Integer programming models for the pre-marshalling problem

Consuelo Parreño-Torres*, Ramon Alvarez-Valdes

Department of Statistics and Operations Research, University of Valencia, Doctor Moliner 50, Burjassot, 46100, Valencia, Spain

Rubén Ruiz

Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edifico 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, 46021, València, Spain.

Abstract

The performance of shipping companies greatly depends on reduced berthing times. The trend towards bigger ships and shorter berthing times places severe stress on container terminals, which cannot simply increase the available cranes indefinitely. Therefore, the focus is on optimizing existing resources. An effective way of speeding up the loading/unloading operations of ships at the container terminal is to use the idle time before the arrival of a ship for sorting the stored containers in advance. The pre-marshalling problem consists in rearranging the containers placed in a bay in the order in which they will be required later, looking for a sequence with the minimum number of moves. With sorted bays, loading/unloading operations are significantly faster, as there is no longer a need to make unproductive moves in the bays once ships are berthed. In this paper, we address the pre-marshalling problem by developing and testing integer linear programming models. Two alternative families of models are proposed, as well as an iterative solution procedure that does not depend on a difficult to obtain upper bound. An extensive computational analysis has been carried out over several well-known datasets from the literature. This analysis has allowed us to test the performance of the models, and to conclude that the performance of the best proposed model is superior to that of previously published alternatives.

Keywords:

logistics, integer programming, optimization, pre-marshalling, storage area

1. Introduction

Nowadays, international maritime transport is considered to be the lifeblood of world trade, with over 80% of freight worldwide being carried by sea. In 2016, according to UNCTAD (2017), 16.7% of the total tonnage was transported on container ships, which carry all their load in standard-size intermodal containers. This enables goods to be moved from manufacturers to customers using different modes of transport (ships, railroads, trucks) with standardized handling equipments that makes re-handling unnecessary.

The growth in container shipping has put strong pressure on container terminals, whose main functions are the transshipment of containers from one transportation mode to another and their temporary storage. Due to the fierce competition in this sector, container terminals have to continuously improve their performance. Efficiency concerns are focused on minimizing the berthing time of ships and providing adequate infrastructure for all ship types and sizes. To avoid long berthing times, terminals can use the time before the arrival of a ship, when the workload at the terminal is at a minimum, to sort the containers located in the yard, so as to reduce later container retrieval times.

Storage yards occupy a large space at terminals, where inbound containers are stored before being transferred to the hinterland and outbound containers are also stored before being loaded onto ships. Basically, this area should ensure rapid loading and unloading of ships, and it has become the bottleneck of container port terminals. Hence, its management is very important for the terminals to achieve a competitive advantage. It is usually divided into several blocks, which are composed of parallel bays. A bay is formed by a set of stacks where containers are piled up vertically without exceeding a maximum height. This maximum height varies between terminals depending on the type of yard cranes used, but for safety reasons it rarely exceeds 4-5 tiers. Only in ports like Hong Kong, where space is at a premium, are stacks higher (up to 12 container tiers). As a consequence of stacking, to reach a specific container it may be necessary to rehandle containers on top of it. When

*Corresponding author

Email addresses: Consuelo.Parreno@uv.es (Consuelo Parreño-Torres*), Ramon.Alvarez@uv.es (Ramon Alvarez-Valdes), rruiz@eio.upv.es (Rubén Ruiz)

containers need to be relocated, it must also be decided to which stack each one is moved.

If the stowage planning of the ship is known beforehand, two types of activities in the container yard can be carried out before the ship arrives, re-marshalling and pre-marshalling. The aim in both cases is to relocate containers in order to reduce the number of relocations while the ship is being loaded, and therefore its berthing time. Their main feature is that no containers leave the yard, that is, containers are not allowed to be temporarily stored outside the yard. On the one hand, the re-marshalling problem consists of rearranging containers into bays within the same block. On the other, the pre-marshalling problem consists of rearranging containers within the same bay.

The subject of this study is the pre-marshalling problem. Given an initial bay configuration, the pre-marshalling problem looks for a working sequence that minimizes the number of moves required to obtain a final configuration in which containers will be located in the bay having no blocking containers. This can lead to drastic reductions in loading time which result in shorter berthing times, thus helping terminals to achieve higher efficiency.

In this paper, we focus on mathematical formulations for solving the pre-marshalling problem. Besides the conceptual advantage of providing a deeper understanding of the problem, integer models are useful for three main reasons: they are easy to implement by practitioners, they can be solved by commercial solvers, which have shown a dramatic increase in performance in the last decades, often by five or even more orders of magnitude (Bixby, 2002), and are constantly improving at a constant rate, and they are very flexible, allowing constraints to be added or removed to meet the requirements of the specific problem being solved. Other sophisticated exacts methods can be more efficient nowadays, but are difficult to implement, results are hard to reproduce without access to the source code, cannot take advantage of the advances obtained by commercial solvers, to the point that they might even be surpassed in performance by new solver versions, and are tailored to the problem, making it difficult to accommodate new constraints.

Our study of integer models for the pre-marshalling problem has two phases. In a first phase, we develop a series of models in which, besides the variables describing the configuration of the bay, the moves required to rearrange the bay are described by just one type of variables. The developed models are increasingly complex, improving the relationship of the variables and the linear relaxation at the expense of enlarging the set of variables. In a second phase, variables describing the moves are split into two types, one indicating the origin of the move and another indicating the destination of the move. This alternative strategy is shown to be much more efficient, in terms of the number of optimal solutions obtained and running times, in an extensive computational study on several datasets from the literature. The computational study also shows that the best of these new models outperforms the previously published models proposed by Lee and Hsu (2007), and the recent model presented by de Melo da Silva et al. (2018).

The remainder of the paper is as follows. Section 2 reviews the relevant literature for the pre-marshalling problem. A formal description of the problem is presented in Section 3. Sections 4 and 5 describe the developed mathematical models and Section 6 the iterative procedure used to solve them. Models are thoroughly computationally tested and compared with existing models in Section 7. Finally, Section 8 considers possible extensions of the models to cope with more realistic conditions and Section 9 highlights the conclusions and suggests future lines of research.

2. Previous work

Achieving a good yard plan has become a primary objective of many container port terminals. This is the main reason why academic work in this field has been intense in recent years. Lehnfeld and Knust (2014) identified various problems in storage areas arranged in stacks, including pre-marshalling, re-marshalling, and relocation problems. All of them are post-stacking problems that deal with minimizing the future loading time of the ship. While in the pre-marshalling and re-marshalling problems the number of containers remains constant during the reshuffling, in the relocation problem the number of containers decreases, i.e., containers are retrieved at the same time. In other words, the relocation problem deals with the management of the bay once the ship has berthed and while it is being loaded. The three problems are surveyed by Caserta et al. (2011a). The re-marshalling problem is also studied by Kang et al. (2005); Kang et al. (2006); Choe et al. (2011), and the relocation problem by Caserta et al. (2011b); Caserta et al. (2012); Petering and Hussein (2013).

Several papers related to pre-marshalling propose heuristic methods for solving the problem. Lee and Chao (2009) present a neighborhood search process based on a combination of improvement moves. An extension of this research is proposed by Huang and Lin (2012), who also include another version of the pre-marshalling problem in which at the end of the reshuffling each type of container should be located in pre-defined cells. Hence, the objective is to find the minimum sequence of moves for sorting the containers in a specific way. Caserta and Voß (2009) propose a metaheuristic approach using the paradigm of the corridor method developed by Sniedovich and Voß (2006). Bortfeldt and Forster (2012) develop a tree search procedure and an algorithm to obtain a lower bound for the problem. In addition to a multi-start technique for the problem using the heuristic called Lowest Priority First (LPFH), Expósito-Izquierdo et al. (2012) propose an instance generator able to construct instances with different difficulty levels. More recently, Jovanovic et al. (2017) have presented an extension of the LPFH whose main idea is to provide different heuristics for each stage of the initial algorithm and mix them in a multi-heuristic approach. Target-guided algorithms are proposed by Wang et al. (2015) and by Wang et al. (2017). Wang et al. (2015) also present an extension of the problem that considers a dummy stack next to the bay where containers can be temporarily stored during the reshuffling of the bay. Finally, a genetic approach is proposed by Gheith et al. (2016) and a biased random-key genetic method by Hottung and Tierney (2016).

A significantly smaller group of papers address the problem using exact methods. An A* algorithm is presented by Expósito-Izquierdo et al. (2012) in which a simple lower bound, counting the number of blocking containers, is used. Tierney et al. (2016) present an A* technique that includes branching rules, symmetry breaking, and the tighter lower bound proposed by Bortfeldt and Forster (2012). Zhang et al. (2015) propose a branch and bound whose branching procedure is guided by a heuristic algorithm and Tanaka and Tierney (2018) propose an improvement branch and bound algorithm that is also guided by a heuristic algorithm and a tighter lower bound together with dominance rules that allow a greater number of branches to be cut, obtaining better results. A different exact method is introduced by van Brink and van der Zwaan (2014) who develop a branch and price approach based on column generation.

Despite the recent increase in the number of contributions related to the pre-marshalling problem, to the best of our knowledge there are only two papers that provide a mathematical formulation. Lee and Hsu (2007) developed an integer programming model based on a multi-commodity flow formulation. In this model, time is discretized into time segments separated by time points. The slots in the bay are represented by nodes in the network and the possible moves for each container by arcs, with the moves of containers in each time segment being represented by flows in the network. They illustrate the behaviour of their model on one layout, including several extensions of the basic formulation. More recently, de Melo da Silva et al. (2018) propose a unified model for the pre-marshalling and the container relocation problem (CRP). They also discretize time into time segments separated by time points and consider two types of variables: variables describing the state of the bay and variables defining the movements. In this paper, we study the performance and limits of the mathematical formulations for the pre-marshalling problem through an exhaustive computational analysis of the models proposed by Lee and Hsu (2007), by de Melo da Silva et al. (2018), and of the several formulations we have developed.

