

Document downloaded from:

<http://hdl.handle.net/10251/165899>

This paper must be cited as:

Parreño-Torres, C.; Álvarez-Valdés, R.; Ruiz García, R.; Tierney, K. (2020). Minimizing crane times in pre-marshalling problems. *Transportation Research Part E Logistics and Transportation Review*. 137:1-20. <https://doi.org/10.1016/j.tre.2020.101917>



The final publication is available at

<https://doi.org/10.1016/j.tre.2020.101917>

Copyright Elsevier

Additional Information

Minimizing crane times in pre-marshalling problems

Consuelo Parreño-Torres^{b,*}, Ramon Alvarez-Valdes^b, Rubén Ruiz^c, Kevin Tierney^a

^a*Decision and Operation Technologies Group, Bielefeld University, Universitätsstraße 25 D-33615, Bielefeld, Germany.*

^b*Department of Statistics and Operations Research, Valencia University, Doctor Moliner 50, Burjassot, 46100, Valencia, Spain.*

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

Abstract

The pre-marshalling problem has been extensively studied in recent years with the aim of minimizing the number of movements needed to rearrange the bay. Since we consider time to be a more realistic objective for measuring process efficiency and does not correlate with the number of movements, we study the problem of minimizing crane times and develop two exact approaches to solve it: an integer linear model, and a branch and bound algorithm, with new upper and lower bounds, dominance criteria, and a heuristic procedure to provide feasible solutions. An extensive computational study shows the performance of both approaches.

Keywords: logistics, container pre-marshalling, crane time, maritime transport, terminal operations

1. Introduction

Prior to the 1970s, loading and unloading ships was an expensive and labor-intensive task, making it unprofitable to transport many types of cargo overseas. With the standardization of containers, port efficiency skyrocketed, drastically reducing shipping costs, and resulted in a shift of production to countries with cheaper manufacturing costs. In this context, containerization is understood as one of the main drivers of globalization. The import costs of many products have decreased to such an extent that it is sometimes cheaper to transport goods to the other side of the world than to produce them locally. According to UNCTAD (2018), container ports handled 752.2 million 20-foot equivalent units (TEUs) in 2017, and global container port throughput rose

*Corresponding author

by 6%, three times the rate of 2016. Adding to the challenges in the sector is not only the rise in the volume of cargo transported worldwide, but also the increase in the size of ships. The largest container ships today can handle more than 20,000 TEUs (e.g., the MSC Mina and MSC Gülsün).

The increasing size of ships and creation of strategic liner shipping alliances have triggered a new dynamic in which shipping lines have greater bargaining power and influence over the container terminals and in which terminals must compete for fewer services from larger ships. The need to handle more containers in the same time puts pressure on both berthing and yard operations. A good yard arrangement before the arrival of ships contributes significantly to soften workload peaks and to an increase in terminal productivity, which refers to the number of container moves per hour of time spent by ships in port, weighted by the call size (UNCTAD, 2018).

Pre-marshalling operations contribute to the proper arrangement of the yard prior to the arrival of the ship, using the yard cranes when the workload in the terminal is at a minimum, thus accelerating service times once the ship arrives. It consists of sorting the *bays* (rows of container stacks in the yard) so that, when the containers are required, they can be accessed without additional relocations. The classic objective of the container pre-marshalling problem (CPMP) is to minimize the number of moves to transform the initial layout of a bay into a final layout without any containers blocking the removal of others. The number of moves has previously been used as an indication of the time employed by the crane to rearrange the bay. However, in many cases, the number of moves is not entirely representative of crane times. For example, it takes significantly more time to move a container between the farthest slots of a bay than to do so from the top level of a stack to the top level of an adjacent stack. Furthermore, such moves differ not only in the amount of time required by the crane, but also in terms of the energy consumption. According to Wilmsmeier and Spengler (2016), horizontal activities, such as those used by yard cranes, consume a total of 45% share of energy in the terminal. Their findings regarding the current energy consumption of container terminals worldwide highlight the need for action to address competitiveness, energy security, and climate change. With most of the terminal energy being consumed by yard cranes, it becomes clear how important it is to reduce crane times in order to support green port policies.

We study a variant of the CPMP in which the goal is to minimize the time spent by the crane to transform the bay into one without blocking containers. We refer to this problem as the pre-marshalling problem with crane time minimization objective, CPMPCT. We compute the crane

time by considering whether the movement made is in the vertical or the horizontal axis and if the movement is loaded or unloaded, since the speed and acceleration of the crane depends on these aspects. We also consider twistlock¹ times that are proportional to the tier where the container to be moved is located. With a precise estimation of the crane time, we carry out a computational study that motivates the importance of the problem and shows the non-direct relation between the number of moves and the crane time. To solve the problem optimally, we propose two different approaches: a mathematical formulation, and a branch-and-bound based approach. Moreover, we propose new upper and lower bounds for the number of moves and for the time spent by the crane to solve the CPMPCT, and we deal with dominance rules proposed in previous academic works discussing whether or not they can be adapted to this new objective function. Our contributions are tested on an extensive computational analysis, using datasets from the literature, and the results show that instances of up to 7 tiers and 5 stacks can be solved to optimality.

We first review the relevant literature in Section 2. In section 3, we formally describe the new problem of minimizing the crane times, showing that there is no direct relation with the classic objective of minimizing the number of times and how time cranes can be calculated. Section 4 introduces new upper and lower bounds and dominance criteria that would be used in the exact approaches. The integer linear model is presented in Section 5, while the components of the branch and bound algorithm are described in Section 6. The computational results are provided in Section 7 and conclusions and future work are discussed in Section 8.

2. Literature review

Pre-marshalling problems have been extensively studied in recent years both in terms of exact and heuristic approaches. Concerning mathematical models, Lee and Hsu (2007) propose a first integer linear model based on a multicommodity network flow. More recently, de Melo da Silva et al. (2018) have developed a unified model for pre-marshalling and the closely related block relocation problem. They discretize time, with a move occurring at each time period. An upper bound is required and it is obtained using a simple heuristic. Parreño-Torres et al. (2019) explore several alternative ways of modeling the problem, also discretizing time, but they develop an iterative

¹A twistlock and corner casting together form a standardized rotating connector for securing shipping containers that is used for lifting the containers by the cranes.

procedure in which the upper bound on the number of moves is not needed. van Brink and van der Zwaan (2014) propose an IP model for which they apply column generation and develop a branch-and-price procedure. Other exact approaches have also been proposed, such as the A^* algorithm of Expósito-Izquierdo et al. (2012) or the iterative deepening A^* (IDA*) of Tierney et al. (2017). Several branch and bound algorithms have also been developed in recent years (Prandtstetter, 2013; Zhang et al., 2015; Tanaka and Tierney, 2018; Tanaka et al., 2019).

Concerning heuristics, Caserta and Voß (2009) develop a heuristic algorithm based on the Corridor Method, Lee and Chao (2009) a neighborhood search algorithm, and Bortfeldt and Forster (2012) a heuristic tree search procedure. Jovanovic et al. (2017) develop a deterministic algorithm based on the randomized greedy procedure by Expósito-Izquierdo et al. (2012). Further heuristic proposals are the biased random-key genetic algorithm (Hottung and Tierney, 2016), target-guided procedures (Wang et al., 2015, 2017) and the deep learning tree search of Hottung et al. (2020).

However, all previously mentioned models and algorithms use the minimization of the number of moves required to rearrange the bay as the objective function. It is a simple and intuitive measure of the effort required, and allows for the development of high-quality bounds and dominance criteria. However, as will be shown later, the number of moves does not accurately represent the time or the cost needed for the pre-marshalling process.

In the closely related block relocation problem, in which containers have to be retrieved from a bay in a given order, several authors have considered as the objective the time required by the crane to perform these moves, although most studies consider the minimization of the moves required. Lee and Lee (2010) use as their objective function the weighted sum of the number of moves and the crane working time. Lin et al. (2015) develop a heuristic algorithm to minimize the number of moves. The rule used to select the stack to which a container should be relocated to penalizes moves to distant stacks instead of favoring shorter crane moves and therefore shorter working times. They show that solutions with the minimum number of moves do not produce minimum working times and the tradeoff between these objectives can be controlled by adjusting the penalty values. Jovanovic et al. (2019) develop an ant colony optimization algorithm, initially considering the standard minimization of the number of moves, but then adapt the same algorithm to the objective of minimizing the total crane working time. da Silva Firmino et al. (2019) only consider the crane working time as the objective of their GRASP algorithm. It should be noted

that in all these studies the crane time is calculated in very simple ways, just considering a linear combination of the number of stacks and tiers involved in the container movements. By contrast, in this paper we consider a more complex crane time computation, considering different speeds for vertical and horizontal movements (with and without load), as well as crane acceleration and twistlock times. This crane time closely resembles real observed crane working times in practice.

3. Pre-marshalling problem with crane time minimization objective

The pre-marshalling problem has so far been studied with the aim of minimizing the number of crane moves. In this section we propose to minimize the time required by the crane to sort the bay, calculating the crane time precisely. In addition, we motivate the use of this new objective function by giving examples and computational results that show that decreasing the number of crane moves is not always linked to a decrease in crane time.

3.1. Problem description

The container yard is a large space where inbound containers are stored before being transferred to the hinterland, and outbound containers are also stored before being loaded onto ships. The yard is divided into several blocks that are composed of parallel bays. Each bay has the same number of stacks of containers that have a maximum number of tiers up to a height that is fixed according to the height of the crane, see Figures 1 and 2. The problem of pre-marshalling with a crane time minimization objective seeks to obtain a sequence of moves that minimizes the time spent by the crane to transform the initial configuration of a bay into a final one in which no containers are blocking the retrieval of others. The main constraint in this problem is that containers do not leave the bay during the sequence of moves.

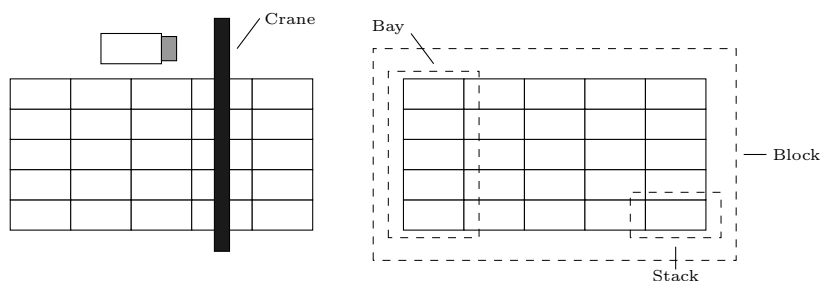


Figure 1: Container yard scheme.

We formally define \mathcal{S} as the set of stacks of the bay, where $S = |\mathcal{S}|$ is the total number of stacks, and \mathcal{H} as the set of tiers, where $H = |\mathcal{H}|$ represents the highest tier of the stacks. Each of the positions in the bay can be described by a tuple (s, h) where $s \in \mathcal{S}$ and $h \in \mathcal{H}$. The order in which containers will be moved out of the bay is known in advance. We therefore assign a priority to each container $p \in \{1, \dots, P\} = \mathcal{P}$, where 1 is the highest priority and P the lowest.

