Transportation Research Part E 137 (2020) 101917

Contents lists available at ScienceDirect

Transportation Research Part E

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

Minimizing crane times in pre-marshalling problems

Consuelo Parreño-Torres[a,*], Ramon Alvarez-Valdes[a], Rubén Ruiz[b], Kevin Tierney[c]

[a] Department of Statistics and Operations Research, Valencia University, Doctor Moliner 50, Burjassot 46100, Valencia, Spain
[b] 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
[c] Decision and Operation Technologies Group, Bielefeld University, Universitätsstraße 25, D-33615 Bielefeld, Germany

ARTICLE INFO

Keywords:
Logistics
Container pre-marshalling
Crane time
Maritime transport
Terminal operations

ABSTRACT

The pre-marshalling problem has been extensively studied in recent years with the aim of minimizing the number of movements needed to rearrange a bay of containers. Time is a more realistic objective for measuring process efficiency, and we show that it does not correlate with the number of movements. As a result, 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 optimal solutions for problems of practical size.

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. According to UNCTAD (2018), container ports handled 752.2 million 20-foot equivalent units (TEUs) in 2017, and global container port throughput rose by 6%, three times the rate of 2016. The rise in the volume of cargo transported worldwide and the increase in the size of ships further add to the challenges facing this sector. In fact, 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. In this scenario, 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 (UNCTAD, 2018).

The container yard is a large space in which 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. Fig. 1 shows the top view of a container yard scheme with 2 blocks, 5 bays per block, and 5 stacks per bay.

Pre-marshalling contributes to the proper arrangement of the yard prior to the arrival of ships. When the workload in a terminal is low, bays are rearranged so that containers leaving the stacks early do not block containers leaving later. This process accelerates service times once the ship arrives. The classic objective of the container pre-marshalling problem (CPMP) is to minimize the number

* Corresponding author.

https://doi.org/10.1016/j.tre.2020.101917
Received 24 September 2019; Received in revised form 24 February 2020; Accepted 8 March 2020

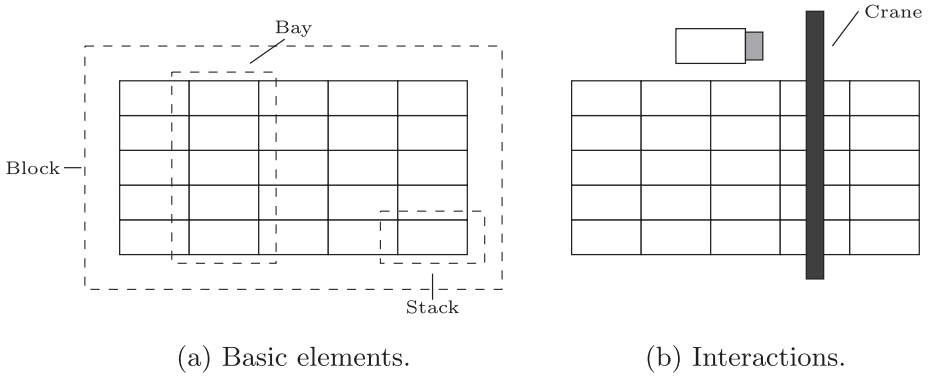


Fig. 1. Top view of a container yard scheme. Subfigure (a) shows the basic elements of the problem: the block, bay, and stacks. Subfigure (b) shows the interaction of block, crane and truck. Yard cranes move along the block on top of containers. Trucks move along the bays to the bay in which a container has to be loaded or unloaded.

of container moves to transform the initial layout of a bay into a final layout without any containers blocking the removal of other containers. 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 (Lin et al., 2015; Firmino et al., 2019; Jovanovic et al., 2019b). 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 twistlock[1] 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 practical size 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 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).

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

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; Wang et al., 2017) and the deep learning tree search of Hottung et al. (2020).

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 (BRP), the containers must be retrieved from a bay in a given order. Surveys by Caserta et al. (2011) and by Lehnfeld and Knust (2014) cover both the CPMP and BRP as well as related problems. Recent papers in block relocation include exact approaches (Tanaka and Mizuno, 2018; Zhu et al., 2019; Quispe et al., 2018; Tanaka and Voß, 2019) and different heuristic approaches, such as tree search procedures (Zhang et al., 2016), Beam Search (Bacci et al., 2019), GRASP (Jovanovic et al., 2019a), and Ant Colony Optimization (Jovanovic et al., 2019b).