3. Description of the problem

The pre-marshalling problem considered here consists in determining a minimal sequence of moves to transform the initial layout of a bay into a final layout without containers that block the removal of others. All containers are of the same size and the crane can only move one container at a time, without considering the distance between stacks.

The bay can be seen as an $H \times S$ matrix, where H represents the highest tier of stacks and S the number of stacks. Obviously, the bay can accommodate at most $H \times S$ containers. However, if the bay is at 100% occupancy, intra-bay moves of containers are not possible, so the occupancy level will always be lower than the total capacity. For each container c its position is described by the tuple (s, h) where $s \in \{1, \dots, S\}$ and $h \in \{1, \dots, H\}$ are respectively the stack and the tier in which container c is currently placed. Note that pair $(1, 1)$ refers to the container located on the bottom tier of the leftmost stack. A number is assigned to each container indicating the order in which it has to be retrieved in order to be loaded onto a ship (export containers) or a truck or train (import containers). In this way, each container c located in some position (s, h) is assigned a priority p_c such that $p_c \in \{1, \ldots, P\}$, 1 being the highest priority and P the lowest priority. It is possible that more than one container may share the same priority, that is, within a set of containers with the same priority the loading sequence is irrelevant. We denote n_p as the total number of containers of priority p in the bay.

Given an initial layout of the bay, if there is a container c in position (s, h) and there is a container c' in a higher tier $h' > h$ on the same stack s (container c' is located in position (s, h')), such that $p_{c'} > p_c$, this container and all the containers placed above it are referred to as blocking containers and they will need to be moved so as to achieve a correct final layout. Pre-marshalling looks for a final layout without any blocking containers.

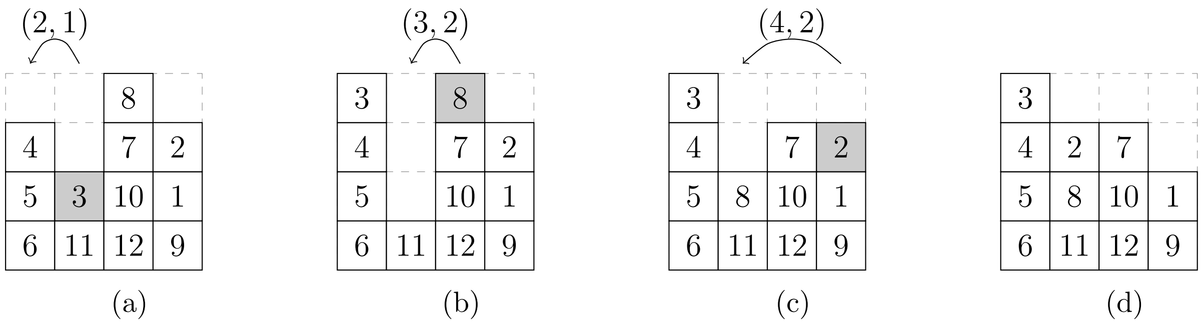


Figure 1: Example solution to the pre-marshalling problem.

A solution is a sequence of moves of the form $(s, k) : s \neq k \in \{1, \ldots, S\}$ where s is the origin stack to which the topmost container to be moved belongs and k is the destination stack where the container will also be placed at the top. An empty column will never be an origin and a full column will never be a destination. Figure 1a shows a bay with 4 tiers, 4 stacks, and 12 containers. Each container is represented by a box and the number in the box is its priority. This layout is not ordered in such a way that containers can be retrieved without any reshuffling. Containers with priorities 2 and 8 block containers with higher priorities. An optimal solution for this instance is the sequence of moves $(2,1)$, $(3,2)$, $(4,2)$. Figures 1b and 1c show the intermediate layouts associated with a solution that reaches the final layout of Figure 1d without any blocking container.

4. Integer programming models

In our models, time is discretized into time segments separated by T time points with the assumption that at most one move can be made in each time segment. Each time point corresponds to a layout of the bay. If the layout at time t is different from that at time $t+1$, a move has taken place in the time segment between t and $t+1$ with $t \in \{1,\dots,T\}$. The first time point corresponds to the initial layout of the bay, and at the last time point T the bay should be arranged, needing no further moves. The value of T has to be strictly greater than the optimal number of moves necessary to order the bay. Otherwise, the problem will be unfeasible. The way in which we set this value is discussed in Section 6.

In this section we describe a first set of integer models, based on the use of two types of variables. Variables x describe the layout of the bay at each period and variables y describe the move of one container from one stack to another, defining the transition from one time point to the next. Different ways of defining variables y define different models. According to the notation proposed in Section 3, the total number of stacks in the bay is represented by S and the slots in each stack will be numbered from 1 to H, H being the highest tier (equal for all the stacks in the bay). In the same way, priorities are numbered from 1 (the highest priority) to P (the lowest priority). We consider sets $\mathcal{S} = \{1,\dots,S\}$, $\mathcal{H} = \{1,\dots,H\}$, $\mathcal{P} = \{1,\dots,P\}$, and $\mathcal{T} = \{1,\dots,T\}$ to simplify notation.

4.1. An integer linear model with 3-indexed y variables (IP3)

The two types of binary variables involved in the first integer programming model (*IP3*) are:

$$x_{shpt} = \begin{cases} 1, & \text{If at time point } t \text{ there is a container in stack } s \text{, tier } h \text{, whose priority is } p \\ 0, & \text{Otherwise} \end{cases}$$
$$\forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T}$$

$$y_{skt} = \begin{cases} 1, & \text{If in the time segment between } t \text{ and } t+1 \text{ a move from stack } s \text{ to } k \text{ takes place} \\ 0, & \text{Otherwise} \end{cases}$$
$$\forall s \in \mathcal{S}; \quad \forall k \in \mathcal{S} \setminus \{s\}; \quad \forall t \in \mathcal{T} \setminus \{T\};$$

Variables y_{skt} represent the simplest way of describing a move between stacks at a given time segment.

The initial layout of the bay is given. Then, we initialize variables x_{shct} at time point $t = 1$ $\forall s, h, c$. Thus, they are parameters of the model. The pre-marshalling problem can be formulated in the *IP3* model as follows:

$$\textbf{Min} \quad \sum_{s=1}^{S} \sum_{\substack{k=1 \\ k \neq s}}^{S} \sum_{t=1}^{T-1} y_{skt} \tag{1}$$

$$\sum_{s=1}^{S} \sum_{h=1}^{H} x_{shpt} = n_p \qquad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T}; \tag{2}$$

$$\sum_{s=1}^{S} \sum_{\substack{k=1 \\ k \neq s}}^{S} y_{skt} \leq 1 \qquad \forall t \in \mathcal{T} \setminus \{T\} \tag{3}$$

$$\sum_{s=1}^{S} \sum_{\substack{k=1 \\ k \neq s}}^{S} y_{skt+1} \leq \sum_{s=1}^{S} \sum_{\substack{k=1 \\ k \neq s}}^{S} y_{skt} \qquad \forall t \in \mathcal{T} \setminus \{T-1, T\}; \tag{4}$$

$$\sum_{p=1}^{P} x_{s1pt} \leq 1 \qquad \forall s \in \mathcal{S}; \quad \forall t \in \mathcal{T} \setminus \{1\}; \tag{5}$$

$$\sum_{p=1}^{P} x_{sh+1pt} \leq \sum_{p=1}^{P} x_{shpt} \qquad \forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H} \setminus \{H\}; \quad \forall t \in \mathcal{T} \setminus \{1, T\};; \tag{6}$$

$$x_{shpt} + \sum_{k=1}^{P} x_{sh+1kt} \leq 1 + x_{shpt+1} \qquad \forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H} \setminus \{H\}; \tag{7}$$

$$x_{shpt} \leq x_{shpt+1} + \sum_{\substack{k=1 \\ k \neq s}}^{S} y_{skt} \qquad \forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T} \setminus \{T\}; \tag{8}$$

$$x_{shpt+1} \leq x_{shpt} + \sum_{\substack{k=1 \\ k \neq s}}^{S} y_{kst} \qquad \forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T} \setminus \{T\} \tag{9}$$

$$\sum_{k=p}^{P} x_{sh+1kT} \leq \sum_{k=p}^{P} x_{shkT} \qquad \forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H} \setminus \{H\}; \quad \forall p \in \mathcal{P}; \tag{10}$$

$$\sum_{\substack{s=1 \\ s \neq k}}^{S} y_{skt} + \sum_{\substack{s=1 \\ s \neq k}}^{S} y_{kst+1} \leq 1 \qquad \forall k \in \mathcal{S}; \quad \forall t \in \mathcal{T} \setminus \{T\} \tag{11}$$

$$x_{shpt} \in \{0,1\}, \quad y_{skt} \in \mathcal{R}^{+} \qquad \forall s,k \in \mathcal{S} : s \neq k; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T}; \tag{12}$$

The objective function (1) minimizes the total number of moves between stacks necessary to arrange the bay. The number of containers of each priority remains constant over time by constraints (2). Constraints (3) force each time segment to have at most one move. By (4), moves are assigned to the earliest possible times, so if there are no moves in the time segment between t and $t+1$, there will be no moves in any later time segment. The assumption that each slot can contain at most one container is forced by constraints (5) for the lowest tier. For the remaining tiers, this assumption is forced by constraints (6) and (7). Constraints (6) also ensure that if tier $h+1$ of a stack s is occupied, all the other slots in lower tiers of stack s are also occupied. In addition, constraints (7) ensure that containers that do not occupy the topmost position at a stack cannot be moved between two successive time points. In order to control topmost containers, we have constraints (8) and (9). On the one hand, if there was a container with priority p in slot (s,h) at time point t but it is not there at time point $t+1$, there has been a move whose origin was stack s in the time segment between t and $t+1$ (constraints (8)). On the other hand, if there is a container with priority p in slot (s,h) at time point $t+1$ but it was not there at time point t, there has been a move whose destination was stack s in the time segment between t and $t+1$ (constraints (9)). The bay is already arranged at time T by constraint (10), otherwise the problem is infeasible.