Moves are defined by indicating the origin and destination stacks and tiers. Therefore, a solution is a sequence of moves of the form $(s, h, k, e) : s \neq k \in \mathcal{S}$ and $h, e \in \mathcal{H}$ where (s, h) is the origin position of the container being moved and (k, e) its destination position. Given a solution $sol = (s_1, h_1, k_1, e_1) \dots (s_i, h_i, k_i, e_i) \dots (s_n, h_n, k_n, e_n)$, the time spent by the crane is denoted as $time(sol)$. We include the time spent by the crane to position the spreader from its initial position outside of the bay to the position of the first container to be moved. We also distinguish the movements carried out by the crane with and without a loaded container. The crane moves unloaded from the destination of the last container moved to the position of the next container to be moved. The crane then picks up the next container to be moved, and moves loaded to the container's destination. The function $p_i(s, h)$ gives the priority of the container stored at position (s, h) after the i -th move. The bay is considered ordered if $p_n(s, h) \geq p_n(s, h + 1)$ for all $s \in \mathcal{S}$ and $h \in \mathcal{H} \setminus \{H\}$.

3.2. Parameters used to compute the time required by the crane

There are several types of cranes used to handle containers in port yards, each with different technical specifications. In this work we consider the rubber tired gantry (RTG) cranes used in the Noatum terminal of Valencia's port in Spain. The company Noatum facilitated this information during their collaboration in the EU Project Transforming Transport (TT, Grant agreement No. 731932). The information corresponds to an RTG Transtainer 79 from Konecranes. To move a container, the crane is first positioned above it, moving horizontally along the top travel lane to the corresponding stack and then moving down to reach the requested position. Then the container is twistlocked, hoisted up, moved horizontally to the destination stack and hoisted down to its new position. This is a common operation that can be observed in most gantry cranes.

Figure 2 shows the structure of the RTG crane. The values of the parameters represented in the figure are shown in Table 1. Note that in this work we assume (w.l.o.g.) that all containers in the bay have the same dimensions.

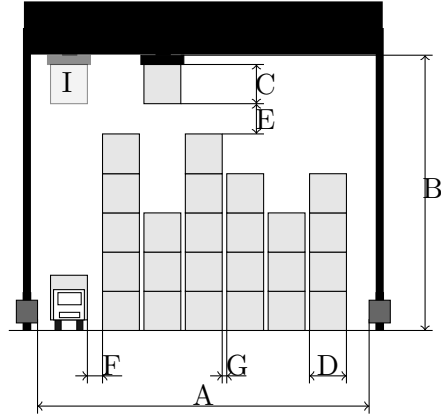


Figure 2: Front view of an RTG crane.

Table 1: Parameters referring to storage distances represented in Figure 2.

Letter	Notation	Description	Value
A	-	Useful distance between legs [m]	21.925
B	-	Lifting height under spreader [m]	18.200
C	ch	Container height [m]	2.591
D	cw	Container width [m]	2.438
E	$vsep$	Distance between the top level and the container at the topmost tier [m]	2.000
F	$hsep$	Distance between first stack and the truck (or the initial point axis x) [m]	1.000
G	$msep$	Distance between containers [m]	0.300

The data provided by the company leads to a differentiation between the time spent by the crane according to whether it performs a loaded vertical move ($vmax^{vl}$), an unloaded vertical move ($vmax^{vu}$), a loaded horizontal move ($vmax^{rl}$) or an unloaded horizontal move ($vmax^{ru}$). The maximum speeds of the crane for each of these cases and the horizontal and vertical distance the crane has to travel to reach them can be seen in Table 2. Furthermore, this table shows the twistlock time, which is directly proportional to the level at which the container to move is placed. This is due to the oscillation of the crane spreader, which depends on the cable's length, due to the sway motion of the suspended load. Crane operators have to wait until the oscillation stops to twistlock the container. Some cranes employ anti-sway systems that arrest the pendulums and the rotational sway motion, resulting in reduced twistlock times, but this is not the case for the studied cranes. We suppose that the crane moves horizontally and requires d' meters to reach its

maximum speed. Up to that distance, the acceleration is constant and therefore depicts a uniformly accelerated rectilinear motion. Once it reaches the maximum speed, the crane follows a uniformly rectilinear motion and starts to decelerate when there are d^r meters left until its destination point ($a' = -a$). If the distance travelled by the crane is less than $2d^r$, half the distance is accelerating and the other half is decelerating.

Table 2: Parameters employed to compute the crane costs provided by the company.

Notation	Description	Value
$vmax^{vl}$	Hoisting time loaded [m/s]	0.5
$vmax^{vu}$	Hoisting time unloaded [m/s]	1.00
$vmax^{rl}$	Travelling speed loaded [m/s]	$1.1\bar{6}$
$vmax^{ru}$	Travelling speed unloaded [m/s]	$2.1\bar{6}$
d^v	Distance to max. hoisting speed [m]	2.65
d^r	Distance to max. travelling speed [m]	2.50
b_h	Twistlock time at level h [s]	$5 \cdot (H-h+1)$

Once the maximum speeds of the crane and the distances required to reach them have been set, equation (1) calculates the acceleration [m/s^2] for the moves considered.

$$a^{\alpha\beta} = \frac{(vmax^{\alpha\beta})^2}{2d^\alpha} \quad \forall \alpha \in \{v, r\}; \quad \forall \beta \in \{l, u\} \quad (1)$$

Let x be the distance travelled by the crane (vertically or horizontally) in meters. The time spent by the crane (loaded or unloaded) in seconds to carry out the move can be calculated according to equation (2).

$$t^{\alpha\beta}(x) = \begin{cases} \frac{2\sqrt{a^{\alpha\beta}x}}{a^{\alpha\beta}}, & \text{If } x < 2d^\alpha \\ \frac{2vmax^{\alpha\beta}}{a^{\alpha\beta}} + \frac{x-2d^\alpha}{vmax^{\alpha\beta}}, & \text{If } x \geq 2d^\alpha \end{cases} \quad \forall \alpha \in \{v, r\}; \quad \forall \beta \in \{l, u\} \quad (2)$$

With reference to the distances described in Table 1, the distance in meters between the top travel lane and level h , $distance^v(h)$, follows equation (3). Furthermore, the distance in meters between stacks s and k , $distance^r(s, k)$, follows equation (4). If $k = 0$, the crane is moved from (or

to) the initial point, represented as I in Figure 2, to (or from) stack s .

$$distance^v(h) = vsep + (H - h + 1) \cdot ch \quad (3)$$

$$distance^r(s, k) = \begin{cases} |k - s| \cdot (msep + cw), & \text{If } k \neq 0 \\ (s - 1) \cdot msep + s \cdot cw + hsep, & \text{If } k = 0 \end{cases} \quad (4)$$

Note that if $s < S$, $distance^r(s, s + p) = distance^r(1, p + 1)$ for all $p \in \mathbb{N}$ such that $p < S - s$. Otherwise, $distance^r(s, s - p) = distance^r(1, p + 1)$ for all $p \in \mathbb{N}$ such that $p \leq S$. With all parameters adjusted, we then calculate the individual costs to compute the total time spent in the rearrangement of the bay following the sequence $sol = (s_1, h_1, k_1, e_1) \dots (s_i, h_i, k_i, e_i) \dots (s_n, h_n, k_n, e_n)$.

$$time(sol) = t(s_1, h_1, k_1, e_1) + \dots + t(s_i, h_i, k_i, e_i) + \dots + t(s_n, h_n, k_n, e_n) \quad (5)$$

where:

$$t(s_i, h_i, k_i, e_i) = \begin{cases} c_{k_i-1s_i}^0 + v_{h_i}^0 + b_{h_i} + v_{h_1}^1 + c_{s_i k_i}^1 + v_{e_i}^1 + v_{e_i}^0, & \text{If } i \neq 0 \\ c_{s_i 0}^0 + v_{h_i}^0 + b_{h_i} + v_{h_1}^1 + c_{s_i k_i}^1 + v_{e_i}^1 + v_{e_i}^0 & \text{If } i = 0 \end{cases} \quad (6)$$

$$\begin{aligned} v_h^0 &= t^{vu}(distance^v(h)); & v_h^1 &= t^{vl}(distance^v(h)); \\ c_{sk}^0 &= t^{ru}(distance^r(s, k)); & c_{sk}^1 &= t^{rl}(distance^r(s, k)); \end{aligned} \quad (7)$$

For each move we consider the time spent to move the crane (unloaded) along the upper travel line from its current position to the origin stack of the next container to be moved, the time spent to move the crane down to reach the container, the twistlock time, the time spent to hoist the container up, the time spent to move the crane (loaded) along the upper travel line to the destination stack, the time spent to move the crane down to release the container, and the time spent to position the crane on the upper travel line.

Hence, a lower bound for the time spent to perform a move t_{min} is:

$$t_{min} = c_{12}^0 + v_H^0 + b_H + v_H^1 + c_{12}^1 + v_H^1 + v_H^0 = (2v_H^0 + c_{12}^0) + (2v_H^1 + b_H + c_{12}^1) \quad (8)$$

in which the crane moves to an adjacent stack, picks up a container at the topmost tier, and

moves it to the topmost tier of an adjacent stack.

3.3. Crane time against number of movements

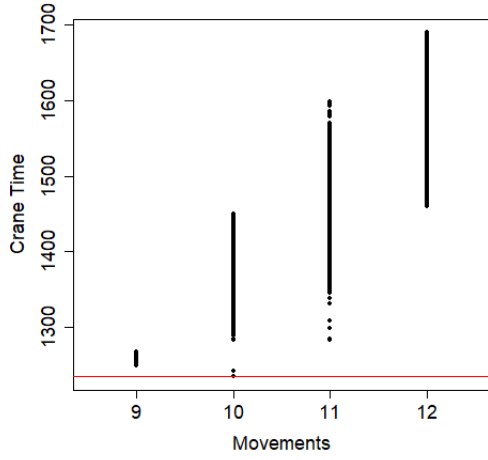
We now study the relation between the number of moves required to rearrange the bay and the time taken by the crane to do it. Figure 3 shows the information of a large set of solutions of four instances belonging to datasets frequently studied in the literature. The x -axis describes the number of movements to solve the CPMP, with the minimum value being the optimal solution. The y -axis represents the total crane time measured in seconds. All the alternate optimal solutions for the CPMP are represented, whereas for a greater number of movements only a set of solutions is represented. It can be seen that the crane time can increase up to 24% over two feasible solutions with the same number of moves. It can also be seen that an optimal solution for the CPMP can have a crane time higher than a non-optimal solution. Even though the examples represented are small, notice that the variation in the crane time between solutions with the same number of moves increases with the number of moves. The best solution found for the CPMPCT, not necessarily the optimal one for this problem, is the optimal one for the CPMP in the cases shown in Figure 3b and 3c. However, in Figures 3a and 3d the best solution found for the CPMPCT is not the optimal one for the CPMP.

Figure 4 illustrates an example in which the minimum number of moves to solve the CPMP is 3, with a crane time of 393.7 seconds (Figure 4a). However, the number of moves to optimally solve the CPMPCT is 4, with a crane time of 386.5 seconds (Figure 4b).