Several authors working on block relocation problems have considered the time required by the crane to perform moves as the objective function, although most studies consider the minimization of the moves required. Lee and Lee (2010) use the weighted sum of the number of moves and the crane working time as their objective function. Lin et al. (2015) develop a heuristic algorithm to minimize the number of moves. They show that solutions with the minimum number of moves do not produce minimum working times, and the trade-off between these objectives can be controlled by adjusting the penalty values. Jovanovic et al. (2019b) develop an ant colony optimization algorithm. The approach initially considers the standard minimization of the number of moves, but then adapts the same algorithm to the objective of minimizing the total crane working time. Firmino et al. (2019) only consider the crane working time as the objective of their reactive GRASP algorithm, integer linear model, and A$*$ search algorithm. It should be noted that in all these studies crane times are calculated considering constant speeds for the crane trolley and spreader, distinguishing if the crane is loaded or unloaded. By contrast, in this paper, we consider a more complex crane time computation that also considers crane acceleration and twistlock times. The crane time we model closely resembles real observed crane working times.

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 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.

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 $p \in \{1, ...,P\} = \mathcal{P}$, 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 $sol = (s_1, h_1, k_1, e_1)...(s_i, h_i, k_i, e_i)...(s_n, h_n, k_n, e_n)$, 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 after move n if $p_n(s, h) \geqslant p_n(s, h + 1)$ for all $s \in \mathcal{S}$ and $h \in \mathcal{H}\backslash\{H\}$. The CPMP is a well known NP-Hard problem. It follows that the CPMPCT is also NP-Hard as it can be reduced to the CPMP considering unitary times for the crane moves.

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

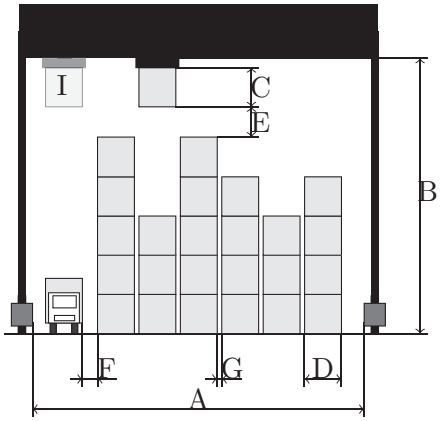


Fig. 2. Front view of an RTG crane.

common operation that can be observed in most gantry cranes.

Fig. 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.

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 when the crane moves horizontally or vertically it requires d^{α} meters to reach its maximum speed. Up to that distance, the acceleration is constant and therefore depicts a uniformly accelerated rectilinear motion (u.a.r.m). Once it reaches the maximum speed, the crane follows a uniformly rectilinear motion (u.r.m) and starts to decelerate when there are d^{α} meters left until its destination point. If the distance travelled by the crane is less than $2d^{\alpha}$, half the distance is accelerating and the other half is decelerating.

Once the maximum speeds of the crane and the distances required to reach them have been set, Eq. (1) calculates the acceleration $[m/s^2]$ for the moves considered. It is obtained by using the equations of the u.a.r.m.

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

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 Eq. (2). If $x < 2d^{\alpha}$, the time spent by the crane to reach $x/2$ meters equals $\sqrt{x/a^{\alpha\beta}}$ according to the second equation of the u.a.r.m. Since we consider deceleration and acceleration to be opposite terms, the time spent by the crane to travel x is $t^{\alpha\beta}(x) = 2\sqrt{x/a^{\alpha\beta}} = (2\sqrt{a^{\alpha\beta}x})/a^{\alpha\beta}$. If $x \geqslant 2d^{\alpha}$, the first d^{α} meters in u.a.r.m require $\sqrt{2d^{\alpha}/a^{\alpha\beta}}$ seconds (second equation of u.a.r.m). The crane then follows u.r.m during $x - 2d^{\alpha}$ metres taking $(x - 2d^{\alpha})/vmax^{\alpha\beta}$ seconds. The last d^{α} metres it is decelerating, which is again u.a.r.m with an associated time equal to $\sqrt{2d^{\alpha}/a^{\alpha\beta}}$. Therefore, the total time spent by the crane to perform the move is $t^{\alpha\beta}(x) = 2\sqrt{2d^{\alpha}/a^{\alpha\beta}} + (x - 2d^{\alpha})/vmax^{\alpha\beta} = (2vmax^{\alpha\beta})/a^{\alpha\beta} + (x - 2d^{\alpha})/vmax^{\alpha\beta}$.