If in the time segment between t and $t+1$ there is a move from stack s to stack k, in the following time segment there should not be a move from stack k to another stack r because these moves are dominated by a single move from s to r. An example can be seen in Figure 2, in which the pair of moves $(3,4),(4,1)$ leads to the same layout as the single move $(3,1)$. Constraints (11) avoid these cases, which are known in the literature as transitive moves (Tierney et al. (2016)).

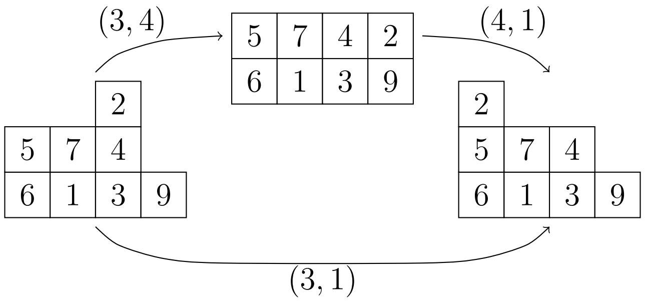


Figure 2: Transitive moves

Transitive moves that are not directly successive can be avoided by adding constraints (13). Nevertheless, these relations have more theoretical than practical interest because of the large number of constraints involved, which render them impractical and therefore are not included in the model.

$$y_{skt} + \sum_{i=1}^{t'-1} y_{uvt+i} + \sum_{\substack{r=1 \\ r \notin \{k,u,v\}}}^{S} y_{krt+t'} \leq t' \quad \forall s \in \mathcal{S}; \quad \forall k \in \mathcal{S} \setminus \{s\}; \quad \forall u \in \mathcal{S} \setminus \{s,k\}; \quad \forall v \in \mathcal{S} \setminus \{s,k,u\}; \quad \forall t \in \mathcal{T} \setminus \{T-1,T\}; \quad \forall t' \in \mathcal{T} \setminus \{1, T-t+1, \ldots, T\}; \tag{13}$$

Although in the definition of variables x_{shpt} and y_{skt} we have declared them as binary variables, only variables x_{shpt} have to be binary. If these variables, defining the configuration of the bay at each period, only take $0-1$ values, variables y_{skt} will also take $0-1$ values, even if they are declared as non-negative real variables. The proof is provided in Proposition 1.

Proposition 1. *The integrality of y_{skt} variables is inherited from the structure of IP3 model.*

Proof. If there is no move at time point t, variables $x_{shpt} = x_{shpt+1}$ for all s, h, p. In this case, y_{skt} variables will be zero for all s, k, as the objective function minimizes their sum and they are declared as positive real numbers.

Otherwise, the move at time point t will involve a container of a given priority p_1, initially placed at stack s_1 and tier h_1, being moved to stack s_2, tier h_2. Considering the origin of the move: $x_{s_1h_1p_1t} = 1$ and $x_{s_1h_1p_1t+1} = 0$. Since $y_{skt} \in \mathcal{R}^+$, by constraints (3) and (8) we obtain the following inequalities:

$$1 \overset{(8)}{\leq} \sum_{\substack{k=1\\k\neq s_1}}^{S} y_{s_1kt} \leq \sum_{\substack{k=1\\k\neq s_1}}^{S} y_{s_1kt} + \sum_{\substack{s=1\\s\neq s_1}}^{S}\sum_{\substack{k=1\\k\neq s}}^{S} y_{skt} = \sum_{s=1}^{S}\sum_{\substack{k=1\\k\neq s}}^{S} y_{skt} \overset{(3)}{\leq} 1 \Longrightarrow \sum_{\substack{s=1\\s\neq s_1}}^{S}\sum_{\substack{k=1\\k\neq s}}^{S} y_{skt} = 0 \tag{14}$$

Considering now the destination of the move: $x_{s_2h_2p_1t} = 0$ and $x_{s_2h_2p_1t+1} = 1$. By constraints (3) and (9):

$$1 \overset{(3)(9)}{=} \sum_{\substack{s=1\\s\neq s_2}}^{S} y_{ss_2t} = \sum_{\substack{s=1\\s\neq s_1\\s\neq s_2}}^{S} y_{ss_2t} + y_{s_1s_2t} \tag{15}$$

We can decompose the final equation of expression (14) as follows:

$$0 = \sum_{\substack{s=1\\s\neq s_1}}^{S}\sum_{\substack{k=1\\k\neq s}}^{S} y_{skt} = \sum_{\substack{s=1\\s\neq s_1}}^{S}\sum_{\substack{k=1\\k\neq s\\k\neq s_2}}^{S} y_{skt} + \sum_{\substack{s=1\\s\neq s_1\\s\neq s_2}}^{S} y_{ss_2t} \Longrightarrow \sum_{\substack{s=1\\s\neq s_1\\s\neq s_2}}^{S} y_{ss_2t} = 0 \Longrightarrow y_{s_1s_2t} = 1$$

If there is a move from stack s_1 to stack s_2 and variables x_{skpt} take binary variables, $y_{s_1s_2t} = 1$ and the remaining $y_{skt} = 0$. □

4.2. An integer programming model with 4-indexed y variables (IP4)

Variables y_{skt} of Model *IP3* do not control the priority of the container being moved at each time segment. Model *IP3* is correct and produces feasible solutions, but in its linear relaxation the relationship between variables x_{shpt} and y_{skt} is very weak, producing bad lower bounds and long solution times. One way of strengthening the $x - y$ relationship is to define y variables with a fourth index indicating the priority of the container being moved. A new model, *IP4*, can be defined using the variables:

$$x_{shpt} = \begin{cases} 1, & \text{if at time point } t \text{ there is a container in the stack } s\text{, tier } h\text{, whose priority is } p \\ 0, & \text{otherwise} \end{cases}$$
$$\forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T}$$

$$y_{skpt} = \begin{cases} 1, & \text{if in the segment between } t \text{ and } t+1\text{, a container with priority } p \\ & \text{is moved from stack } s \text{ to } k \\ 0, & \text{otherwise} \end{cases}$$
$$\forall s \in \mathcal{S}; \quad \forall k \in \mathcal{S} \setminus \{s\}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T} \setminus \{T\};$$

Variables y_{sk1T-1} turn into parameters because containers with priority 1 will not be involved in the last move, hence $y_{sk1T-1} = 0, \forall s, k$. The objective function and constraints of model *IP4* are very similar to that of model *IP3*, considering that each variable y_{skt} is now decomposed into P variables y_{skpt}. Constraints (8) and (9) are substituted by constraints:

$$x_{shpt} \leq x_{shpt+1} + \sum_{\substack{k=1\\k\neq s}}^{S} y_{skpt} \quad \forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T} \setminus \{T\}; \tag{16}$$

$$x_{shpt+1} \leq x_{shpt} + \sum_{\substack{k=1\\k\neq s}}^{S} y_{kspt} \quad \forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T} \setminus \{T\}; \tag{17}$$

in which variables x_{shpt} and y_{skpt} correspond now to containers with the same priority p, making their relationship tighter. Besides that, we can now avoid same priority symmetries by adding constraints (18).

$$\sum_{\substack{k=1\\k\neq s}}^{S} y_{skpt} + \sum_{\substack{k=1\\k\neq s}}^{S} y_{kspt+1} \leq 1 \quad \forall s \in \mathcal{S}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T} \setminus \{T-1, T\}; \tag{18}$$

Rules regarding same priority symmetries are introduced in Tanaka and Tierney (2018). If in the time segment between t and $t + 1$, a container of priority p is moved from stack s, in the following time segment there should not be a move of a container with priority p to that stack s. An example can be seen in Figure 3, in which the pair of moves $(1, 4), (2, 1)$ lead to the same layout as the single move $(2, 4)$.