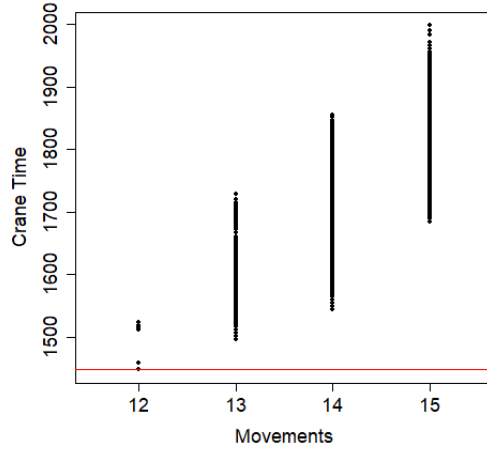
4. Bounds and dominance criteria

This section proposes upper bounds for the number of moves, lower bounds for the time spent by the crane, as well as dominance rules to be used in the models and algorithms developed in this paper for the CPMPCT.

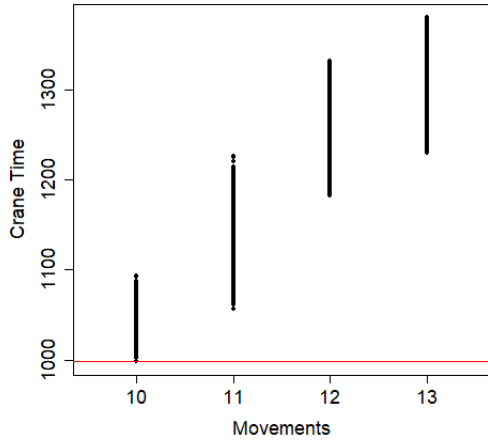
In the following proofs, let sol be a feasible solution for the CPMP and $time(sol)$ be the time spent by the crane to carry out the solution. We denote as γ the number of blocking containers in the initial layout of the bay and $(s_{b1}, h_{b1}), \dots, (s_{b\gamma}, h_{b\gamma})$ the positions where they are initially stored.



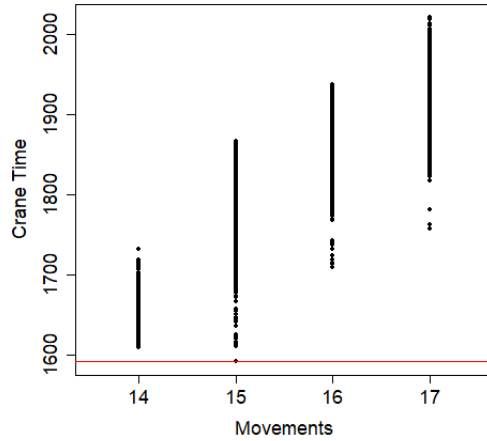
(a) BZ: data_p6_s5_h6_f50_2.txt



(b) CV: data4-5-36.txt



(c) EMM: emm_s7_t4_p0.75_c1.21.txt

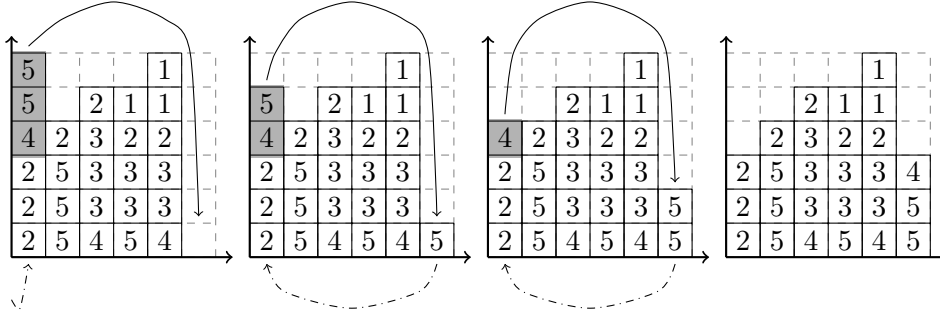


(d) ZJY: 6_5_7.txt

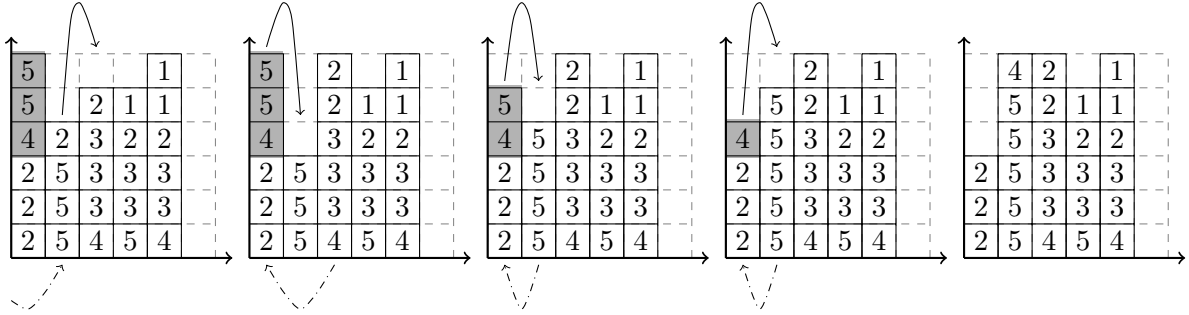
Figure 3: Comparison of crane times and number movements on instances from datasets studied in the state-of-the-art. The horizontal line in red indicates the shortest crane time shown in the figure.

4.1. Upper bound for the number of moves to solve the CPMPTC

Establishing an upper bound on the number of moves to rearrange the bay is not easy in the classic CPMP, but it can be done using an iterative procedure, progressively increasing the number of moves until a feasible solution is found (Parreño-Torres et al., 2019). When we solve the CPMPTC, we cannot determine this upper bound by a similar iterative procedure because it is possible to get better solutions by increasing the number of moves. Nor can we set the bound as the number of moves of any solution to the CPMP, as the optimal solution may have a greater number



(a) Optimal sequence of moves to solve the CPMP. The crane time is 393.696 seconds.



(b) Optimal sequence of moves to solve the CPMPCT. The crane time is 386.534 seconds.

Figure 4: An example layout in which the optimal sequence of moves to solve the CPMPCT has a larger number of moves than the optimal sequence to solve the CPMP. Blocking containers are shown in gray and arcs represent the movements of the crane loaded (continuous lines) and unloaded (dashed lines).

of moves. However, when using mathematical models for solving the problem, it is necessary to have an upper bound on the number of movements to ensure that we are solving the problem optimally.

Proposition 1. *The optimal number of moves to solve the CPMPCT is bounded above by $\lfloor \frac{time(sol)}{t_{min}} \rfloor$.*

Proof. Let $sol^* = (s_1, h_1, k_1, e_1) \dots (s_i, h_i, k_i, e_i) \dots (s_n, h_n, k_n, e_n)$ be an optimal solution for the CPMPCT. We compute the total time spent by the crane as follows:

$$\begin{aligned} time(sol^*) &= t(s_1, h_1, k_1, e_1) + \dots + t(s_i, h_i, k_i, e_i) + \dots + t(s_n, h_n, k_n, e_n) \\ &\geq t_{min} + \dots + t_{min} + \dots + t_{min} \geq n \cdot t_{min} \end{aligned}$$

As sol^* is the optimal solution to solve the CPMPCT and $t_{min} > 0$,

$$time(sol) \geq time(sol^*) \geq n \cdot t_{min} \iff \frac{time(sol)}{t_{min}} \geq n$$

Since $n \in \mathbb{N}$, $n \leq \lfloor \frac{time(sol)}{t_{min}} \rfloor$.

□

Proposition 2. *The optimal number of moves to solve the CPMPCT is bounded above by:*

$$\left\lceil \frac{\text{time}(\text{sol}) - \sum_{i=1}^{\gamma} \left(c_{12}^0 + v_{h_{b_i}}^0 + b_{h_{b_i}} + v_{h_{b_i}}^1 + c_{12}^1 + v_H^1 + v_H^0 \right) + (\gamma \cdot t_{\min})}{t_{\min}} \right\rceil$$

Proof. In order to arrange the bay, all the blocking containers in the initial layout must be moved at least once. Let $\text{sol}^* = (s_1, h_1, k_1, e_1) \dots (s_i, h_i, k_i, e_i) \dots (s_n, h_n, k_n, e_n)$ be an optimal solution of the CPMPCT. The blocking containers should be moved from their positions so the spreader has to go down to their initial tier and the following inequality is satisfied:

$$\text{time}(\text{sol}) \geq \text{time}(\text{sol}^*) \geq (n - \gamma) \cdot t_{\min} + \sum_{i=1}^{\gamma} \left(c_{12}^0 + v_{h_{b_i}}^0 + b_{h_{b_i}} + v_{h_{b_i}}^1 + c_{12}^1 + v_H^1 + v_H^0 \right)$$

Since $n \in \mathbb{N}$, we reformulate the inequality above to get:

$$\left\lceil \frac{\text{time}(\text{sol}) - \sum_{i=1}^{\gamma} \left(c_{12}^0 + v_{h_{b_i}}^0 + b_{h_{b_i}} + v_{h_{b_i}}^1 + c_{12}^1 + v_H^1 + v_H^0 \right) + (\gamma \cdot t_{\min})}{t_{\min}} \right\rceil \geq n$$

□

With these proofs, we can take advantage of the many heuristic and exact procedures developed for the CPMP to obtain feasible solutions in very short times.

4.2. Lower bound for the time spent by the crane

Let $LB := IBF^4$ be the lower bound for the number of moves described in Tanaka et al. (2019). We use this bound to propose valid lower bounds for the total time spent by the crane.

Proposition 3. $LB_{ct}^0 := LB \cdot t_{\min}$ is a valid lower bound for the CPMPCT.

Proof. If LB is the smallest number of moves to sort the bay, the minimum time spent by the crane is that number multiplied by the minimum time spent to perform a single move. □

Proposition 4. A valid lower bound for the CPMPCT is:

$$LB_{ct}^1 := (LB - \gamma) \cdot t_{\min} + \sum_{i=1}^{\gamma} \left(c_{12}^0 + v_{h_{b_i}}^0 + b_{h_{b_i}} + v_{h_{b_i}}^1 + c_{12}^1 + v_H^1 + v_H^0 \right) \quad (9)$$

Proof. To sort the bay in the required order, all the blocking containers must be moved at least once, so the crane has to go down to tier h_{b_i} to move blocking container i . Therefore, the minimum time to move container i is $t_{min}(b_i) = \left(c_{12}^0 + v_{h_{b_i}}^0 + b_{h_{b_i}} + v_{h_{b_i}}^1 + c_{12}^1 + v_H^1 + v_H^0 \right)$, which is greater than (or equal to) t_{min} . \square

Let $h'_{b_1}, \dots, h'_{b_\gamma}$ be the γ highest tiers to which the γ blocking containers in the initial layout of the bay could be moved on top of non blocking containers without considering blocking containers. See Figure 5 for further details.

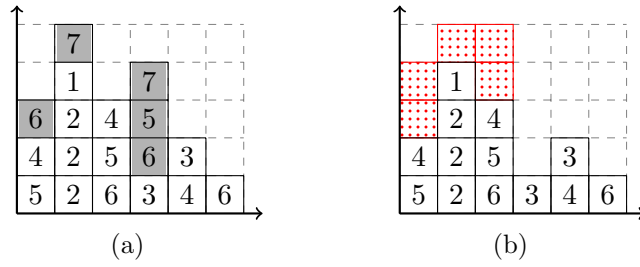


Figure 5: Figure (a) shows a layout with 5 blocking containers. Only the non blocking containers are represented in Figure (b) and the 5 highest tiers to which the blocking containers could be moved are highlighted in red.