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

Table 1
Parameters referring to storage distances represented in Fig. 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

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

Notation	Description	Value
$vmax^{vl}$	Hoisting speed loaded [m/s]	0.5
$vmax^{vu}$	Hoisting speed 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·(H-h + 1)

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 Eq. (3). Furthermore, the distance in meters between stacks s and k, $distance^r(s, k)$, follows Eq. (4). If $k = 0$, the crane is moved from (or to) the initial point, represented as I in Fig. 2, to (or from) stack s.

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

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

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 $p \leqslant S$. 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 $sol = (s_1, h_1, k_1, e_1)...(s_i, h_i, k_i, e_i)...(s_n, h_n, k_n, e_n)$.

$$time(sol) = t(s_1, h_1, k_1, e_1) + ...+t(s_i, h_i, k_i, e_i) + ...+t(s_n, h_n, k_n, e_n) \tag{5}$$

where

$$t\left(s_i, h_i, k_i, e_i\right) = \begin{cases} c^0_{k_{i-1}s_i} + v^0_{h_i} + b_{h_i} + v^1_{h_i} + c^1_{s_ik_i} + v^1_{e_i} + v^0_{e_i}, & \text{If } i \neq 0 \\ c^0_{s_i0} + v^0_{h_i} + b_{h_i} + v^1_{h_i} + c^1_{s_ik_i} + v^1_{e_i} + v^0_{e_i}, & \text{If } i = 0 \end{cases} \tag{6}$$

$$v^0_h = t^{vu}(distance^v(h)); \quad v^1_h = t^{vl}(distance^v(h));$$
$$c^0_{sk} = t^{ru}(distance^r(s, k)); \quad c^1_{sk} = t^{rl}(distance^r(s, k)); \tag{7}$$

For each move (s_i, h_i, k_i, e_i) 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 $c^0_{k_{i-1}s_i}$ (or $c^0_{s_i0}$ if the crane is originally at the initial point), the time spent to move the crane down to reach the container $v^0_{h_i}$, the twistlock time b_{h_i}, the time spent to hoist the container up $v^1_{h_i}$, the time spent to move the crane (loaded) along the upper travel line to the destination stack $c^1_{s_ik_i}$, the time spent to move the crane down to release the container $v^1_{e_i}$, and the time spent to position the crane on the upper travel line $v^0_{e_i}$.

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

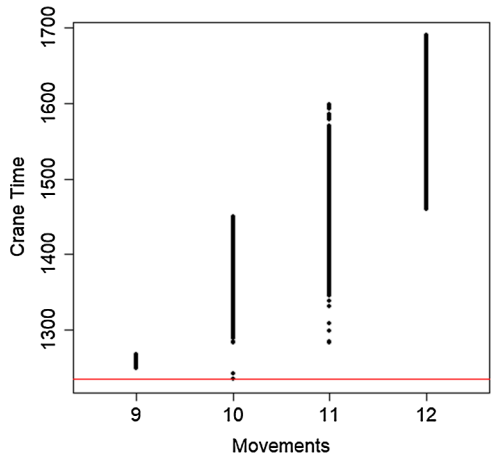
$$t_{min} = c^0_{12} + v^0_H + b_H + v^1_H + c^1_{12} + v^1_H + v^0_H = (2v^0_H + c^0_{12}) + (2v^1_H + b_H + c^1_{12}) \tag{8}$$

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.

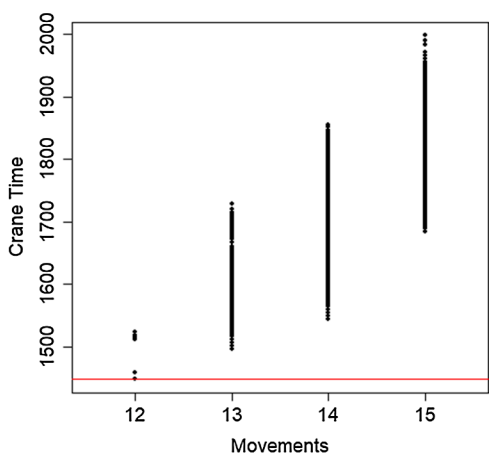
3.3. Crane time against number of movements

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. Fig. 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 Fig. 3b and c. However, in Fig. 3a and d the best solution found for the CPMPCT is not the optimal one for the CPMP.