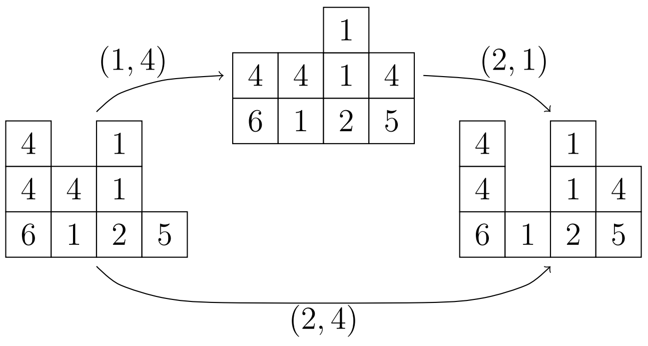


Figure 3: Same group symmetries

Same priority symmetries can be extended to cases in which the stacks are not used during an intermediate sequence by adding constraints (19). The interest of these constraints is just theoretical as in the general case of transitive moves.

$$y_{skpt} + \sum_{g=1}^{P} \sum_{i=1}^{t'-1} y_{uvgt+i} + \sum_{\substack{r=1 \\ r \notin \{s,u,v\}}}^{S} y_{rspt+t'} \leq t' \quad \forall s \in \mathcal{S}; \quad \forall k \in \mathcal{S} \setminus \{s\}; \quad \forall u \in \mathcal{S} \setminus \{s,k\}; \quad \forall v \in \mathcal{S} \setminus \{s,k,u\}; \quad \forall t \in \mathcal{T} \setminus \{T\}; \quad \forall t' \in \mathcal{T} \setminus \{1, T-t+1, \ldots, T\}; \quad \forall p \in \mathcal{P}; \tag{19}$$

4.3. An integer programming model with 5-indexed y variables (IP5)

The relationship between variables x_{shpt}, describing the bay configuration at time t, and variables y_{skt}, describing the moves in the *IP3* model, can be further strengthened if variables y_{skt} include not only the origin and destination stacks of the move, but also the origin and destination tiers. A new model, *IP5*, can be defined using the variables:

$$x_{shpt} = \begin{cases} 1, & \text{If at time point } t \text{ there is a container in the stack } s, \text{ tier } h, \text{ whose priority is } p; \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T};$$

$$y_{shket} = \begin{cases} 1, & \text{If in the time segment between } t \text{ and } t+1, \text{ a move from } (s,h) \text{ to } (k,e) \text{ takes place;} \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}; \quad \forall k \in \mathcal{S} \setminus \{s\}; \quad \forall h, e \in \mathcal{H}; \quad \forall t \in \mathcal{T} \setminus \{T\};$$

An optimal sequence will not include moves from the lowest tier in the last time segment, hence $y_{s1keT-1} = 0, \forall s, k, e$. Taking as a reference the initial model *IP3*, each variable y_{skt} is now decomposed into H^2 variables y_{shket}, where h is the tier in which the container is before the move and e the tier after the move.

Constraints (5) can be strengthened by adding the moves to the lowest tier (constraints 20). In this way, each slot can contain at most one container and if the slot is occupied in the time point t, it cannot receive a container.

$$\sum_{p=1}^{P} x_{s1pt} + \sum_{\substack{k=1 \\ k \neq s}}^{S} \sum_{e=1}^{H} y_{kes1t} \leq 1 \quad \forall s \in \mathcal{S}; \quad \forall t \in \mathcal{T} \setminus \{1\}; \tag{20}$$

Constraints (21) ensure the following three assumptions: each slot can contain at most one container in the tiers up to tier 1; if one slot is occupied, the slots below it should also be occupied; and only topmost containers can be moved at each time segment. Therefore, *IP5* includes constraints (21), and constraints (6) and (7) are no longer needed.

$$\sum_{p=1}^{P} x_{sh+1pt} + \sum_{\substack{k=1\\k\neq s}}^{S}\sum_{e=1}^{H} y_{shket} + \sum_{\substack{k=1\\k\neq s}}^{S}\sum_{e=1}^{H} y_{kesh+1pt} \leq \sum_{p=1}^{P} x_{shpt} \tag{21}$$

$$\forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H} \setminus \{H\}; \quad \forall t \in \mathcal{T} \setminus \{T\};$$

Considering the origin and destination tiers of each move, model *IP5* can be strengthened by including constraints (22) to make the relationship between x and y variables stronger. If two consecutive levels of a stack are occupied at a given time, no container can be moved to any of these levels and no container can be moved from a lower position of these levels in the next time segment.

$$\sum_{p=1}^{P} x_{shpt} + \sum_{p=1}^{P} x_{sh+1pt} + \sum_{\substack{k=1\\k\neq s}}^{S}\sum_{e=1}^{H} y_{shket} + \sum_{\substack{k=1\\k\neq s}}^{S}\sum_{e=1}^{H} y_{kesh+1pt} + \sum_{\substack{k=1\\k\neq s}}^{S}\sum_{e=1}^{H} y_{keshpt} \leq 2 \tag{22}$$

$$\forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H} \setminus \{H\}; \quad \forall t \in \mathcal{T} \setminus \{T\};$$

Constraints (2) and (10) involving only x_{shpt} do not change. In the objective function (1) and in all other constraints involving variables y_{shket}, two more sums, $\forall h, \forall e \in \mathcal{H}$, have to be added.

4.4. An integer programming model with 6-indexed y variables (IP6)

An alternative model, *IP6*, uses a new type of variables y describing completely the move of a container, including the period t, the origin and destination stacks, s, k, the origin and destination tiers, h, e, and the priority p. The variables are:

$$x_{shpt} = \begin{cases} 1, & \text{If at time point } t \text{ there is a container in the stack } s\text{, tier } h\text{, whose priority is } p; \\ 0, & \text{Otherwise;} \end{cases}$$

$$\forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T};$$

$$y_{shkept} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1\text{, a container with priority } p \\ & \text{is moved from } (s,h) \text{ to } (k,e); \\ 0, & \text{Otherwise;} \end{cases}$$

$$\forall s \in \mathcal{S}; \quad \forall k \in \mathcal{S} \setminus \{s\}; \quad \forall h, e \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T} \setminus \{T\};$$

We fix variables $y_{shke1T-1}$ and $y_{s1kepT-1}$ to zero because these moves are not possible in the last time segment. Taking as a reference model *IP3*, the two inequalities (8) and (9) controlling the topmost containers of each stack are substituted by the equality:

$$x_{shpt} + \sum_{\substack{k=1 \\ k \neq s}}^{S} \sum_{e=1}^{H} y_{keshpt} = x_{shpt+1} + \sum_{\substack{k=1 \\ k \neq s}}^{S} \sum_{e=1}^{H} y_{shkept} \quad \forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T} \setminus \{T\}; \tag{23}$$

Constraints (2), keeping fixed the number of containers of each priority at each stack, are no longer necessary. Moreover, constraints (18) avoiding same priority symmetries are also included in this formulation.

5. Alternative integer models

The models described in the previous section were progressively strengthening the relationship between bay configurations (variables x) and the moves producing these configurations (variables y) but at the cost of significantly increasing the number of variables. Therefore, the advantage of tighter relationships, tighter linear relaxations, and tighter lower bounds can be partially offset by the computational cost of solving much larger models. In this section we keep the structure of the models described above, but change the definition of the variables in such a way that tighter relationships between configurations and moves do not require large increases in the number of variables. Each model of the previous section will have its counterpart in this new approach.

5.1. An integer programming model with split 2-indexed variables (IPS3)

Variables x_{shpt}, describing the configuration of the bay at each period are maintained. However, former variables y_{skt} describing the move are split into two variables, one variable w_{st} indicating the stack origin of the move, and another variable z_{kt} indicating the destination. Instead of $S \times (S-1) \times T$ variables for the moves, the new $IPS3$ model has $2 \times S \times T$.

The definition of the variables is:

$$x_{shpt} = \begin{cases} 1, & \text{If at time point } t \text{ there is a container in the stack } s, \text{ tier } h, \text{ whose priority is } p; \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T};$$

$$w_{st} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1, \text{ a container is moved from stack } s; \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}; \quad \forall t \in \mathcal{T} \setminus \{T\};$$

$$z_{st} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1, \text{ a container is moved to stack } s; \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}; \quad \forall t \in \mathcal{T} \setminus \{T\};$$

Using these variables, the model $IPS3$ inherits the constraints of $IP3$, replacing $\sum_{\substack{k=1 \\ k\neq s}}^{S} y_{skt}$ by w_{st}, and $\sum_{\substack{k=1 \\ k\neq s}}^{S} y_{kst}$ by z_{st}. The objective function can be expressed in terms of either variables w_{st} or variables z_{st}, because at each time t the number of containers leaving their initial stacks equals the number of containers arriving to a new stack. By including constraints (24), a stack will not be origin and destination of a move at one time segment.

$$z_{st} + w_{st} \leq 1 \quad \forall s \in \mathcal{S}; \quad \forall t \in \mathcal{T} \setminus \{T\}; \tag{24}$$

5.2. An integer programming model using priorities to define split 3-indexed variables (IPS4)

The relationship between variables x_{shpt} and variables w_{st}, z_{st} is reinforced if these variables include the reference to the priority p of the container being moved. An integer model *IPS4*, mirroring the *IP4* model but splitting variables y_{skpt} into w_{spt} and z_{spt} can be defined using variables:

$$x_{shpt} = \begin{cases} 1, & \text{If at time point } t \text{ there is a container in the stack } s, \text{ tier } h, \text{ whose priority is } p; \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T};$$

$$w_{spt} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1, \text{ a container with priority } p \\ & \text{is moved from stack } s; \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T} \setminus \{T\};$$

$$z_{spt} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1 \text{, a container with priority } p \\ & \text{is moved to stack } s; \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T} \setminus \{T\};$$

The model *IPS4* is very similar to *IP4*, considering that expressions $\sum_{\substack{k=1\\k\neq s}}^{S} y_{skpt}$ are now substituted by w_{spt}, and $\sum_{\substack{k=1\\k\neq s}}^{S} y_{kspt}$ by z_{spt}. We also strengthen the formulation by adding constraints (25).

$$\sum_{p=1}^{P} z_{spt} + \sum_{p=1}^{P} w_{spt} \leq 1 \quad \forall s \in \mathcal{S}; \quad \forall t \in \mathcal{T} \setminus \{T\}; \tag{25}$$

The *IP4* model had $S \times (S-1) \times P \times T$ variables for the moves, while the new *IPS4* model only needs $2 \times S \times P \times T$

5.3. An integer programming model using tiers to define split 3-indexed variables (IPS5)

The *IP5* model using 5-index variables y_{shket} can also be substituted by a model using split 3-index variables w_{sht} and z_{ket}. The variables of *IPS5* are:

$$x_{shpt} = \begin{cases} 1, & \text{If at time point } t \text{ there is a container in the stack } s\text{, tier } h\text{, whose priority is } p; \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T};$$

$$w_{sht} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1 \text{, a container is moved from } (s,h); \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall t \in \mathcal{T} \setminus \{T\};$$

$$z_{sht} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1 \text{, a container is moved to } (s,h); \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall t \in \mathcal{T} \setminus \{T\};$$

Taking as a reference the *IP5* model, expressions $\sum_{\substack{k=1\\k\neq s}}^{S} \sum_{e=1}^{H} y_{shket}$ are now simplified to w_{sht}, and $\sum_{\substack{k=1\\k\neq s}}^{S} \sum_{e=1}^{H} y_{kesht}$ to z_{sht}. Constraints avoiding a stack to be origin and destination at the same time segment are no necessary because it is already considered by the equivalent of constraints (22) in *IPS5*.