Proposition 5. *A valid lower bound for the CPMPT is:*

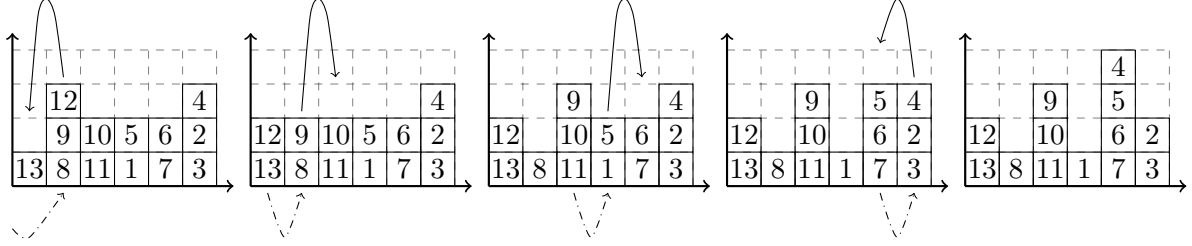
$$LB_{ct}^2 := (LB - \gamma) \cdot t_{min} + \sum_{i=1}^{\gamma} \left(c_{12}^0 + v_{h_{b_i}}^0 + b_{h_{b_i}} + v_{h_{b_i}}^1 + c_{12}^1 + v_{h'_{b_i}}^1 + v_{h'_{b_i}}^0 \right) \quad (10)$$

Proof. Starting from the initial layout of the bay, all blocking containers must be moved and the highest tiers to which they can be moved are $h'_{b_1}, \dots, h'_{b_\gamma}$. Suppose that we can move some blocking container to a higher position, this happens in the case that a non blocking container has been placed in one of the levels $h'_{b_1}, \dots, h'_{b_\gamma}$ and therefore this vertical distance has already been accounted for, so the bound holds. \square

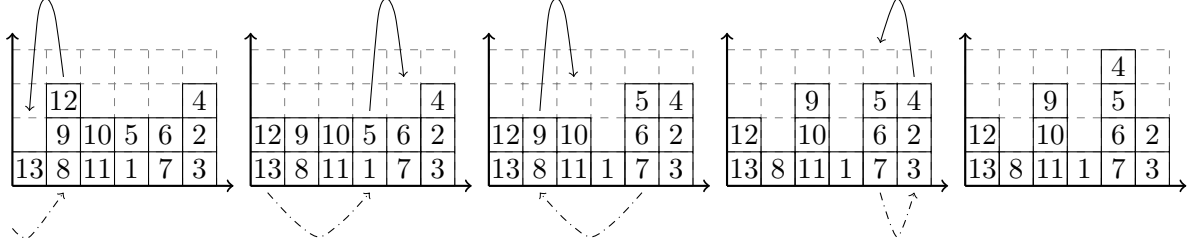
4.3. Dominance criteria

Dominance rules for the CPMP have been highly studied in the literature (Tanaka et al. (2019); Tierney et al. (2017); Tanaka and Tierney (2018)) and have shown their effectiveness in reducing the number of nodes to be explored in branch and bound algorithms. Nevertheless, when we study the CPMPT we cannot consider the vast majority of them. For example, empty stack rules stating that if there is more than one empty stack only the move to the left-most stack is considered

are not satisfied here because all empty stacks are not equivalent in this problem. Nor could we consider unrelated move symmetries because even in the simplest case in which two consecutive moves have different origin and destination stack the order of the movements is relevant as can be seen in Figure 6. Two sequences of 4 moves, leading to the same configuration are not equivalent in terms of crane times.



(a) The time spent by the crane to perform the sequence (2,1)(2,3)(4,5)(6,5) is 405.813 seconds.



(b) The time spent by the crane to perform the sequence (2,1)(4,5)(2,3)(6,5) is 413.864 seconds.

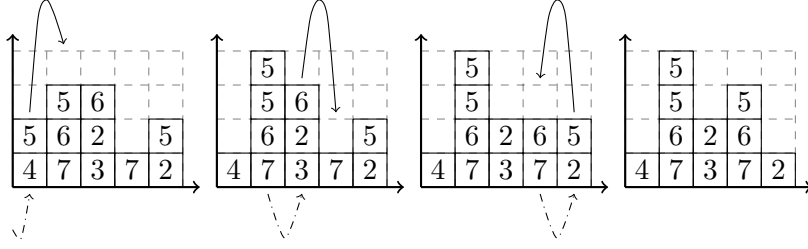
Figure 6: An example showing two sequences considered as unrelated move symmetries in the CPMP whose crane time is not the same.

Eliminating same group symmetries, which involve the relocation of two containers with the same priority, is highly relevant in the context of the CPMP. However, in the CPMPCT, two movements do not take the same crane time if the origin or/and destination stack is modified, so most of these rules are not satisfied. In Figure 7 two sequences of 3 moves, involving two containers of priority 5 lead to the same configuration but with different times. The only same group rule that we can apply to the CPMPCT states that if a container of priority p is moved from one stack s , no container of priority p will be moved to that stack in the next movement.

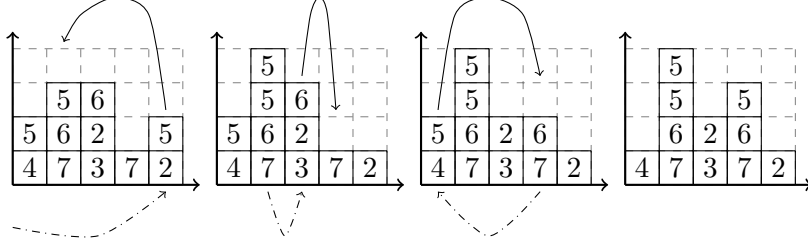
Proposition 6. Let $t1 = c_{s_i k_i}^1 + c_{k_i s_{i+1}}^0 + c_{s_{i+1} s_i}^1$ and $t2 = c_{s_i s_{i+1}}^0 + c_{s_{i+1} k_i}^1 + c_{k_i s_i}^0$, then $t1 > t2$.

Proof. See Appendix A. □

Proposition 7. A sequence $sol = (s_1, h_1, k_1, e_1) \dots (s_i, h_i, k_i, e_i)(s_{i+1}, h_{i+1}, s_i, h_i) \dots (s_n, h_n, k_n, e_n)$ is disallowed for the CPMPCT if $p_{i-1}(s_i, h_i) = p_{i+1}(s_i, h_i)$.



(a) The time spent by the crane to perform the sequence (1,2)(3,4)(5,4) is 296.9831 seconds.



(b) The time spent by the crane to perform the sequence (5,2)(3,4)(1,4) is 314.755 seconds.

Figure 7: An example showing two sequences considered as same group symmetries in the CPMP and whose crane time is not the same.

Proof. The same layout as in sol is obtained by the sequence $sol_1 = (s_1, h_1, k_1, e_1) \dots (s_{i+1}, h_{i+1}, k_i, e_i) \dots (s_n, h_n, k_n, e_n)$. Moreover the time spent by the crane to carry out sol_1 is shorter than that spent to carry out sol :

$$\begin{aligned}
time(sol) &= t(s_1, h_1, k_1, e_1) + \dots + \left(c_{k_{i-1}s_i}^0 + c_{s_i k_i}^1 + c_{k_i s_{i+1}}^0 + c_{s_{i+1} s_i}^1 \right) + \left(v_{h_i}^0 + b_{h_i} + v_{h_i}^1 \right. \\
&\quad \left. + v_{e_i}^1 + v_{e_i}^0 + v_{h_{i+1}}^0 + b_{h_{i+1}} + v_{h_{i+1}}^1 + v_{h_i}^1 + v_{h_i}^0 \right) + \dots + t(s_n, h_n, k_n, e_n) \\
&> t(s_1, h_1, k_1, e_1) + \dots + \left(c_{k_{i-1}s_i}^0 + c_{s_i s_{i+1}}^0 + c_{s_{i+1} k_i}^1 + c_{k_i s_i}^0 \right) + \left(v_{e_i}^1 + v_{e_i}^0 \right. \\
&\quad \left. + v_{h_{i+1}}^0 + b_{h_{i+1}} + v_{h_{i+1}}^1 \right) + \dots + t(s_n, h_n, k_n, e_n) \\
&> t(s_1, h_1, k_1, e_1) + \dots + \left(c_{k_{i-1}s_{i+1}}^0 + c_{s_{i+1} k_i}^1 \right) + \left(v_{e_i}^1 + v_{e_i}^0 + v_{h_{i+1}}^0 + b_{h_{i+1}} + v_{h_{i+1}}^1 \right) \\
&\quad + \dots + t(s_n, h_n, k_n, e_n) = time(sol_1)
\end{aligned}$$

The first inequality is satisfied by Proposition 6 and the fact that the times are always positive. The second inequality is satisfied because $(c_{k_{i-1}s_i}^0 + c_{s_i s_{i+1}}^0) \leq c_{k_{i-1}s_{i+1}}^0$ and $(c_{k_i s_i}^0 + c_{s_i s_{i+2}}^0) \leq c_{k_i s_{i+2}}^0$. \square

Another dominance rule from the CPMP that applies is the direct transitive rule. This rule refers to the movement of a container from one stack α to another stack β , and then from the stack

β to a different one ω in two moves rather than in a single move from stack α to ω .

Proposition 8. *A sequence $sol = (s_1, h_1, k_1, e_1) \dots (s_i, h_i, k_i, e_i)(k_i, e_i, k_{i+1}, e_{i+1}) \dots (s_n, h_n, k_n, e_n)$ is disallowed for the CPMPCT.*

Proof. Since the sequence $(s_i, h_i, k_i, e_i)(k_i, e_i, k_{i+1}, e_{i+1})$ and $(s_i, h_i, k_{i+1}, e_{i+1})$ start and end in the same stacks and slot (k_i, e_i) is only used for temporary storage of the container, the same layout as in sol is obtained by the sequence $sol_1 = (s_1, h_1, k_1, e_1) \dots (s_i, h_i, k_{i+1}, e_{i+1}) \dots (s_n, h_n, k_n, e_n)$ with a shorter crane time:

$$\begin{aligned} time(sol) &= t(s_1, h_1, k_1, e_1) + \dots + \left(c_{k_{i-1}s_i}^0 + c_{s_i k_i}^1 + c_{k_i k_i}^0 + c_{k_i k_{i+1}}^1 \right) \\ &\quad + \left(v_{h_i}^0 + b_{h_i} + v_{h_i}^1 + v_{e_i}^1 + v_{e_i}^0 + v_{e_i}^0 + b_{e_i} + v_{e_i}^1 + v_{e_{i+1}}^1 + v_{e_{i+1}}^0 \right) + \dots + t(s_n, h_n, k_n, e_n) \\ &> t(s_1, h_1, k_1, e_1) + \dots + \left(c_{k_{i-1}s_i}^0 + c_{s_i k_{i+1}}^1 + v_{h_i}^0 + b_{h_i} + v_{h_i}^1 + v_{e_{i+1}}^1 + v_{e_{i+1}}^0 \right) + \dots \\ &\quad + t(s_n, h_n, k_n, e_n) = time(sol_1) \end{aligned}$$

The inequality follows from the fact that times are always positive and from the fact that we consider crane acceleration $\left((c_{s_i k_i}^1 + c_{k_1 k_{i+1}}^1) > c_{s_i k_{i+1}}^1 \right)$. \square

5. Integer programming models

We model the CPMPCT by considering the *segments* in which a single move takes place, separated by points in which we have the layout of the bay after the move. If there is a move between points t and $t + 1$, the layout of the bays just differ in the position of the container being moved. The first point corresponds to the initial layout of the bay and the last point T to the arranged configuration after a solution of at most $T - 1$ moves. Note that T should be strictly greater than the upper bound for the number of moves required to solve the problem. Otherwise, the problem could be infeasible or return a solution for which we cannot guarantee its optimality.