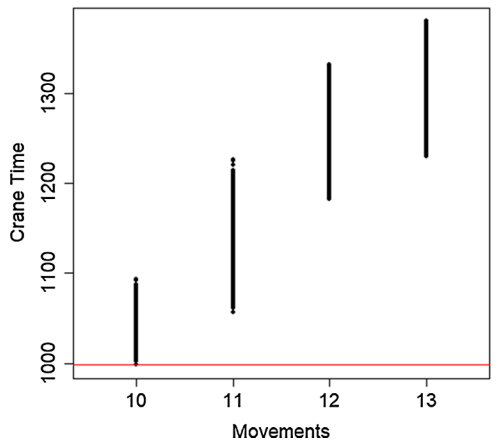
Fig. 4 illustrates an example in which the minimum number of moves to solve the CPMP is 3, with a crane time of 393.7 s (Fig. 4a). However, the number of moves to optimally solve the CPMPCT is 4, with a crane time of 386.5 s (Fig. 4b).



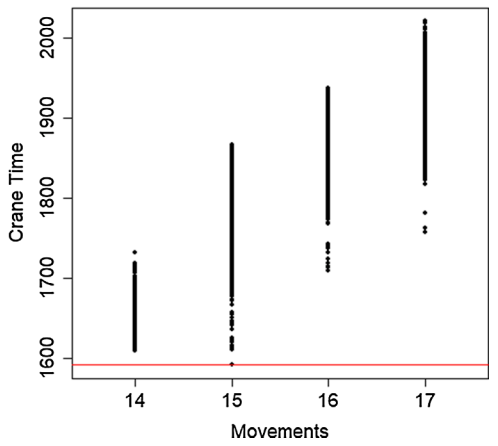
(a) BZ: data_p6_s5_h6_f50_2.txt



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



(c) EMM: emm_s7_t4_p0.75_c1_21.txt



(d) ZJY: 6_5_7.txt

Fig. 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.

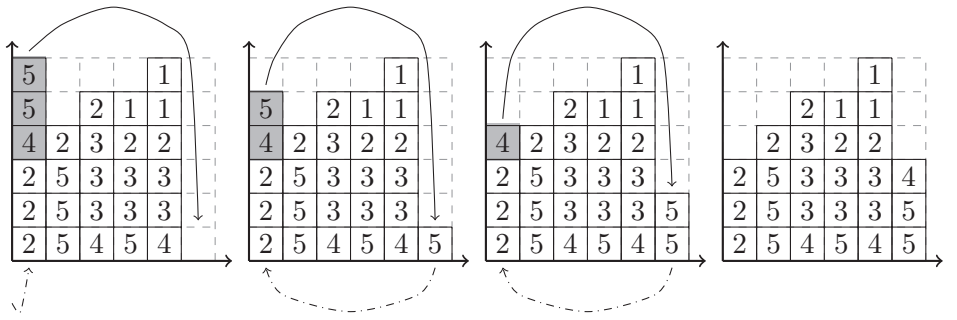
4. Bounds and dominance criteria

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.

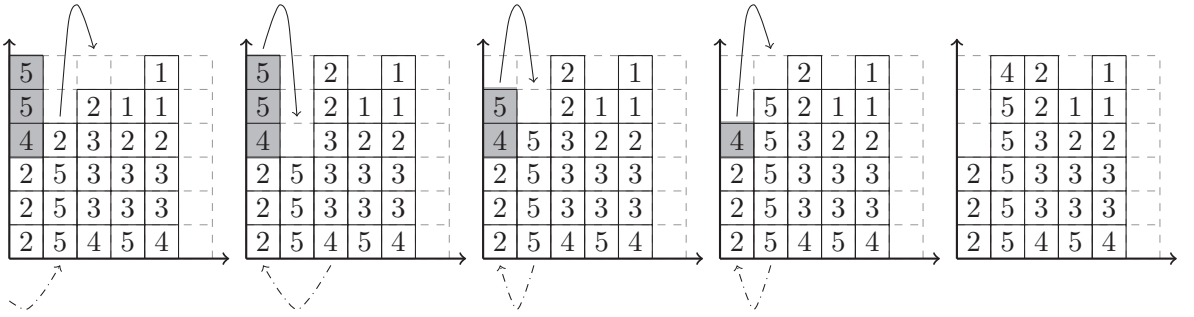
In the following proofs, let *sol* be a feasible solution for the CPMP and $time(sol)$ be the time spent by the crane to carry out the solution. We denote as γ the number of blocking containers in the initial layout of the bay and the $\eta_1, ...,\eta_\gamma$ tiers where they are initially stored.