5.4. An integer programming model using tiers and priorities to define split 4-indexed variables (IPS6)

Similarly, variables y_{shkept} of the *IP6* model can be decomposed into split 4-indexed variables w_{shpt} and z_{kept}, defining a new model, *IPS6.* Its variables are:

$$x_{shpt} = \begin{cases} 1, & \text{If at time point } t \text{ there is a container in the stack } s, \text{ tier } h, \text{ whose priority is } p; \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T};$$

$$w_{shpt} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1, \text{ a container with priority } p \\ & \text{is moved from } (s,h); \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T} \setminus \{T\};$$

$$z_{shpt} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1, \text{ a container with priority } p \\ & \text{is moved to } (s,h); \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T} \setminus \{T\};$$

This approach cuts the number of variables down from $S(S-1)H^2PT$ to $2SHPT$. For completeness, we present the whole *IPS6* model which is our best proposal for solving the pre-marshalling problem.

$$\textbf{Min} \quad \sum_{s=1}^{S}\sum_{h=1}^{H}\sum_{p=1}^{P}\sum_{t=1}^{T-1} z_{shpt} \tag{26}$$

$$\sum_{s=1}^{S}\sum_{h=1}^{H} w_{shpt} = \sum_{s=1}^{S}\sum_{h=1}^{H} z_{shpt} \qquad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T} \setminus \{T\}; \tag{27}$$

$$\sum_{s=1}^{S}\sum_{h=1}^{H}\sum_{p=1}^{P} w_{shpt} \leq 1; \quad \sum_{s=1}^{S}\sum_{h=1}^{H}\sum_{p=1}^{P} z_{shpt} \leq 1 \qquad \forall t \in \mathcal{T} \setminus \{T\}; \tag{28}$$

$$\sum_{s=1}^{S}\sum_{h=1}^{H}\sum_{p=1}^{P} w_{shpt+1} \leq \sum_{s=1}^{S}\sum_{h=1}^{H}\sum_{p=1}^{P} w_{shpt} \qquad \forall t \in \mathcal{T} \setminus \{T-1, T\}; \tag{29}$$

$$\sum_{p=1}^{P} x_{s1pt} + \sum_{p=1}^{P} z_{s1pt} \leq 1 \qquad \forall s \in \mathcal{S}; \quad \forall t \in \{2, \ldots, T\}; \tag{30}$$

$$\sum_{k=p}^{P} x_{sh+1kT} \leq \sum_{k=p}^{P} x_{shkT} \qquad \forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H} \setminus \{H\}; \quad \forall p \in \mathcal{P}; \tag{31}$$

$$x_{shpt} + z_{shpt} = x_{shpt+1} + w_{shpt} \qquad \forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T} \setminus \{T\}; \tag{32}$$

$$\sum_{p=1}^{P} x_{sh+1pt} + \sum_{p=1}^{P} w_{shpt} + \sum_{p=1}^{P} z_{sh+1pt} \leq \sum_{p=1}^{P} x_{shpt} \qquad \forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H} \setminus \{H\}; \quad \forall t \in \mathcal{T} \setminus \{T\}; \tag{33}$$

$$\sum_{p=1}^{P} x_{shpt} + \sum_{p=1}^{P} x_{sh+1pt} + \sum_{p=1}^{P} w_{shpt} + \sum_{p=1}^{P} z_{sh+1pt} + \sum_{p=1}^{P} z_{shpt} \leq 2 \qquad \forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H} \setminus \{H\}; \quad \forall t \in \mathcal{T} \setminus \{T\}; \tag{34}$$

$$\sum_{h=1}^{H}\sum_{p=1}^{P} z_{shpt} + \sum_{h=1}^{H}\sum_{p=1}^{P} w_{shpt+1} \leq 1 \qquad \forall s \in \mathcal{S}; \quad \forall t \in \mathcal{T} \setminus \{T-1, T\}; \tag{35}$$

$$\sum_{h=1}^{H} w_{shpt} + \sum_{h=1}^{H} z_{shpt+1} \leq 1 \qquad \forall s \in \mathcal{S}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T} \setminus \{T-1, T\}; \tag{36}$$

$$w_{sh1(T-1)} = 0; \quad z_{sh1(T-1)} = 0; \qquad \forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \tag{37}$$

$$w_{s1p(T-1)} = 0; \qquad \forall s \in \mathcal{S}; \quad \forall p \in \mathcal{P} \setminus \{1\}; \tag{38}$$

$$x_{shpt} \in \{0,1\}, \quad w_{shpt}, z_{shpt} \in \mathcal{R}^{+} \qquad \forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T}; \tag{39}$$

Taking again as a reference the *IP6* model, the main difference is that constraints (2) are substituted by constraints (27). It allows a tighter relation between w and z variables.

6. An iterative procedure for solving the integer models

All models described in the previous sections require to determine a value of T, the maximum number of moves needed to rearrange the bay, plus one, because at the last period the bay must be completely ordered. While the lower bound proposed by Tanaka and Tierney (2018) is available and can be used in the models, the use of an upper bound provided by any of the heuristics developed so far is not clear. On the one hand, the number of variables and constraints, and therefore the performance of the models, depend directly on the value of T. Using the upper bound provided by a heuristic will make the performance of the model dependent on the performance of the heuristic. On the other hand, there is the unsolved question of deciding whether a pre-marshalling instance has a feasible solution or not. It is a difficult question, because it depends not only on the dimensions of the bay and the number of containers, but also on the positions of the containers. The vast majority of heuristic algorithms avoid the question by adding extra tiers to the instance, sometimes at the beginning of the procedure, sometimes when they are needed to allow further movements to be made. However, in a real situation, the maximum tier H of the bay is imposed by the height of the crane, and adding additional levels to the bay may not be possible. Moreover, a value obtained by modifying the original instance cannot be considered a valid upper bound. Heuristic algorithms that do not increase the bay size, as that used by de Melo da Silva et al. (2018), do not guarantee that a feasible solution will be found in all kind of instances. In de Melo da Silva et al. (2018), they test their PMP$_{m1}$ model on the CV dataset from Caserta and Voß (2009). CV instances are filled up to the same height and have two empty tiers on top of each stack. These two empty tiers make the heuristic algorithm proposed by de Melo da Silva et al. (2018) work. Nonetheless, it fails on instances with a different structure, for example, instances with a fill percentage 75% on the EMM dataset from Expósito-Izquierdo et al. (2012), going into a loop and not providing a value for T.

Therefore, we have developed an iterative procedure that does not require the use of an upper bound. Initially, we set $T = LB + 1$. If the model gives a feasible solution using this value of T, it is optimal. Otherwise, we set $T = T + 1$ and solve the model again. The procedure is repeated for increasing values of T until a feasible solution is obtained, producing an optimal solution. When we solve the models using this iterative procedure, constraints that assign the moves to the earliest possible times are no longer necessary. Although Proposition 1 proves that variables describe the movements can be defined as real variables, we keep them as binary variables because that helps to identify unfeasible instances faster.

7. Computational experiments

To test the performance of the mathematical models, exact solutions were obtained using CPLEX version 12.7 considering 2 threads and a time limit of 3600 seconds. We conducted all computational experiments on virtual machines with 4 virtual processors and 8 GBytes of RAM running Windows 10 Enterprise 64 bits. Virtual machines are run in an OpenStack virtualization platform supported by 12 blades, each with four 12-core AMD Opteron Abu Dhabi 6344 processors running at 2.6 GHz and 256 GBytes of RAM, for a total of 576 cores and 3 TBytes of RAM.

7.1. Test instances

We have used instances from four datasets of the literature:

EMM dataset. The original instances from Expósito-Izquierdo, Melián-Batista, and Moreno-Vega (2012) were lost, but they were generated again by Tierney, Pacino, and Voß (2016), using the same distributions. There are several categories of instances, with 4 tiers; 4, 7 or 10 stacks; and container fill percentages of 0.50, 0.75 and 1. In this study we use the instances with fill percentages 0.50 and 0.75, without adding any additional tier. Instances with fill percentage 1 cannot be solved unless some tiers are added on top of each stack, thus changing the fill percentage, and have not been considered here.

ZJY dataset. Zhang, Jiang, and Yun (2015) generated 100 instances, divided into 5 categories of 20 instances each, containing 6, 7, 8 or 9 stacks and 4 tiers, and 6 stacks and 5 tiers. Although generation details are not provided, it can be observed that there may be several containers with the same priority, and the stacks are usually filled up to $|H| - 1$ tiers.