Several mathematical models for the CPMP are studied in Parreño-Torres et al. (2019). We take as starting point the description of the variables of their *IPS6* model since it has the best performance. This model uses two sets of variables, one describing the layout of the bay at each period, the x variables, and another describing the movement of the container being moved at each segment. The second set can be divided into two different groups of variables: w variables that describe the initial position of the container being moved; and z variables that describe the

destination position where it will be placed. Our model, *IPCT*, uses two new groups of variables to link the origin and destination stack of each move, and the destination stack of a move and the origin stack of the next one.

$$\begin{aligned}
x_{shpt} &= \begin{cases} 1, & \text{If at time point } t \text{ there is a container in stack } s, \text{ tier } h, \text{ whose priority is } p \\ 0, & \text{Otherwise} \end{cases} \\
&\quad \forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T} \\
w_{shpt} &= \begin{cases} 1, & \text{If in the segment between } t \text{ and } t + 1, \text{ a container with priority } p \\ & \text{is moved from } (s, h); \\ 0, & \text{Otherwise;} \end{cases} \\
&\quad \forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T} \setminus \{T\}; \\
z_{shpt} &= \begin{cases} 1, & \text{If in the segment between } t \text{ and } t + 1, \text{ a container with priority } p \\ & \text{is moved to } (s, h); \\ 0, & \text{Otherwise;} \end{cases} \\
&\quad \forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T} \setminus \{T\}; \\
l_{skt} &= \begin{cases} 1, & \text{If in the segment between } t \text{ and } t + 1 \text{ there is a move from stack } s \text{ to stack } k; \\ 0, & \text{Otherwise;} \end{cases} \\
&\quad \forall s, k \in \mathcal{S} : s \neq k; \quad \forall t \in \mathcal{T} \setminus \{T\}; \\
u_{skt} &= \begin{cases} 1, & \text{If in the segment between } t \text{ and } t + 1 \text{ there is a move from stack } k \text{ and the} \\ & \text{destination stack of the previous move was stack } s; \\ 0, & \text{Otherwise;} \end{cases} \\
&\quad \forall s, k \in \mathcal{S} : s \neq k; \quad \forall t \in \mathcal{T} \setminus \{T\};
\end{aligned}$$

When a container is moved there are two vertical moves in both the origin and destination positions: one with the crane loaded and another with the crane unloaded. We therefore consider $v_h = v_h^0 + v_h^1$ to simplify notation.

The objective function of our model, *IPCT*, can be formulated as follows:

$$\begin{aligned}
\text{Min } \sum_{s=1}^S \left(\sum_{\substack{k=1 \\ k \neq s}}^S \left(c_{0s}^0 \cdot l_{sk1} \right) + \sum_{t=1}^{T-1} \left(c_{sk}^1 \cdot l_{skt} \right) + \sum_{t=2}^{T-1} \left(c_{sk}^0 \cdot u_{skt} \right) \right) \\
+ \sum_{h=1}^H \sum_{p=1}^P \sum_{t=1}^{T-1} \left((v_h + b_h) \cdot w_{shpt} + v_h \cdot z_{shpt} \right)
\end{aligned} \tag{11}$$

The upper line of equation (11) considers the time spent by the crane to perform all the horizontal moves and the bottom line the time spent by the crane to perform all the vertical moves. The first term of the upper line corresponds to the movement of the spreader from its initial position (out of the bay) to the position of the first container to be moved.

The constraints of *IPCT* model are as follows.

$$\sum_{s=1}^S \sum_{h=1}^H w_{shpt} = \sum_{s=1}^S \sum_{h=1}^H z_{shpt} \quad \forall p \in \mathcal{P}, \forall t \in \mathcal{T} \setminus \{T\} \tag{12}$$

$$\sum_{s=1}^S \sum_{h=1}^H \sum_{p=1}^P w_{shpt} \leq 1; \quad \sum_{s=1}^S \sum_{h=1}^H \sum_{p=1}^P z_{shpt} \leq 1 \quad \forall t \in \mathcal{T} \setminus \{T\} \tag{13}$$

$$\sum_{s=1}^S \sum_{h=1}^H \sum_{p=1}^P w_{shpt+1} \leq \sum_{s=1}^S \sum_{h=1}^H \sum_{p=1}^P w_{shpt} \quad \forall t \in \mathcal{T} \setminus \{T-1, T\} \tag{14}$$

$$\sum_{k=p}^P x_{sh+1kT} \leq \sum_{k=p}^P x_{shkT} \quad \forall s \in \mathcal{S}, \forall h \in \mathcal{H} \setminus \{H\}, \forall p \in \mathcal{P} \tag{15}$$

$$\sum_{p=1}^P x_{s1pt} + \sum_{p=1}^P z_{s1pt} \leq 1 \quad \forall s \in \mathcal{S}, \forall t \in \{2, \dots, T\} \tag{16}$$

$$\sum_{p=1}^P x_{sh+1pt} + \sum_{p=1}^P w_{shpt} + \sum_{p=1}^P z_{sh+1pt} \leq \sum_{p=1}^P x_{shpt} \quad \forall s \in \mathcal{S}, \forall h \in \mathcal{H} \setminus \{H\}, \forall t \in \mathcal{T} \setminus \{T\} \tag{17}$$

$$x_{shpt} + z_{shpt} = x_{shpt+1} + w_{shpt} \quad \forall s \in \mathcal{S}, \forall h \in \mathcal{H}, \forall p \in \mathcal{P}, \forall t \in \mathcal{T} \setminus \{T\} \tag{18}$$

$$\sum_{p=1}^P x_{shpt} + \sum_{p=1}^P x_{sh+1pt} + \sum_{p=1}^P w_{shpt} + \sum_{p=1}^P z_{sh+1pt} + \sum_{p=1}^P z_{shpt} \leq 2 \\
\forall s \in \mathcal{S}, \forall h \in \mathcal{H} \setminus \{H\}, \forall t \in \mathcal{T} \setminus \{T\} \tag{19}$$

$$\sum_{h=1}^H \sum_{p=1}^P z_{shpt} + \sum_{h=1}^H \sum_{p=1}^P w_{shpt+1} \leq 1 \quad \forall s \in \mathcal{S}, \forall t \in \mathcal{T} \setminus \{T-1, T\} \quad (20)$$

$$\sum_{h=1}^H w_{shpt} + \sum_{h=1}^H z_{shpt+1} \leq 1 \quad \forall s \in \mathcal{S}, \forall p \in \mathcal{P}, \forall t \in \mathcal{T} \setminus \{T-1, T\} \quad (21)$$

$$\sum_{h=1}^H \sum_{p=1}^P (w_{shpt} + z_{khpt}) \leq 1 + l_{skt} \quad \forall s, k \in \mathcal{S} : s \neq k, \forall t \in \mathcal{T} \setminus \{T\} \quad (22)$$

$$\sum_{h=1}^H \sum_{p=1}^P (z_{shpt-1} + w_{khpt}) \leq 1 + u_{skt} \quad \forall s, k \in \mathcal{S} : s \neq k, \forall t \in \mathcal{T} \setminus \{1, T\} \quad (23)$$

$$w_{sh1(T-1)} = 0 \quad z_{sh1(T-1)} = 0 \quad \forall s \in \mathcal{S}, \forall h \in \mathcal{H} \quad (24)$$

$$w_{s1p(T-1)} = 0 \quad \forall s \in \mathcal{S}, \forall p \in \mathcal{P} \setminus \{1\} \quad (25)$$

$$x_{shpt}, w_{shpt}, z_{shpt} \in \{0, 1\} \quad \forall s \in \mathcal{S}, \forall h \in \mathcal{H}, \forall p \in \mathcal{P}, \forall t \in \mathcal{T} \setminus \{T\} \quad (26)$$

$$l_{skt}, u_{skt} \in \{0, 1\} \quad \forall s, k \in \mathcal{S}, \forall t \in \mathcal{T} \setminus \{T\} \quad (27)$$

Constraints (12) force the number of containers of each priority remains constant throughout the moves. At most one move can be made in each segment by constraints (13). Moves are placed in the earliest time segments by constraints (14), i.e., if there has been a move in the segment between t and $t + 1$ points, another move must have been made between $t - 1$ and t , otherwise the former move could have been made earlier. In this way, we force the bay to be ordered at the last point T , constraints (15). Constraints (16) and (17) ensure that each slot can place at most one container. Moreover, constraints (16) are strengthened so that if one slot of the bottom tier is occupied, it cannot receive another container. Constraints (17) also have two other effects: when a slot is occupied, the slots below it must also be occupied; and only the highest containers can move. Constraints (18) say that if a container does not move, it is still present in the layout of the bay of the next point. If a container moves in the segment between t and $t + 1$, it will be in its initial position in the layout of point t , but not in the layout of point $t + 1$. Analogously, the position to which a container moves cannot be occupied in the layout of point t since it will be occupied in $t + 1$. Constraints (19) strengthen our formulation by limiting the moves that can be made when we consider two consecutive tiers of the same stack. Constraints (20) avoid direct transitive moves and (21) the movement of the same container in two successive moves. Constraints (20) and (21) are not used on instances where all containers have different priority. Finally, constraints (22) and

Algorithm 1 Branch and bound algorithm pseudocode.

```
1: function CPMPCT-BB(bay)
2:   lb, lbc  $\leftarrow$  LOWERBOUND(bay)
3:    $\pi \leftarrow$  HEURISTIC(bay) ▷ Feasible solution
4:   if  $\pi = \emptyset$  then u  $\leftarrow \infty$ 
5:   else
6:     u  $\leftarrow$  UPPER(bay,  $\pi$ ) ▷ Upper bound for the number of moves
7:   l  $\leftarrow lb$ 
8:   while l  $\leq$  u do
9:     DFS(bay,  $\pi$ , l, u)
10:    if l = moves( $\pi$ ) + 1 and l  $\neq$  u then l = u
11:    else
12:      l  $\leftarrow$  l + 1
13:  return  $\pi$ 
```

(23) establish between which stacks the movements are made. Constraints (22) connect the stacks between which a move of a container occurs and (23) connect the destination stack of a move with the origin stack of the next container to be moved.

