and Benefits in a Look Ahead Heuristic*, pages 37–48. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-01009-5_4.

Caserta, M., Schwarze, S., and Voß, S. (2012). A mathematical formulation and complexity considerations for the blocks relocation problem. *European Journal of Operational Research*, 219(1):96–104. https://doi.org/10.1016/j.ejor.2011.12.039.

Eskandari, H. and Azari, E. (2015). Notes on mathematical formulation and complexity considerations for blocks relocation problem. *Scientia Iranica. Transaction E, Industrial Engineering*, 22(6):2722–2728. http://www.sid.ir/en/VEWSSID/J_pdf/955201506E10.pdf.

Expósito-Izquierdo, C., Melián-Batista, B., and Moreno-Vega, J. M. (2015). An exact approach for the blocks relocation problem. *Expert Systems with Applications*, 42(17-18):6408 – 6422. https://doi.org/10.1016/j.eswa.2015.04.021.

Galle, V., Borjian, S., Manshadi, V., Barnhart, C., and Jaillet, P. (2017). The stochastic container relocation problem. *submitted manuscript*. https://dspace.mit.edu/handle/1721.1/107762.

Lee, Y. and Lee, Y.-J. (2010). A heuristic for retrieving containers from a yard. *Computers & Operations Research*, 37(6):1139–1147. https://doi.org/10.1016/j.cor.2009.10.005.

Lehnfeld, J. and Knust, S. (2014). Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. *European Journal of Operational Research*, 239(2):297–312. https://doi.org/10.1016/j.ejor.2014.03.011.

López-Plata, I., Expósito-Izquierdo, C., Lalla-Ruiz, E., Melián-Batista, B., and Moreno-Vega, J. M. (2017). Minimizing the waiting times of block retrieval operations in stacking facilities. *Computers & Industrial Engineering*, 103(2):70–84. https://doi.org/10.1016/j.cie.2016.11.015.

Petering, M. E. and Hussein, M. I. (2013). A new mixed integer program and extended look-ahead heuristic algorithm for the block relocation problem. *European Journal of Operational Research*, 231(1):120–130. https://doi.org/10.1016/j.ejor.2013.05.037.

Tanaka, S. and Takii, K. (2016). A faster branch-and-bound algorithm for the block relocation problem. *IEEE Transactions on Automation Science and Engineering*, 13(1):181–190. https://doi.org/10.1109/TASE.2015.2434417.

Tang, L., Jiang, W., Liu, J., and Dong, Y. (2015). Research into container reshuffling and stacking problems in container terminal yards. *IIE Transactions*, 47(7):751–766. http://doi.org/10.1080/0740817X.2014.971201.

Tang, L., Zhao, R., and Liu, J. (2012). Models and algorithms for shuffling problems in steel plants. *Naval Research Logistics (NRL)*, 59(7):502–524. https://doi.org/10.1002/nav.21503.

Ünlüyurt, T. and Aydın, C. (2012). Improved rehandling strategies for the container retrieval process. *Journal of Advanced Transportation*, 46(4):378–393. http://doi.org/10.1002/atr.1193.

Wan, Y.-w., Liu, J., and Tsai, P.-C. (2009). The assignment of storage locations to containers for a container stack. *Naval Research Logistics (NRL)*, 56(8):699–713. http://doi.org/10.1002/nav.20373.

Zehendner, E., Casserta, M., Feillet, D., Schwarze, S., and 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.

Zhu, W., Qin, H., Lim, A., and Zhang, H. (2012). Iterative deepening A* algorithms for the container relocation problem. *IEEE Transactions on Automation Science and Engineering*, 9(4):710–722. https://doi.org/10.1109/TASE.2012.2198642.

A First extension: non uniform relocations

Suppose that moving some containers has a higher cost than other containers, *i.e.* let R_d be the cost of relocating container d. Then CRP-I can be used to solve this problem just by considering the same constraints and the modified objective function:

$$Min \left(\sum_{n=1}^{N-1} \sum_{d=n+1}^{C} R_d \times a_{n,n,d} + \sum_{d=N+1}^{C} R_d \times b_d \right).$$

B Second extension: minimizing crane travel time

In this version of the CRP, the goal is to minimize the travel time of the crane (still in the restricted setting). This extension requires that we are given:

- the initial position of the crane. We suppose here that the crane starts at the position where it delivers containers to trucks (also called the I/O-point). This can easily be extended to any starting position.
- $t_{s,r}^{(1)}$ (respectively, $t_s^{(2)}$), the time to relocate a container from stack s to stack r (respectively, the time to retrieve a container from stack s).

In order to extend CRP-I to this case (same as in Ünlüyurt and Aydın (2012)), we need to withdraw variable b and Constraint (8), take $N = C$ and introduce the binary variable:

$$\forall\, n \in \{1,\dots,N-1\}, c \in \{n+1, C\}, s \in \{1,\dots,S\}, r \in \{1,\dots,S\},$$
$$z_{n,c,s,r} = \begin{cases} 1 & \text{if container } c \text{ "goes" from stack } s \text{ to stack } r \text{ when } n \text{ is the target container,} \\ 0 & \text{otherwise.} \end{cases}$$

Note that $z_{n,c,s,s} = 1$ means that container c is not relocated when container n is the target container, hence for the sake of clarity, we define $t_{s,s}^{(1)} = 0$. CRP-I can be adapted for minimizing crane travel time as follows:

$$Min \left(\sum_{n=1}^{N-1} \sum_{c=n+1}^{C} \sum_{s=1}^{S} \sum_{r=1}^{S} 2 \times t_{s,r}^{(1)} z_{n,c,s,r} + \sum_{n=1}^{N} \sum_{s=1}^{S} 2 \times t_s^{(2)} a_{n,C+s,n} \right)$$

s.t.