CV dataset. In this dataset from Caserta and Voß (2009), all containers have different priorities, all stacks are filled to the same height, and there are two empty tiers on top of each stack. The instances used in this study range in size from 3 stacks and 5 tiers up to 7 stacks and 6 tiers, totalling 400 instances.

BZ dataset. These instances from van Brink and van der Zwaan (2014) range in size from 3 stacks and 4 tiers up to 9 stacks and 6 tiers, filled 50% or 70% with containers with 2, 3, or 6 different priorities, with a total of 960 instances.

7.2. Performance of the proposed IP models

Results on the EMM dataset. Table 1 shows the number of optimal solutions obtained by the models described in Sections 4 and 5 on the 450 instances of the *EMM* dataset. Concerning the initial models, their performance increases from *IP3* to *IP6.* Looking at the models using split variables, their performance also increases from *IPS3* to *IPS6.* The *IPS6* model is clearly better with a moderate increase in the number of variables. All the models solve well the instances with a low fill percentage, but there are significant differences between them when solving larger ones. The first row in Table 2 completes the information by showing the average running time in seconds computed on the set of instances optimally solved by all the models on the EMM dataset. *IPS6* is faster that the other split-variable models and they are in turn faster than the corresponding *IP3*-*IP6* models.

Table 1: Number of optimal solutions obtained by the proposed models on the EMM dataset from Expósito-Izquierdo et al. (2012) grouped by number of stacks (S) and fill percentage (f).

S	f	#	IP3	IP4	IP5	IP6	IPS3	IPS4	IPS5	IPS6
4	0.50	75	75	75	75	75	75	75	75	75
4	0.75	75	50	50	50	75	50	75	75	75
7	0.50	75	75	75	75	75	75	75	75	75
7	0.75	75	31	36	40	44	31	40	42	51
10	0.50	75	72	75	75	75	75	75	75	75
10	0.75	75	11	17	18	24	12	24	23	40
	Total	450	314	328	333	368	318	364	365	391

Table 2: Average running time in seconds only on the instances (O) of the studied dataset optimally solved by the eight proposed models.

Dataset	#	O	IP3	IP4	IP5	IP6	IPS3	IPS4	IPS5	IPS6
EMM	450	312	94.58	20.38	21.52	9.29	52.78	7.82	6.51	1.10
ZJY	100	15	575.78	95.32	75.17	39.89	243.68	41.29	28.04	8.10
CV	400	64	496.42	214.92	118.51	46.08	138.98	54.18	41.68	17.00
BZ	960	745	106.42	41.84	27.96	16.81	65.96	15.48	10.86	1.92
		Average	131.34	46.40	31.92	16.70	68.80	15.90	11.63	2.63

Results on the ZJY dataset. Table 3 contains the number of optimal solutions obtained by the eight models on the 100 instances of the ZJY dataset. The table exhibits two main characteristics. On the one hand, instances with 5 tiers are harder for all the models. On the other hand, the performance of the models in the other four categories follows the same pattern of the previous subsection. Again split-variable *IPS3-IPS6* models are increasingly better, outperforming *IP3-IP6* models. Concerning running times, included in the second row of Table 2, the differences between models are even larger than in the previous dataset. Models *IPS3-IPS6* are much faster than the other models, with *IPS6* being again the fastest alternative. *IPS6* cuts the CPU time down on average by 99% and 80% with respect to *IP3* and *IP6*, respectively.

Table 3: Number of optimal solutions obtained by the proposed models on the ZJY dataset from Zhang et al. (2015), grouped by number of tiers (H), and number of stacks (S).

H	S	#	IP3	IP4	IP5	IP6	IPS3	IPS4	IPS5	IPS6
4	6	20	6	9	11	16	9	12	12	20
4	7	20	2	9	12	14	4	13	14	17
4	8	20	4	8	10	13	7	12	12	18
4	9	20	3	3	6	11	3	6	7	17
5	6	20	0	1	1	3	0	1	2	9
	Total	100	15	30	40	57	23	44	47	81

Results on the CV dataset. Table 4 shows the number of optimal solutions obtained by the models on the CV instances. These instances are considered particularly difficult, because containers must be handled several times (Tanaka and Tierney (2018)). They are really difficult for the integer linear models tested, especially with a higher number of tiers. It is worth noting that, unlike previous tests, *IPS4* outperforms *IPS5*. Apart from that, a clear trend among *IP3-IP6* and *IPS3-IPS6* models is again observed looking at average running times on the third row of Table 2.

Results on the BZ dataset. Table 5 shows the number of optimal solutions obtained by the eight models tested on the BZ instances. As in previous tests, instances with fewer tiers are easier to solve even when the number of stacks increases. Comparing the models, we observe that *IPS4-IPS6*, the split-variable models from *IP4-IP6*, obtain the largest number

,

Table 4: Number of optimal solutions obtained by the models on the CV dataset from Caserta and Voß (2009) grouped by number of stacks (S) and number of tiers (H).

H	S	#	IP3	IP4	IP5	IP6	IPS3	IPS4	IPS5	IPS6
5	3	40	33	34	38	40	35	39	39	40
5	4	40	20	29	29	36	23	34	33	38
5	5	40	7	15	18	25	12	26	24	32
5	6	40	3	5	6	12	3	12	9	23
5	7	40	0	2	1	5	1	2	3	11
5	8	40	0	1	1	2	0	1	1	7
6	4	40	1	2	3	5	2	5	5	9
6	5	40	0	0	0	1	0	1	0	2
6	6	40	0	0	0	0	0	0	0	1
6	7	40	0	0	0	0	0	0	0	0
	Total	400	64	88	96	126	76	120	114	163

of optimal solutions. Nevertheless, *IP3* outperforms *IPS3*. For the subset of instances with 6 tiers, the number of optimal solutions ranges from 320-352 for *IP4-IP6*, and 337-413 for *IPS4-IPS6*. Table 2 shows the average running times in seconds for the 745 instances solved by all the models. The *IPS3-IPS6* models are the fastest and *IPS6* is the fastest of them all.

Table 5: Number of optimal solutions obtained by the proposed models on the BZ dataset from van Brink and van der Zwaan (2014), grouped by number of stacks (S), number of tiers (H), and fill percentage (f).

H	S	f	#	IP3	IP4	IP5	IP6	IPS3	IPS4	IPS5	IPS6
4	3	0.50	60	60	60	60	60	60	60	60	60
4	3	0.70	60	60	60	60	60	60	60	60	60
4	4	0.50	60	60	60	60	60	60	60	60	60
4	4	0.70	60	60	60	60	60	60	60	60	60
4	7	0.50	60	60	60	60	60	60	60	60	60
4	7	0.70	60	55	59	60	60	59	60	60	60
4	9	0.50	60	60	60	60	60	60	60	60	60
4	9	0.70	60	44	49	54	56	47	59	57	60
6	3	0.50	60	60	60	60	60	60	60	60	60
6	3	0.70	60	47	48	54	56	46	52	55	60
6	4	0.50	60	55	58	58	60	53	59	60	60
6	4	0.70	60	25	27	32	35	24	29	35	44
6	7	0.50	60	49	51	52	53	49	54	55	59
6	7	0.70	60	18	21	24	25	16	21	27	37
6	9	0.50	60	40	47	46	48	37	51	52	60
6	9	0.70	60	10	8	15	15	5	11	20	33
		Total	960	763	788	815	828	756	816	841	893

In summary, splitting the variables that describe the moves appears to be a very useful strategy. The relationship between variables describing the configurations and those describing the moves can be made tighter with a moderate increase in the number of variables, as can be seen in Table 6.

Table 6: Number of variables describing the moves involved in the eight new models.

Model	Variables	Model	Variables
IP3	$S(S-1)T$	IPS3	$2ST$
IP4	$S(S-1)PT$	IPS4	$2SPT$
IP5	$S(S-1)H^2T$	IPS5	$2SHT$
IP6	$S(S-1)H^2PT$	IPS6	$2SHPT$

7.3. Comparison with the state-of-the-art

There are several exact methods dealing with the pre-marshalling problem, being the algorithm *TT* recently proposed by Tanaka and Tierney (2018) the most competitive. It solves all the instances tested here in seconds, well below the time limit of one hour per instance. Nonetheless, as discussed in Section 1, mathematical models are more flexible and easier to implement which motivates their use as an alternative to sophisticated exact methods. In this section, we compare the performance of the proposed model with the state-of-the-art mathematical models proposed by Lee and Hsu (2007) and more recently by de Melo da Silva et al. (2018). We re-implemented the models proposed by Lee and Hsu (2007) and by de Melo da Silva et al. (2018) and they have also been tested using the iterative procedure presented in Section 6.

EMM dataset. The performance of the models proposed by Lee and Hsu (2007) and de Melo da Silva et al. (2018) and of *IPS6* model on the EMM dataset is shown in Table 7. Our model outperforms state-of-the-art models by solving 33% and 2% more instances to optimality than Lee and Hsu (2007) model and PMP$_{m1}$, respectively. Moreover, *IPS6* is much faster than the other models.

ZJY dataset. On the ZJY dataset, while the model proposed by Lee and Hsu (2007) solves 11 instances to optimality, PMP$_{m1}$ solves 79 and *IPS6* solves 81, as shown in Table 8. The main difference between PMP$_{m1}$ and *IPS6* lies again on the average running times, existing a clear superiority of model *IPS6*.

CV dataset. *IP6S* increases the number of optimal solution on average by 176% with respect to the model proposed by Lee and Hsu (2007) and by 5% with respect to PMP$_{m1}$, as shown in Table 9. Our model finds more optimal solutions on the categories: 5-5, 5-7, 5-8, and 6-4.