6. Branch and Bound algorithm

6.1. Structure of the search tree

We propose a branch and bound algorithm whose search strategy is to repeatedly run a limited depth version of the depth first search (DFS) with increasing depth limits, see Algorithm 1 for its pseudocode. It is known in computer science as iterative deepening search and combines depth-first search with breadth-first search (BFS). Given the initial configuration of the bay, our approach computes the lower bound for the number of moves, lb , and run the DFS until a depth $l = lb$. Unlike in the CPMP, even if it finds a solution with l moves at level l , the algorithm increases the depth by one and the DFS is run again, because it is possible to obtain solutions with a lower crane time with a higher depth limit. Every time a better solution π is found, the upper bound for the number of moves to solve the CPMPCT, u , is updated. Note that the upper bound u is computed according to Proposition 2 taking into account the disposition of the initial layout of the bay (root node). The algorithm stops when $l = u$, guaranteeing that the solution obtained is optimal. Once the algorithm reaches a solution with $moves(\pi) = l$, if the DFS with the depth limit increased by one does not improve that solution, it directly updates $l = u$ and explores the whole tree.

All possible moves are explored at each level, pruning branches that satisfy at least one of

these criteria: (i) Proposition 8 is satisfied, (ii) a lower bound for the number of moves is greater than the current depth limit, (iii) the lower bound for the crane time lbc , using LB_{ct}^1 , exceeds the crane time of the best solution obtained so far. If LB_{ct}^1 fails, we use LB_{ct}^2 , but as its computational cost is higher, we only compute it if considering that the blocking containers move to the bottom tier the branch could be pruned.

Our algorithm first explores branches with the lowest lower bound for the number of moves, using as a tie breaking criterion the lowest lower bound for the crane time LB_{ct}^1 . During the search, a heuristic tries to repair unfeasible solutions into a feasible solution for the problem.

6.2. A heuristic algorithm for repairing non-feasible solutions

The heuristic makes BG² and GG moves such as those proposed in Tanaka and Tierney (2018), but also GB moves, stopping when the partial solution obtained is worse than the best current solution. (see Algorithm 2 for the pseudocode). It first performs BG moves, selecting the best move (BEST) according to the following order of preference:

1. The move with the smallest difference between the container at the top of the destination stack and the container moved.
2. The container with the highest priority.

When no further BG moves are possible, the algorithm performs GG moves selecting them in order according to the following list:

1. The move with the largest difference between the new top of the origin stack and the top of the destination stack before the movement. We only consider the moves in which the difference is at least one.
2. The move with the largest priority of the new top of the origin stack.
3. The move with the fewest number of containers in the origin stack.

As soon as the bay is ordered, the heuristic ends and updates the best solution obtained so far, π , with the new solution reached if the crane time is lower. Otherwise, as long as the number of moves that have been made is lower than the upper bound for the number of moves, u , and the partial solution does not exceed the crane time of the best solution, GB moves are carried out.

²According to Bortfeldt and Forster (2012), a *bad-good* (BG) move is a move in which a blocking container is moved to a stack where it is no longer blocking. A *good-good* (GG) move is a move in which a non blocking container is moved to a stack where it is also well placed. A *good-bad* (GB) move is a move in which a well placed container is moved to a stack where it blocks the retrieval of other containers.

Algorithm 2 Heuristic pseudocode.

```

1: function HEURISTIC(bay,  $\pi'$ ,  $\pi$ )
2:    $M \leftarrow \pi'$  ▷ Partial solution
3:    $B \leftarrow \{(s, d) \in S \times S\}$ 
4:   while  $\text{moves}(M) < u$  and  $\text{time}(M) + \text{cmin} < \text{time}(\pi)$  do
5:     ▷  $u$  is the upper bound for the number of moves and  $\text{cmin}$  the minimum cost of performing a move
6:      $m \leftarrow \emptyset$ 
7:      $m \leftarrow \text{BEST}(B \cap BG)$ 
8:     if  $m = \emptyset$  then  $m \leftarrow \text{BEST}(B \cap GG)$ 
9:     if  $m = \emptyset$  then
10:       $\text{flag} \leftarrow \text{TRY\_GB}(M, \text{bay})$ 
11:     else
12:       $M \leftarrow \text{APPLY}(\text{bay}, M)$  ▷ Make the move
13:      if  $\text{BLOCKS}(M) = 0$  and  $\text{time}(M) < \text{time}(\pi)$  then
14:         $\pi \leftarrow M$  ▷ Update the best solution so far
15:        return
16:      if  $m = \emptyset$  and  $\text{flag} = \text{FALSE}$  then
17:        return
18:   return

```

The idea of the TRY_GB function is to empty a stack that has no blocking containers, placing its containers on stacks where there are also no blocking containers. The stack with the smallest number of containers is always selected to be emptied and its containers are moved to the second stack with the smallest number until it is filled, then to the third stack and so on. Since the containers moved through GB moves will be placed upside down, in the following iterations the empty column will be filled up employing BG moves. If a feasible solution is not obtained, the function returns FALSE. This function is just used if there is at most one blocking container per stack. Refer to Figure 8 for a example in which a feasible solution is obtained by performing GB moves.

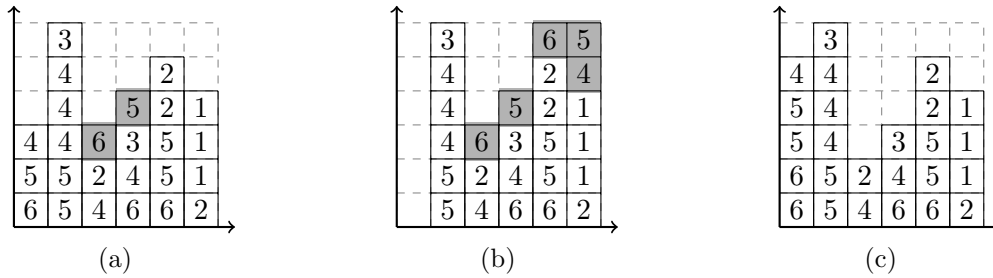


Figure 8: Figure (a) shows a layout in which neither BG nor GG moves satisfying the heuristic criteria can be performed. Figure (b) shows the layout after emptying one stack. Finally, the bay is arranged by performing only BG moves as shown in Figure (c).

7. Computational experiments

We conduct an extensive computational analysis to test the performance of the mathematical model and the branch and bound algorithm proposed. We refer them as *IPCT* and *CTA*, respectively. Both were coded in C/C++ and executed on virtual machines with 4 virtual processors and 16 GBytes of RAM running Windows 10 Enterprise 64 bits. The 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. We fix the time limit to be 3600 seconds and use CPLEX 12.7 with 4 threads as a solver.

We assess our approaches using these four well-known datasets from the scientific literature:

EMM dataset. The instances from Expósito-Izquierdo et al. (2012). In total, 450 instances with 4 tiers and with the number of stacks varying among 4, 7 or 10 stacks.

ZJY dataset. Instances from Zhang et al. (2015) that are grouped into 5 categories of 20 instances each, with 6, 7, 8 or 9 stacks and 4 tiers, and 6 stacks and 5 tiers.

BZ dataset. This dataset from van Brink and van der Zwaan (2014) contains instances with 4 and 5 tiers and a number of stacks ranging in size from 3 to 9, with a total of 960 instances.

CV dataset. Instances from Caserta and Voß (2009) range in size from 3 stacks and 5 tiers up to 10 stacks and 10 tiers, totalling 760 instances. These instances are among the most difficult in the pre-marshalling literature because all containers have different priorities and all stacks are filled to the same tier with two empty tiers on top of each stack.

7.1. Performance of the mathematical model *IPCT*

Since *IPCT* requires an initial feasible solution to set the T value, we assess its performance using the instances from EMM, ZJY, BZ, and CV datasets solved by *IPS6* in Parreño-Torres et al. (2019). Thus, the T used to solve *IPCT* is obtained from the solution provided by *IPS6* according to Proposition 2. Mathematical models for the CPMP are flexible and easier to implement, but fail to solve the largest CV instances sizes. Therefore, only the CV instances ranging in size from 3 stacks and 5 tiers up to 7 stacks and 6 tiers, are considered.

We focus here on evaluating the performance of our model and on seeing if the crane time of the solutions obtained really differ from that of the solutions obtained with *IPS6* (which solves the classic CPMP). Table 3 shows the results obtained for the four datasets grouping the instances by number of tiers (H) and number of stacks (S). Column “#Inst.” gives the total number of instances tested at each group and columns “#Opt.” and “#Feas.” represent the number of optimal and feasible solutions obtained by *IPCT*, respectively. The “Crane Time” columns represent the average time spent by the crane on those instances and the “CPU(s)” columns the average running time (in seconds) spent by each model. Even though the objective function of *IPS6* does not minimize the crane time, given a solution for the model it is easy to compute its crane time. Column “AVRPD” shows the average of the relative percentage deviation in the instances optimally solved, i.e., the average (%) of the differences between the crane time obtained with *IPS6* and the crane time obtained with *IPCT* divided by the crane time obtained with *IPS6*. Finally, column “MAXRPD” represents the maximum relative percentage deviation again in the instances solved to optimality.

In the light of the results, *IPCT* solves to optimality 87.98%, 85.33%, 43.21%, and 67.48% of the instances belonging to EMM, ZJY, BZ, and CV datasets. These percentages increase significantly to 97.95%, 87.65%, 97.76%, and 83.43% if we consider the instances in which the model obtains a feasible solution. The average crane time of the solutions reached with *IPS6* do not correspond in any case to that of the optimal solution obtained by *IPCT* for the CPMPCT and furthermore the relative deviation can be up to 24%.

We focus now on the BZ dataset, groups with 7 and 9 stacks, and 4 and 6 tiers. The crane time decreases on average between 4.7% and 7.7% and between 13.83% and 24% in instances that present the best improvement. These cases represent fairly common bay sizes in terminals around the world. The practical implications are huge. The presented results indicate that minimizing crane time results in much shorter pre-marshalling operations, effectively saving on expensive hourly wages of stevedores, crane operations, etc.

With respect to the running times, the highest are obtained in the datasets in which the model solves a lower percentage of instances, CV and ZJY datasets. In these datasets the average times are 486.68 and 683.26 seconds. On the other hand, BZ dataset has the shortest average running time, 129.77 seconds and is followed by EMM dataset with an average of 248.23 seconds.

Table 3: Performance of *IPCT* model on the instances optimally solved by *IPS6* from EMM, ZJY, BZ, and CV dataset.