$(1) - (7), (9) - (13)$

$$\sum_{r=1}^{S} z_{n,c,s,r} = a_{n,C+s,c}, \qquad \forall\, n \in \{1,\dots,N-1\}, c \in \{n+1,\dots,C\}, s \in \{1,\dots,S\} \tag{15}$$

$$\sum_{r=1}^{S} z_{n,c,r,s} = a_{n+1,C+s,c}, \qquad \forall\, n \in \{1,\dots,N-1\}, c \in \{n+1,\dots,C\}, s \in \{1,\dots,S\} \tag{16}$$

$$z_{n,c,s,r} \in \{0,1\} \qquad \forall\, n \in \{1,\dots,N-1\}, c \in \{n+1, C\}, s \in \{1,\dots,S\}, r \in \{1,\dots,S\} \tag{17}$$

The objective function is the total travel time of the crane, the first term accounts for relocations and the second term for retrievals. Moreover, we only have to add three types of constraints. Constraint (15) ensures that container c starts at stack s only if it is stack in s when n is the target container. Similarly, Constraint (16) enforces that container c ends at stack s only if it is in stack s when $n+1$ is the target container. Finally, variable z is binary given Constraint (17).

C Third extension: unrestricted CRP

Since the unrestricted CRP allows for relocations from a stack not containing the target container (see Caserta et al. (2012), Petering and Hussein (2013)), the unrestricted version of CRP-I uses an extra

type of variable defined as

$$\forall\, n \in \{1,\dots,N-1\}, c \in \{n+1, C\},$$
$$x_{n,c} = \begin{cases} 1 & \text{if container } c \text{ is relocated when } n \text{ is the target container,} \\ 0 & \text{otherwise.} \end{cases}$$

CRP-I can be adapted to the unrestricted case as follows:

$$Min \left(\sum_{n=1}^{N-1} \sum_{c=n+1}^{C} x_{n,c} + \sum_{d=N+1}^{C} b_d \right)$$

s.t.

$(1)-(8), (13)-(14)$

$$a_{n+1,d,c} \leqslant a_{n,d,c} + x_{n,c}, \qquad \forall\, n \in \{1,\dots,N-1\}, c \in \{n+1,\dots,C\}, d \in \{n+1,\dots,C+S\} \setminus \{c\} \tag{9'}$$

$$a_{n+1,d,c} \geqslant a_{n,d,c} - x_{n,c}, \qquad \forall\, n \in \{1,\dots,N-1\}, c \in \{n+1,\dots,C\}, d \in \{n+1,\dots,C+S\} \setminus \{c\} \tag{10'}$$

$$x_{n,c} + a_{n,C+s,c} + a_{n+1,C+s,c} \leqslant 2, \qquad \forall\, n \in \{1,\dots,N-1\}, c \in \{n+1,\dots,C\}, s \in \{1,\dots,S\} \tag{11'}$$

$$x_{n,c} + x_{n,d} + a_{n,c,d} + a_{n+1,c,d} \leqslant 3, \qquad \forall\, n \in \{1,\dots,N-1\}, c \in \{n+1,\dots,C\}, d \in \{n+1,\dots,C\} \setminus \{c\}. \tag{12'}$$

$$x_{n,c} \geqslant a_{n,n,c}, \qquad \forall\, n \in \{1,\dots,N-1\}, c \in \{n+1,\dots,C\} \tag{18}$$

$$a_{n,c,d} + a_{n,e,f} + a_{n+1,e,d} + a_{n+1,c,f} \leqslant 3 + a_{n,e,d}, \qquad \forall\, n \in \{1,\dots,N-1\}, c, e \in \{n+1,\dots,C+S\}, d, f \in \{n+1,\dots,C\}, c \neq d \neq e \neq f, \tag{19}$$

$$x_{n,c} \in \{0,1\} \qquad \forall\, n \in \{1,\dots,N-1\}, c \in \{n+1, C\} \tag{20}$$

First, $a_{n,n,c}$ has to be replaced by $x_{n,c}$. Moreover, we only have to add three types of constraints. Constraint (18) enforces that containers that are blocking the target container must be relocated. Constraint (19) ensures the LIFO policy holds for two different stacks. Consider two distinct stacks with at least one container in each (hence, two containers in each with the artificial container). Let c, d (respectively, e, f) be the containers in the first (respectively, second) stack with c (respectively, e) being below d

(respectively, f). If these containers are indeed in two different stacks (*i.e.* $a_{e,d} = 0$), then Constraint (19) enforces that both c below f and e below d are not compatible after n has been retrieved. Finally, Constraint (20) ensures that variable x is binary.