Table 7: Performance of the model proposed by Lee and Hsu, the PMP$_{m1}$ model from de Melo da Silva et al. (2018), and *IPS6* on the EMM dataset from Expósito-Izquierdo et al. (2012) grouped by number of stacks (S) and fill percentage (f). CPU(s) represents the average running time (in seconds) on the instances optimally solved by each model (Opt).

			Lee and Hsu		PMP$_{m1}$		*IPS6*	
S	f	#	Opt.	CPU(s)	Opt.	CPU(s)	Opt.	CPU(s)
4	0.50	75	75	0.03	75	0.01	75	0.01
4	0.75	75	50	15.40	75	47.15	75	27.15
7	0.50	75	74	65.90	75	0.78	75	0.60
7	0.75	75	26	693.73	49	190.05	51	99.85
10	0.50	75	63	241.20	75	3.44	75	1.87
10	0.75	75	7	696.09	35	649.84	40	230.91
	Total	450	295	148.32	384	93.52	391	42.33

Table 8: Performance of the model proposed by Lee and Hsu, the PMP$_{m1}$ model from de Melo da Silva et al. (2018), and *IPS6* on the ZJY dataset from Zhang et al. (2015), grouped by number of stacks (S) and number of tiers (H). CPU(s) represents the average running time (in seconds) on the instances optimally solved by each model (Opt).

			Lee and Hsu		PMP$_{m1}$		*IPS6*	
H	S	#	Opt.	CPU(s)	Opt.	CPU(s)	Opt.	CPU(s)
4	6	20	4	777.17	20	692.52	20	323.69
4	7	20	1	3256.73	17	273.00	17	201.91
4	8	20	4	496.86	17	380.08	18	341.26
4	9	20	2	1097.42	17	920.02	17	686.80
5	6	20	0	-	8	1011.21	9	694.39
	Total	100	11	958.88	79	616.24	81	419.43

BZ dataset. Table 10 shows the performance of the models on BZ dataset. The Lee and Hsu model solves 717, PMP$_{m1}$ solves 888, and *IP6S* solves 893 out of the 960 BZ instances. We completely solve two more categories than de Melo da Silva et al. (2018).

We now refer to Figure 4 that shows the running times in seconds for finding an optimal solution required by the model PMP$_{m1}$ de Melo da Silva et al. (2018) and by the proposed model *IPS6* on all datasets. Cases in which an optimal solution is not found are assigned a running time of 3600 seconds. Points below the line indicate instances in which *IPS6* is faster than PMP$_{m1}$, and points above the line indicate that PMP$_{m1}$ is faster than *IPS6*. It can be seen a clear dominance of *IPS6* model on all the studied datasets. Across instances optimally solved by both models, *IPS6* cuts the running times down on average by 68% on EMM dataset (Figure 4a), by 38% on ZJY dataset (Figure 4b), by 26% on CV dataset (Figure 4c), and by 39% on BZ dataset (Figure 4d).

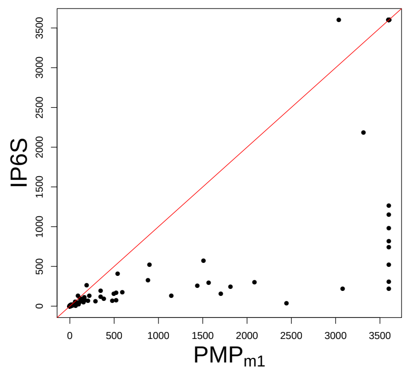
Table 9: Performance of the model proposed by Lee and Hsu, the PMP$_{m1}$ model from de Melo da Silva et al. (2018), and *IP6S* on the CV dataset from Caserta and Voß (2009) grouped by number of stacks (S) and number of tiers (H). CPU(s) represents the average running time (in seconds) on the instances optimally solved by each model (Opt).

			Lee and Hsu		PMP$_{m1}$		*IP6S*	
H	S	#	Opt.	CPU(s)	Opt.	CPU(s)	Opt.	CPU(s)
5	3	40	32	559.81	40	63.78	40	85.65
5	4	40	19	846.67	38	236.45	38	125.71
5	5	40	5	1997.08	31	402.37	32	452.33
5	6	40	2	278.40	23	995.09	23	817.05
5	7	40	0	-	10	1075.44	11	960.26
5	8	40	0	-	4	2034.74	7	1194.23
6	4	40	1	517.81	6	530.92	9	1425.93
6	5	40	0	-	2	1681.84	2	1244.29
6	6	40	0	-	1	1827.20	1	444.99
6	7	40	0	-	0	-	0	-
	Total	400	59	763.74	155	478.49	163	467.24

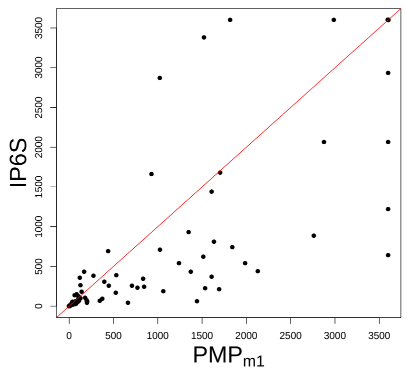
Table 10: Performance of the model proposed by Lee and Hsu, the PMP$_{m1}$ model from de Melo da Silva et al. (2018), and *IP6S* on the BZ dataset from van Brink and van der Zwaan (2014), grouped by number of tiers (H), stacks (S), and fill percentage (f). CPU(s) represents the average running time (in seconds) on the instances optimally solved by each model (Opt).

				Lee and Hsu		PMP$_{m1}$		*IP6S*	
H	S	f	#	Opt.	CPU(s)	Opt.	CPU(s)	Opt.	CPU(s)
4	3	0.50	60	60	0.12	60	0.02	60	0.02
4	3	0.70	60	60	8.06	60	0.88	60	0.91
4	5	0.50	60	60	0.82	60	0.08	60	0.07
4	5	0.70	60	58	103.74	60	1.97	60	1.53
4	7	0.50	60	60	10.92	60	0.21	60	0.17
4	7	0.70	60	53	288.35	60	5.91	60	3.73
4	9	0.50	60	59	107.54	60	1.29	60	1.07
4	9	0.70	60	40	108.08	60	94.92	60	22.03
6	3	0.50	60	60	127.78	60	2.62	60	3.00
6	3	0.70	60	47	301.43	59	82.33	60	102.96
6	5	0.50	60	49	134.79	60	11.62	60	18.59
6	5	0.70	60	22	640.10	44	466.39	44	264.09
6	7	0.50	60	43	271.25	59	123.31	59	101.49
6	7	0.70	60	11	182.38	35	404.02	37	234.19
6	9	0.50	60	31	321.56	59	156.63	60	118.95
6	9	0.70	60	4	336.68	32	659.36	33	385.65
		Total	960	717	140.40	888	94.94	893	62.02

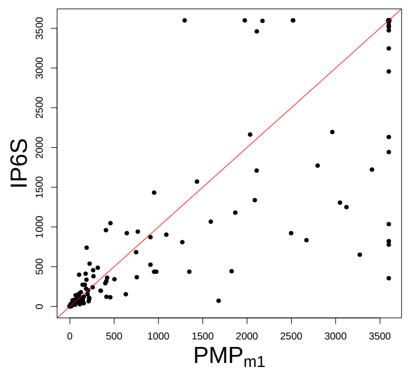
On the EMM dataset, *IPS6* solves 5 more instances of the most challenging category with 10 stacks and fill percentage of 75%. Moreover, the average running times across the 34 instances optimally solved by PMP$_{m1}$ and *IPS6* in this category are 580 and 154 seconds, respectively, which means an average reduction by 73%. On ZJY dataset, *IPS6* also outperforms PMP$_{m1}$ in all the categories, cutting the average running time by 44%



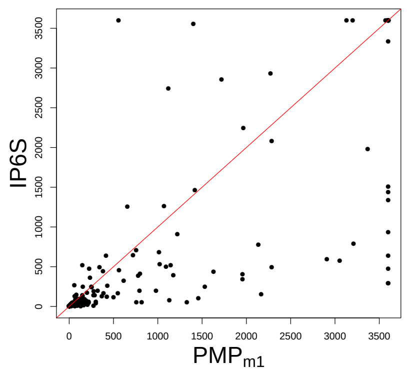
(a) EMM dataset



(b) ZJY dataset



(c) CV dataset



(d) BZ dataset

Figure 4: CPU times necessary for finding an optimal solution on all datasets with IP6S and PMP$_{m1}$ by de Melo da Silva et al. (2018). The running times are expressed in seconds and unsolved instances are assigned a CPU time of 3600 seconds.

in the most difficult category of 6 stacks and 5 tiers. The trend continues on CV and BZ datasets, solving *IPS6* more instances in the most challenging categories and reducing the average running time in almost all instances. There is just one category in which the average running time does not decrease with *IPS6* model. This is the easiest category of the CV dataset (5 tiers and 3 stacks) in which the average running time increases from 63 to 85 seconds. In the remainder categories, there is a high reduction of the running times.

8. Extensions of the model

The integer models developed in this study address the standard pre-marshalling problem. The objective is to minimize the number of moves required to order the bay and the constraints ensure that this rearrangement is made through a sequence of valid moves. However, other constraints could be included to produce solutions that are more useful in practice. This section outlines some of these possible extensions of the models.