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 CPMPCT, 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 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.



(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.

Fig. 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).

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

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

$$time(sol^*) = t(s_1, h_1, k_1, e_1) + ... + t(s_i, h_i, k_i, e_i) + ... + t(s_n, h_n, k_n, e_n) \geqslant t_{min} + ... + t_{min} + ... + t_{min} \geqslant n \cdot t_{min}$$

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

$$time(sol) \geqslant time(sol^*) \geqslant n \cdot t_{min} \Leftrightarrow \frac{time(sol)}{t_{min}} \geqslant n$$

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

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

$$\left\lfloor \frac{time(sol) - \sum_{i=1}^{\gamma} (c_{12}^0 + v_{\eta_i}^0 + b_{\eta_i} + v_{\eta_i}^1 + c_{12}^1 + v_H^1 + v_H^0) + (\gamma \cdot t_{min})}{t_{min}} \right\rfloor$$

Proof. In order to arrange the bay, all the blocking containers in the initial layout must be moved at least once. Let $sol^* = (s_1, h_1, k_1, e_1)...(s_i, h_i, k_i, e_i)\ ...(s_n, h_n, k_n, e_n)$ 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:

$$time(sol) \geqslant time(sol^*) \geqslant\ (n - \gamma) \cdot t_{min} + \sum_{i=1}^{\gamma} (c_{12}^0 + v_{\eta_i}^0 + b_{\eta_i} + v_{\eta_i}^1 + c_{12}^1 + v_H^1 + v_H^0)$$

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

$$\lfloor time(sol) - \sum_{i=1}^{\gamma} (c_{12}^0 + v_{\eta_i}^0 + b_{\eta_i} + v_{\eta_i}^1 + c_{12}^1 + v_H^1 + v_H^0) + (\gamma \cdot t_{min}) t_{min} \rfloor \geqslant n$$

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. $LB^0_{ct} := LB \cdot t_{min}$ 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:

$$LB^1_{ct} := (LB - \gamma) \cdot t_{min} + \sum_{i=1}^{\gamma} (c^0_{12} + v^0_{\eta_i} + b_{\eta_i} + v^1_{\eta_i} + c^1_{12} + v^1_H + v^0_H) \tag{9}$$

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 η_i to move blocking container i. Therefore, the minimum time to move container i is $(c^0_{12} + v^0_{\eta_i} + b_{\eta_i} + v^1_{\eta_i} + c^1_{12} + v^1_H + v^0_H)$, which is greater than (or equal to) t_{min}.

Let $\tau_1, \ldots, \tau_\gamma$ 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 Fig. 5 for further details.

Proposition 5. A valid lower bound for the CPMPCT is:

$$LB^2_{ct} := (LB - \gamma) \cdot t_{min} + \sum_{i=1}^{\gamma} (c^0_{12} + v^0_{\eta_i} + b_{\eta_i} + v^1_{\eta_i} + c^1_{12} + v^1_{\tau_i} + v^0_{\tau_i}) \tag{10}$$

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 $\tau_1, \ldots, \tau_\gamma$. Suppose that we can move some blocking container to a higher position, this happen in the case that a non blocking container has been placed in one of the levels $\tau_1, \ldots, \tau_\gamma$ and therefore this vertical distance has already been accounted for, so the bound holds.

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 CPMPCT 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 Fig. 6. Two sequences of 4 moves, leading to the same configuration are not equivalent in terms of crane times.