Dataset	H	S	IPCT					IPS6	IPCT vs. IPS6	
			#Inst	#Opt	#Feas.	Avg.		Crane Time	AVRPD	MAXRPD
						Crane Time	CPU(s)			
EMM	4	4	150	150	150	472.93	251.90	477.46	1.13	21.29
	4	7	126	115	124	605.08	185.89	630.02	3.75	14.66
	4	10	115	79	109	670.03	332.02	715.00	6.28	15.30
		Total	391	344	383	562.37	248.23	583.01	3.19	21.29
ZJY	4	6	20	10	17	878.60	740.87	912.82	3.48	9.41
	4	7	17	11	16	987.85	533.23	1012.90	2.62	8.91
	4	8	18	9	17	794.97	839.89	813.80	2.61	5.54
	4	9	17	4	13	846.07	265.92	880.24	3.90	4.45
	5	6	9	1	8	1261.70	2017.25	1263.11	0.11	0.11
	Total	81	35	71	898.66	683.26	925.10	2.94	9.41	
BZ	4	3	120	120	120	374.34	0.62	376.93	0.54	7.68
	4	5	120	120	120	444.76	5.00	457.37	2.61	14.15
	4	7	120	117	120	535.70	100.70	562.20	4.74	18.51
	4	9	96	98	119	600.71	200.94	644.29	6.96	24.03
	6	3	120	116	119	917.30	43.50	924.27	0.79	9.09
	6	5	120	95	100	1023.02	237.54	1054.03	2.87	18.69
	6	7	104	64	92	1050.33	391.55	1111.04	5.42	13.83
	6	9	93	32	83	1018.88	439.58	1106.15	7.69	15.10
	Total	893	762	873	686.69	129.77	712.45	3.38	24.03	
CV	5	3	40	40	40	984.27	201.16	994.09	0.91	4.24
	5	4	38	34	37	973.75	342.69	992.59	2.12	12.66
	5	5	32	21	29	987.73	817.42	1018.80	3.15	7.91
	5	6	23	8	15	956.86	1057.91	985.91	2.76	5.34
	5	7	11	1	5	940.13	868.36	1010.72	6.98	6.98
	5	8	7	0	2	-	-	-	-	-
	6	4	9	5	7	1170.06	861.35	1179.32	0.79	1.74
	6	5	2	1	1	1360.83	3032.89	1391.82	2.23	2.23
	6	6	1	0	0	-	-	-	-	-
	Total	163	110	136	991.15	486.68	1009.93	1.91	12.66	

7.2. Comparing the mathematical model *IPCT* with the branch and bound algorithm *CTA*

We compare the performance of *IPCT* and that of *CTA* over the set of instances tested in the previous section. Table 4 shows the results obtained. It provides the number of optimal and feasible solutions reached by *IPCT* and *CTA*, as well as the average running times for the instances in which both of them reached the optimal solution.

The algorithm *CTA* obtains a feasible solution in all of the instances whereas *IPCT* did not obtain any in a few of the cases. Regarding to the number of optimal solutions, *CTA* obtains 96 more optimal solutions than *IPCT* over the 1528 instances tested, or 6.3% more. Although *CTA* solves 3.6%, 45.7%, and 31.9% more than *IPCT* on EMM, ZJY, and CV datasets, on BZ it solves 0.7% less than *IPCT* due to the effect of the largest instances in this dataset.

Considering the instances optimally solved by both of them (column “#Both”), it is remarkable

Table 4: Comparing *CTA* vs. *IPCT* in the EMM, ZJY, BZ and CV instances used to evaluate the performance of the mathematical model *IPCT* in the previous section.

Dataset	H	S	#Inst.	#Opt.		#Feas.		#Both	Avg.	Avg. CPU(s)	
				IPCT	CTA	IPCT	CTA		Crane Time	IPCT	CTA
EMM	4	4	150	150	150	150	150	150	472.93	251.90	0.27
	4	7	126	115	125	124	126	115	605.08	185.89	0.79
	4	10	115	79	83	109	115	76	654.80	294.54	107.00
		Total	391	344	358	383	391	341	558.03	239.14	24.23
ZJY	4	6	20	10	20	17	20	10	878.60	740.87	0.41
	4	7	17	11	17	16	17	11	987.85	533.23	1.32
	4	8	18	9	17	17	18	9	794.97	839.89	16.90
	4	9	17	4	9	13	17	4	846.07	265.92	18.27
	5	6	9	1	9	8	9	1	1261.70	2017.25	6.26
	Total	81	35	72	71	81	35	898.66	683.26	7.14	
BZ	4	3	120	120	120	120	120	120	374.34	0.62	0.01
	4	5	120	120	120	120	120	120	444.76	5.00	0.02
	4	7	120	117	119	120	120	117	535.70	100.70	4.85
	4	9	120	98	98	119	120	96	590.17	135.02	65.75
	6	3	120	116	120	119	120	116	917.30	43.50	0.02
	6	5	104	95	102	100	104	94	1014.21	211.95	19.80
	6	7	96	64	56	92	96	53	962.17	125.93	201.11
	6	9	93	32	21	83	93	21	918.22	267.80	502.67
		Total	893	762	756	873	893	737	664.38	85.06	40.65
CV	5	3	40	40	40	40	40	40	984.27	201.16	0.01
	5	4	38	34	38	37	38	34	973.75	342.69	0.03
	5	5	32	21	32	29	32	21	987.73	817.42	0.15
	5	6	23	8	23	15	23	8	956.86	1057.91	0.54
	5	7	11	1	11	5	11	1	940.13	868.36	0.52
	5	8	7	0	6	2	7	0	-	-	-
	6	4	9	5	9	7	9	5	1170.06	861.35	0.07
	6	5	2	1	2	1	2	1	1360.83	3032.89	1.13
	6	6	1	0	1	0	1	0	-	-	-
		Total	163	110	162	136	163	110	991.15	486.68	0.10

that the average running times are cut down by *CTA* between 52.20% on the BZ dataset and 99.98% on the CV dataset. We further note that in some groups such as those with 5 tiers and 4 or 5 stacks, the average running times greatly decrease from 342.69 seconds to 0.03 seconds and from 817.42 seconds to 0.15 seconds, though in the largest groups of BZ, with 6 tiers and 7 or 9 stacks, the situation reverses.

7.3. Performance of branch and bound algorithm *CTA*

We evaluate our algorithm on the instances of EMM, ZJY and BZ datasets, and also on all the instances of CV dataset. The results are shown in Table 5 and 6. For the instances optimally solved by *CTA*, they show the average crane times and the average running times. For the instances in which a feasible solution is found, the average crane times and the average running times required to get the best solution.

In Table 5, a solution is found for all the instances and in a running time that on average does not exceed 180 seconds for any of the datasets. In addition, an optimal solution is found for around 80% of the instances in the three datasets.

Table 5: Performance of *CTA* on EMM, ZJY, and BZ datasets.

Dataset	H	S	#Inst	#Opt.	Avg. Opt.		#Feas.	Avg. Feas.	
					Crane Time	CPU(s)		Crane Time	CPU(s)
EMM	4	4	150	150	472.93	0.27	150	472.93	0.27
	4	7	150	125	657.64	27.13	150	905.64	87.25
	4	10	150	83	692.36	197.32	150	1222.22	395.29
		Total	450	358	588.30	55.33	450	866.93	160.94
ZJY	4	6	20	20	1053.42	16.92	20	1053.42	16.92
	4	7	20	18	1071.74	23.39	20	1122.24	21.24
	4	8	20	17	976.58	238.74	20	1053.99	202.94
	4	9	20	10	1078.95	585.43	20	1266.02	295.57
	5	6	20	15	1448.46	488.19	20	1582.43	366.21
		Total	100	80	1118.48	224.94	100	1215.62	180.58
BZ	4	3	120	120	374.34	0.01	120	374.34	0.01
	4	5	120	120	444.76	0.02	120	444.76	0.02
	4	7	120	119	546.53	12.55	120	553.47	12.50
	4	9	120	98	600.48	94.52	120	724.87	128.94
	5	3	120	120	949.12	0.03	120	949.12	0.03
	5	5	120	105	1102.35	47.29	120	1233.84	41.40
	5	7	120	56	996.30	265.96	120	1528.31	265.03
	5	9	120	21	918.22	502.67	120	1932.41	814.76
		Total	960	759	694.19	54.25	960	967.64	157.84

We now focus on Table 6, which contains the results for the CV dataset. The algorithm solves 654 out of the 760 instances, but just 245 to optimality. All the instances with 5 and 6 tiers are solved, and 249 of the 280 instances with 7 tiers. The worst performance is presented in the instance with 8 tiers.

8. Conclusions

We propose the pre-marshalling problem considering crane time minimization objective (CPM-PCT). Even though the pre-marshalling problem has been widely studied in the literature in the last years due to its relationship with the increase in productivity of container terminals in an extremely competitive environment, all previous papers minimize the number of moves as their objective function to rearrange the bay. In this paper, crane times have been calculated in a precise way, considering different speeds and accelerations depending on whether the crane is moving with load or not, as well as twistlock times. We show that the number of moves is not a suitable indicator for measuring crane time.

Table 6: Performance of *CTA* on CV dataset.

H	S	#Inst	#Opt.	Avg. Opt.		#Feas.	Avg. Feas.	
				Crane Time	CPU(s)		Crane Time	CPU(s)
5	3	40	40	984.27	0.01	40	984.27	0.01
5	4	40	40	1043.19	0.11	40	1043.19	0.11
5	5	40	40	1187.69	37.69	40	1187.69	37.69
5	6	40	35	1270.00	204.94	40	1338.96	179.33
5	7	40	25	1374.62	610.68	40	1531.04	384.34
5	8	40	11	1326.97	985.54	40	1633.31	273.04
6	4	40	35	1830.44	61.12	40	1897.71	53.76
6	5	40	9	1789.99	78.93	40	2218.00	18.54
6	6	40	3	1600.40	921.00	40	2405.58	76.17
6	7	40	0	-	-	40	2765.90	181.16
7	4	40	6	2221.35	178.50	40	2933.43	597.13
7	5	40	1	1876.20	434.54	38	3208.36	373.53
7	6	40	0	-	-	37	3925.59	1288.49
7	7	40	0	-	-	36	4282.63	1327.28
7	8	40	0	-	-	36	4973.35	1780.95
7	9	40	0	-	-	32	5588.67	1596.88
7	10	40	0	-	-	30	5864.61	1758.06
8	6	40	0	-	-	2	5252.07	1346.71
8	10	40	0	-	-	3	9134.43	1632.72
Total		760	245	1315.10	171.07	654	2737.96	546.25

We present a novel dominance criteria to solve the CPMPCT, upper bounds for the number of movements required to solve the CPMPCT, and lower bounds for the problem. Two exact approaches are proposed: a mathematical model and a branch and bound algorithm. An extended computational analysis highlight the need to consider the new objective function as opposed to the one used so far, cutting the crane time down by 24% in some cases. The branch and bound algorithm outperforms the mathematical models, obtaining a solution in 95% of the instances tested.

We consider the crane time but the approaches developed in the paper could be used for other goals such as minimizing energy consumption during the arrangement. Further lines of research will develop new lower bounds for the problem and also integrate the problem into a robust framework that could prevent the need for several iterations of the CPMPCT due to last minute changes in container retrieval orders from delays.

Acknowledgements

This work has been partially supported by the Spanish Ministry of Science, Innovation, and Universities, FPU Grant A-2015-12849 and under the project “OPTEP-Port Terminal Operations Optimization” (No. RTI2018-094940-B-I00) financed with FEDER funds.