8.1. Balancing the final configuration

The solution of the standard models could have an empty stack adjacent to a stack fully loaded up to tier H. This configuration is unstable and is avoided in practice. One way of preventing this type of solution could be to set a parameter κ as the maximum difference in the number of containers of adjacent stacks. By adding constraints

$$\sum_{h=1}^{H}\sum_{p=1}^{P} x_{shpT} - \sum_{h=1}^{H}\sum_{p=1}^{P} x_{s+1hpT} \leq \kappa \qquad \forall s \in \{1,\dots,S-1\} \tag{40}$$

$$\sum_{h=1}^{H}\sum_{p=1}^{P} x_{shpT} - \sum_{h=1}^{H}\sum_{p=1}^{P} x_{s+1hpT} \geq -\kappa \quad \forall s \in \{1,\dots,S-1\} \tag{41}$$

we ensure that the difference between adjacent stacks does not exceed κ.

8.2. Avoiding empty or full stacks

Constraints (42) would prevent empty stack is in the final configuration, while constraints (43) would prevent full stacks.

$$\sum_{h=1}^{H}\sum_{p=1}^{P} x_{shpT} \geq 1 \quad \forall s \in \{1,\dots,S\}; \tag{42}$$

$$\sum_{h=1}^{H}\sum_{p=1}^{P} x_{shpT} \leq H-1 \quad \forall s \in \{1,\dots,S\}; \tag{43}$$

8.3. Favouring stable configurations

Among the solutions with the minimum number of moves, a more stable configuration would be preferred. We could define two new variables h_{min} and h_{max} to indicate the minimum and maximum number of containers in a stack at time T, adding constraints (44) and (45) to define the variables and adding the term $(h_{max} - h_{min})/(H+1)$ to the objective function. Note that $(h_{max} - h_{min})/(H+1) \leq H/(H+1) < 1$ and therefore the optimal solution of the modified model would always correspond to the minimum number of moves.

$$\sum_{h=1}^{H} \sum_{p=1}^{P} x_{shpT} \leq h_{max} \quad \forall s \in \{1, \dots, S\} \tag{44}$$

$$\sum_{h=1}^{H} \sum_{p=1}^{P} x_{shpT} \geq h_{min} \quad \forall s \in \{1, \dots, S-1\} \tag{45}$$

8.4. Favouring that containers with the same priority share the same stack

It could be useful to have containers of the same priority one on top of the other in the same stack, making retrieving operations easier. That could be favoured by defining new variables:

$$g_{shp} = \begin{cases} 1, & \text{If a container of priority } p \text{ located at } (s,h) \text{ at time } T \\ & \text{is on top of other container of priority } p \\ 0, & \text{Otherwise} \end{cases}$$
$$\forall s \in \{1, \dots, S\}; \quad \forall h \in \{2, \dots, H\}; \quad \forall p \in \{1, \dots, P\};$$

and subtracting the term $\sum_s \sum_h \sum_p g_{shp} / \sum_{p=1}^{P} n_p$ from the objective function. Note that

$$\frac{\sum_s \sum_h \sum_p g_{shp}}{\sum_{p=1}^{P} n_p} \leq \frac{\sum_{p=1}^{P} (n_p - 1)}{\sum_{p=1}^{P} n_p} < 1$$

and therefore the optimal solution would still produce the minimum number of moves.

In order to define adequately the new variables, we would also need to include the constraints:

$$g_{shp} \leq x_{shpT} \qquad \forall s \in \{1, \dots, S\}; \quad \forall h \in \{2, \dots, H\} \tag{46}$$

$$g_{shp} \leq 1 - (x_{shpT} - x_{sh-1pT}) \quad \forall s \in \{1, \dots, S\}; \quad \forall h \in \{2, \dots, H\} \tag{47}$$

9. Conclusions and future work

In this paper we have studied the pre-marshalling problem, which plays an important role in the efficient management of storage yards, speeding up loading and unloading operations. This problem has been studied by developing integer formulations that can be easily implemented and adapted by researchers and practitioners. We have first proposed an initial model in which the moves are described using 3-indexed variables. As the linear relaxation was very weak, we developed new models using variables with 4, 5, and 6 indices. These models were conceptually better, strengthening the relationship between variables and improving the linear relaxations, but the dramatic increase in the number of variables offset their potential advantages. In a second phase, we decided to use different types of variables to describe the moves, separating the information about the origin of the move from that about its destination. In this case, models were improved with a small increase in the number of variables. We have proposed a total of eight integer linear models and we have also designed an iterative procedure to solve them, avoiding the need of an upper bound to build the model. These eight models have been extensively tested and results show the superior performance of the increasingly complex split-variable models. We have finally tested our best proposal, *IPS6*, and the previously published models by Lee and Hsu (2007) and de Melo da Silva et al. (2018) over an extended computational analysis that shows the better performance of our model on several well-known datasets.

Further lines of research will consider in more detail the extensions of the model outlined in Section 8 and also the possibility of changing the objective function to take into account the time required by the cranes to make the moves.

10. Acknowledgements

This study has been partially supported by the Spanish Ministry of Education, Culture, and Sport, FPU Grant A-2015-12849 and by the Spanish Ministry of Economy and Competitiveness, under projects DPI2014-53665-P and DPI2015-65895-R, partially financed with FEDER funds.

References

Bixby, R. E., 2002. Solving real-world linear programs: A decade and more of progress. Operations Research 50 (1), 3 – 15.

Bortfeldt, A., Forster, F., 2012. A tree search procedure for the container pre-marshalling problem. European Journal of Operational Research 217 (3), 531–540.

Caserta, M., Schwarze, S., Voß, S., 2011a. Container rehandling at maritime container terminals. In: Böse, J. W. (Ed.), Handbook of Terminal Planning. Springer New York, pp. 247–269.

Caserta, M., Schwarze, S., Voß, S., 2012. A mathematical formulation and complexity considerations for the blocks relocation problem. European Journal of Operational Research 219 (1), 96–104.

Caserta, M., Voß, S., 2009. A corridor method-based algorithm for the pre-marshalling problem. In: Giacobini, M., Brabazon, A., Cagnoni, S., Di Caro, G. A., Ekárt, A., Esparcia-Alcázar, A. I., Farooq, M., Fink, A., Machado, P. (Eds.), Applications of Evolutionary Computing. Springer Berlin Heidelberg, pp. 788–797.

Caserta, M., Voß, S., Sniedovich, M., 2011b. Applying the corridor method to a blocks relocation problem. OR spectrum 33 (4), 915–929.

Choe, R., Park, T., Oh, M.-S., Kang, J., Ryu, K. R., 2011. Generating a rehandling-free intra-block remarshaling plan for an automated container yard. Journal of Intelligent Manufacturing 22 (2), 201–217.

de Melo da Silva, M., Toulouse, S., Calvo, R. W., 2018. A new effective unified model for solving the pre-marshalling and block relocation problems. European Journal of Operational Research 271 (1), 40 – 56.

Expósito-Izquierdo, C., Melián-Batista, B., Moreno-Vega, M., 2012. Pre-Marshalling Problem: Heuristic solution method and instances generator. Expert Systems with Applications 39 (9), 8337–8349.

Gheith, M., Eltawil, A. B., Harraz, N. A., 2016. Solving the container pre-marshalling problem using variable length genetic algorithms. Engineering Optimization 48 (4), 687–705.

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

Huang, S.-H., Lin, T.-H., 2012. Heuristic algorithms for container pre-marshalling problems. Computers & Industrial Engineering 62 (1), 13–20.

Jovanovic, R., Tuba, M., Voß, S., 2017. A multi-heuristic approach for solving the pre-marshalling problem. Central European Journal of Operations Research 25 (1), 1–28.

Kang, J., Oh, M.-S., Ahn, E. Y., Ryu, K. R., Kim, K. H., 2006. Planning for intra-block remarshalling in a container terminal. In: Ali, M., Dapoigny, R. (Eds.), Advances in Applied Artificial Intelligence. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1211–1220.

Kang, J., Oh, M.-S., Ryu, K. R., Kim, K. H., 2005. Sequencing container moves for intra-block remarshalling in a container terminal yard. Journal of Navigation and Port Research 29 (1), 83–90.

Lee, Y., Chao, S.-L., 2009. A neighborhood search heuristic for pre-marshalling export containers. European Journal of Operational Research 196 (2), 468–475.

Lee, Y., Hsu, N. Y., 2007. An optimization model for the container pre-marshalling problem. Computers and Operations Research 34 (11), 3295–3313.

Lehnfeld, J., Knust, S., 2014. Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. European Journal of Operational Research 239 (2), 297–312.

Petering, M. E., 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.

Sniedovich, M., Voß, S., 2006. The corridor method: a dynamic programming inspired metaheuristic. Control and Cybernetics 35, 551–578.

Tanaka, S., Tierney, K., 2018. Solving real-world sized container pre-marshalling problems with an iterative deepening branch-and-bound algorithm. European Journal of Operational Research 264 (1), 165 – 180.

Tierney, K., Pacino, D., Voß, S., 2016. Solving the pre-marshalling problem to optimality with A* and IDA*. Flexible Services and Manufacturing Journal 29 (2), 1–37.

UNCTAD, 2017. Review of Maritime Transport. Tech. rep., United Nations Conference on Trade and Development.

van Brink, M., van der Zwaan, R., 2014. A branch and price procedure for the container premarshalling problem. In: European Symposium on Algorithms. Springer, pp. 798–809.

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

Wang, N., Jin, B., Zhang, Z., Lim, A., 2017. A feasibility-based heuristic for the container pre-marshalling problem. European Journal of Operational Research 256 (1), 90–101.

Zhang, R., Jiang, Z.-Z., Yun, W. Y., 2015. Stack pre-marshalling problem: A heuristic-guided branch-and-bound algorithm. International Journal of Industrial Engineering 22 (5), 509 – 523.