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 Fig. 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 $t1 = c^1_{s_i k_i} + c^0_{k_i s_{i+1}} + c^1_{s_{i+1} s_i}$ and $t2 = c^0_{s_i s_{i+1}} + c^1_{s_{i+1} k_i} + c^0_{k_i s_i}$, then $t1 > t2$.

Proof. See A.

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

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

$$\begin{aligned}
time(sol) = {} & t(s_1, h_1, k_1, e_1) + \ldots + (c^0_{k_{i-1} s_i} + c^1_{s_i k_i} + c^0_{k_i s_{i+1}} + c^1_{s_{i+1} s_i}) + (v^0_{h_i} + b_{h_i} + v^1_{h_i} \\
& + v^1_{e_i} + v^0_{e_i} + v^0_{h_{i+1}} + b_{h_{i+1}} + v^1_{h_{i+1}} + v^1_{h_i} + v^0_{h_i}) + \ldots + t(s_n, h_n, k_n, e_n) \\
> {} & t(s_1, h_1, k_1, e_1) + \ldots + (c^0_{k_{i-1} s_i} + c^0_{s_i s_{i+1}} + c^1_{s_{i+1} k_i} + c^0_{k_i s_i}) + (v^1_{e_i} + v^0_{e_i} \\
& + v^0_{h_{i+1}} + b_{h_{i+1}} + v^1_{h_{i+1}}) + \ldots + t(s_n, h_n, k_n, e_n) \\
> {} & t(s_1, h_1, k_1, e_1) + \ldots + (c^0_{k_{i-1} s_{i+1}} + c^1_{s_{i+1} k_i}) + (v^1_{e_i} + v^0_{e_i} + v^0_{h_{i+1}} + b_{h_{i+1}} + v^1_{h_{i+1}}) \\
& + \ldots + t(s_n, h_n, k_n, e_n) = time(sol_1)
\end{aligned}$$

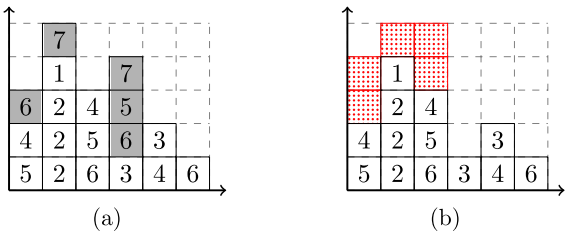
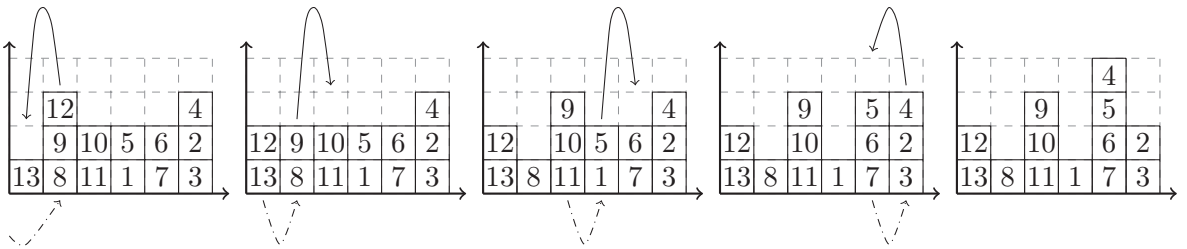
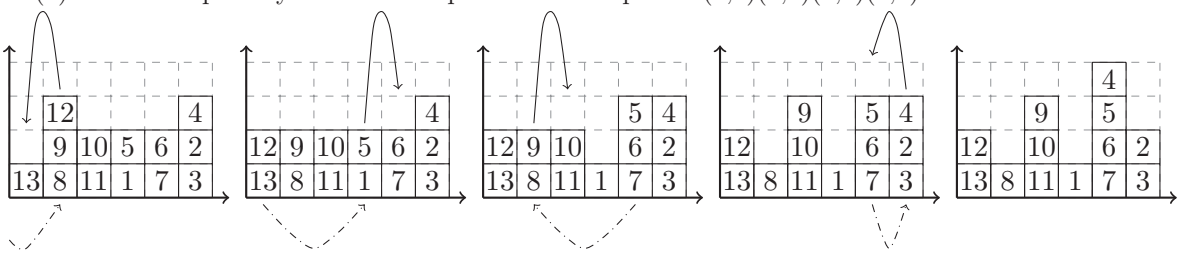