References

- 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., 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.
- da Silva Firmino, A., de Abreu Silva, R., Times, V., 2019. A reactive GRASP metaheuristic for the container retrieval problem to reduce cranes working time. *Journal of Heuristics* 25 (2), 141 – 173.
- 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* 276 (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.
- Hottung, A., Tanaka, S., Tierney, K., 2020. Deep learning assisted heuristic tree search for the container pre-marshalling problem. *Computers & Operations Research* 113, 104781.
- Hottung, A., Tierney, K., 2016. A biased random-key genetic algorithm for the container pre-marshalling problem. *Computers & Operations Research* 75, 83 – 102.
- Jovanovic, R., Tuba, M., Voß, S., 2017. A multi-heuristic approach for solving the pre-marshalling problem. *Central European Journal of Operations Research* 25, 1–28.
- Jovanovic, R., Tuba, M., Voß, S., 2019. An efficient ant colony optimization algorithm for the blocks relocation problem. *European Journal of Operational Research* 274 (1), 78–90.
- 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 & Operations Research* 34 (11), 3295–3313.
- Lee, Y., Lee, Y., 2010. A heuristic for retrieving containers from a yard. *Computers & Operations Research* 37 (6), 1139–1147.
- Lin, D.-Y., Lee, Y.-J., Lee, Y., 2015. The container retrieval problem with respect to relocation. *Transportation Research Part C* 52, 132–143.
- Parreño-Torres, C., Alvarez-Valdes, R., Ruiz, R., 2019. Integer programming models for the pre-marshalling problem. *European Journal of Operational Research* 274 (1), 142–154.
- Prandtstetter, M., 2013. A dynamic programming based branch-and-bound algorithm for the container pre-marshalling problem. Tech. rep., Technical report, AIT Austrian Institute of Technology.
- 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.
- Tanaka, S., Tierney, K., Parreño-Torres, C., Alvarez-Valdes, R., Ruiz, R., 2019. A branch and bound approach for large pre-marshalling problems. *European Journal of Operational Research* 278 (1), 211–225.
- Tierney, K., Pacino, D., Voß, S., 2017. Solving the pre-marshalling problem to optimality with A* and IDA*. *Flexible*

- Services and Manufacturing Journal 29 (2), 223–259.
- UNCTAD, 2018. United Nations Conference on Trade and Development (UNCTAD) Review of Maritime Transport. United Nations.
- van Brink, M., van der Zwaan, R., 2014. A branch and price procedure for the container premarshalling problem. In: Schulz, A., Wagner, D. (Eds.), Algorithms – ESA 2014. Vol. 8737 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, 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.
- Wilmsmeier, G., Spengler, T., 2016. Energy consumption and container terminal efficiency. Bulletin FAL 6 (350), 1–10.
- 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.

Appendix A. Proof of Proposition 6

Proof. Let $t1$ and $t2$ be:

$$t1 = c_{s_i k_i}^1 + c_{k_i s_{i+1}}^0 + c_{s_{i+1} s_i}^1 = t^{rl}(\text{distance}^r(s_i, k_i)) + t^{ru}(\text{distance}^r(k_i, s_{i+1})) + t^{rl}(\text{distance}^r(s_{i+1}, s_i))$$

$$t2 = c_{s_i s_{i+1}}^0 + c_{s_{i+1} k_i}^1 + c_{k_i s_i}^0 = t^{rl}(\text{distance}^r(s_i, s_{i+1})) + t^{ru}(\text{distance}^r(s_{i+1}, k_i)) + t^{rl}(\text{distance}^r(k_i, s_i))$$

To address this proof, we rewrite equation (2) as a function of the distance travelled x , the maximum speed reached $vmax^{r\beta}$, and the distance to achieve the maximum travelling speed d^r . Note that acceleration relies on the last two factors above.

$$t^{r\beta}(x) = \begin{cases} \frac{2\sqrt{2d^r x}}{vmax^{r\beta}}, & \text{If } x < 2d^\alpha \\ \frac{x+2d^r}{vmax^{r\beta}}, & \text{If } x \geq 2d^\alpha \end{cases} \quad \forall \beta \in \{l, u\}$$

Therefore, $t1$ and $t2$ depend on the distances among s_i , s_{i+1} , and k_i , which depend on their relative position in the bay. If we sort 3 stacks in non-decreasing order, we get six cases. Nevertheless we just have to consider three of them because $\text{distance}^r(s, k) = \text{distance}^r(k, s)$:

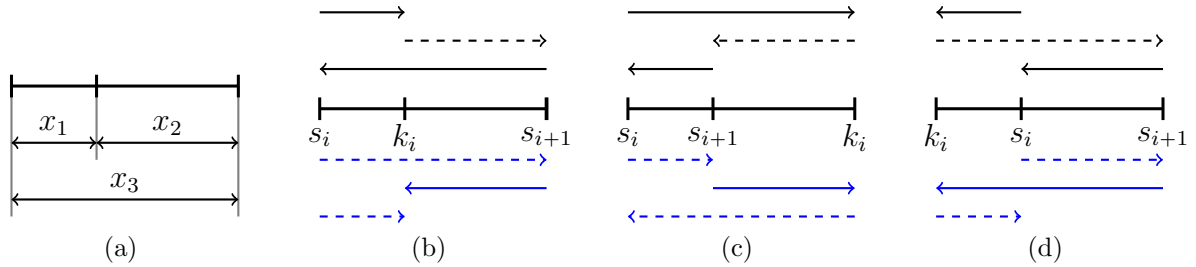


Figure A.9: Given three stacks at distances x_1 , x_2 , and x_3 (Figure A.9a), Figures A.9b to A.9d show the different relative positions that must be addressed.

Case 1. $s_i < k_i < s_{i+1}$, Figure A.9b.

Case 2. $s_i < s_{i+1} < k_i$, Figure A.9c.

Case 3. $k_i < s_i < s_{i+1}$, Figure A.9d.

In Figures A.9b to A.9d, the lines above and below the segment describe the relative position between the stacks. Solid lines represent the movements of the crane loaded and dashed lines show the crane unloaded. The time spent to perform the movements represented above the segment is $t1$ and below the segment is $t2$.

The difference $t1-t2$ can be described as a function of three terms $t1-t2 = (\text{term}_1 \text{term}_3) / \text{term}_2$, where $\text{term}_1 = (vmax^{ru} - vmax^{rl})$, $\text{term}_2 = vmax^{ru}vmax^{rl}$, and term_3 changes according to

the case discussed. We study the different cases, proving that $t1 - t2 > 0 \leftrightarrow t1 > t2$. Since $vmax^{ru} > vmax^{rl}$ and the speeds are positive values, $term_1$ and $term_2$ are greater than 0, thus we just need to prove that $term_3$ is greater than (or equal to) 0.

Case 1. $s_i < k_i < s_{i+1}$, Figure A.9b.

We consider $x_1 = distance^r(s_i, k_i)$, $x_2 = distance^r(k_i, s_{i+1})$, and $x_3 = distance^r(s_i, s_{i+1})$, so $t1 = t^{rl}(x_1) + t^{ru}(x_2) + t^{rl}(x_3)$ and $t2 = t^{ru}(x_3) + t^{rl}(x_2) + t^{ru}(x_1)$. Five scenarios should be discussed:

S1: $x_1, x_2, x_3 < 2d^r$

$$term_3 = 2\sqrt{2d^r x_1} + 2\sqrt{2d^r x_3} - 2\sqrt{2d^r x_2} > 2\sqrt{2d^r x_1} > 0$$

The first inequality is satisfied because $x_3 > x_2$, so $2\sqrt{2d^r x_3} > 2\sqrt{2d^r x_2}$ and the second one because the distance x_1 is positive ($s_i \neq k_i$).

S2: $x_1, x_2 < 2d^r$ and $x_3 \geq 2d^r$

$$term_3 = 2\sqrt{2d^r x_1} + x_3 + 2d^r - 2\sqrt{2d^r x_2} > 2\sqrt{2d^r x_1} + x_3 - 2d^r > 2\sqrt{2d^r x_1} > 0$$

Since $x_2 < 2d^r$, then $2d^r x_2 < 4(d^r)^2 \rightarrow 2\sqrt{2d^r x_2} < 4d^r$ and the first inequality is satisfied.

The second and third one follows from the fact that $x_3 \geq 2d^r$ and that $x_1 > 0$.

S3: $x_1 < 2d^r$ and $x_2, x_3 \geq 2d^r$

Distance x_3 is greater than x_2 and $x_1 > 0$, so $term_3 = (2\sqrt{2d^r x_1} + x_3 - x_2) > 2\sqrt{2d^r x_1} > 0$

S4: $x_2 < 2d^r$ and $x_1, x_3 \geq 2d^r$

As $x_2 < 2d^r$, then $2\sqrt{2d^r x_2} < 4d^r$. Moreover the distances are positive because $s_i \neq k_i \neq s_{i+1}$, thus $term_3 = (x_1 + x_3 + 4d^r - 2\sqrt{2d^r x_2}) > x_1 + x_3 > 0$.

S5: $x_1, x_2, x_3 \geq 2d^r$

The distance x_3 is greater than x_2 and all distances are greater than 0, so the inequality $term_3 = (x_1 + x_3 + 2d^r - x_2) > 0$ is fulfilled.

Therefore, given three stacks s_i, k_i, s_{i+1} such as $s_i < k_i < s_{i+1}$, the inequality $t1 > t2$ is always satisfied.

Case 2. $s_i < s_{i+1} < k_i$, Figure A.9c. Analogous proof to that in the previous case.

Case 3. $k_i < s_i < s_{i+1}$, Figure A.9d.

In this case, $x_1 = distance^r(s_i, k_i)$, $x_2 = distance^r(s_i, s_{i+1})$, and $x_3 = distance^r(k_i, s_{i+1})$,

therefore $t1 = t^{rl}(x_1) + t^{ru}(x_3) + t^{rl}(x_2)$ and $t2 = t^{ru}(x_2) + t^{rl}(x_3) + t^{ru}(x_1)$.

S1: $x_1, x_2, x_3 < 2d^r$

Since $x_3 = x_1 + x_2$,

$$\begin{aligned} term_3 &= 2\sqrt{2d^r x_1} + 2\sqrt{2d^r x_2} - 2\sqrt{2d^r x_3} = 2\sqrt{2d^r}(\sqrt{x_1} + \sqrt{x_2} - \sqrt{x_3}) \\ &= 2\sqrt{2d^r}(\sqrt{x_1} + \sqrt{x_2} - \sqrt{x_1 + x_2}) \end{aligned}$$

The term $2\sqrt{2d^r}$ is greater than 0 and $\sqrt{x_1} + \sqrt{x_2} \geq \sqrt{x_1 + x_2}$. Therefore $term_3 > 0$.

S2: $x_1, x_2 < 2d^r$ and $x_3 \geq 2d^r$

Since $2d^r > x_1 \rightarrow \sqrt{2d^r x_1} > x_1$, $2d^r > x_2 \rightarrow \sqrt{2d^r x_2} > x_2$, $x_3 = x_1 + x_2$, and $x_3 \geq 2d^r$, the following inequalities are satisfied:

$$term_3 = 2\sqrt{2d^r x_1} + 2\sqrt{2d^r x_2} - x_3 - 2d^r > 2(x_1 + x_2) - x_3 - 2d^r = x_3 - 2d^r \geq 0$$

S3: $x_1 < 2d^r$ and $x_2, x_3 \geq 2d^r$

Since $x_3 = x_1 + x_2$, and $2d^r > x_1 \rightarrow \sqrt{2d^r x_1} > x_1$,

$$term_3 = (2\sqrt{2d^r x_1} + x_2 - x_3) = (2\sqrt{2d^r x_1} - x_1) > 0.$$

S4: $x_2 < 2d^r$ and $x_1, x_3 \geq 2d^r$. Proof analogous to the previous scenario.

S5: $x_1, x_2, x_3 \geq 2d^r$

The last scenario follows from $x_3 = x_1 + x_2$ and $2d^r > 0$, then $term_3 = x_1 + x_2 + 2d^r - x_3 = 2d^r > 0$.

Therefore, given three stacks s_i, k_i, s_{i+1} such as $k_i < s_i < s_{i+1}$, the inequality $t1 > t2$ is always satisfied.

□