Fig. 5. An example showing a layout with 5 blocking containers and a selection of the 5 highest tiers to which the blocking containers could be moved. Subfigure (a) shows the initial layout in which the blocking containers are marked in gray. In Subfigure (b), only the non blocking containers are represented and a selection of the highest tiers to which they can be moved are highlighted in red. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

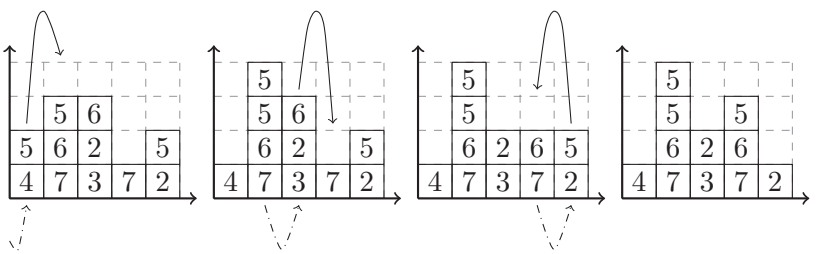


(a) The time spent by the crane to perform the sequence (2,1)(2,3)(4,5)(6,5) is 405.813 seconds.

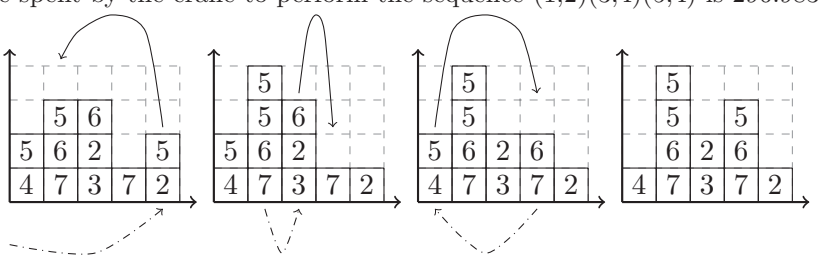


(b) The time spent by the crane to perform the sequence (2,1)(4,5)(2,3)(6,5) is 413.864 seconds.

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



(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.

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

The first inequality is satisfied by Proposition 6 and the fact that the times are always positive. The second inequality is satisfied because $(c^0_{k_{i-1}s_i} + c^0_{s_is_{i+1}}) \leqslant c^0_{k_{i-1}s_{i+1}}$ and $(c^0_{k_is_i} + c^0_{s_is_{i+2}}) \leqslant c^0_{k_is_{i+2}}$.

Another dominance rule from the CPMP that applies is the direct transitive rule. This rule refers to the movement of a container from position (s_i, h_i) to (k_i, e_i) and then from (k_i, e_i) to another position (k_{i+1}, e_{i+1}) in two moves rather than in a single move from (s_i, h_i) to (k_{i+1}, e_{i+1}).

Proposition 8. A sequence $sol = (s_1, h_1, k_1, e_1)...(s_i, h_i, k_i, e_i)(k_i, e_i, k_{i+1}, e_{i+1})...(s_n, h_n, k_n, e_n)$ 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 $sol_1 = (s_1, h_1, k_1, e_1)...(s_i, h_i, k_{i+1}, e_{i+1})...(s_n, h_n, k_n, e_n)$ with a shorter crane time:

$$\begin{aligned} time(sol) &= t(s_1, h_1, k_1, e_1)+...+(c^0_{k_{i-1}s_i} + c^1_{s_ik_i} + c^0_{k_ik_i} + c^1_{k_ik_{i+1}}) \\ &+ (v^0_{h_i} + b_{h_i} + v^1_{h_i} + v^1_{e_i} + v^0_{e_i} + v^0_{e_i} + b_{e_i} + v^1_{e_i} + v^1_{e_{i+1}} + v^0_{e_{i+1}})+...+t(s_n, h_n, k_n, e_n) \\ &> t(s_1, h_1, k_1, e_1)+...+(c^0_{k_{i-1}s_i} + c^1_{s_ik_{i+1}} + v^0_{h_i} + b_{h_i} + v^1_{h_i} + v^1_{e_{i+1}} + v^0_{e_{i+1}})+... \\ &+ t(s_n, h_n, k_n, e_n) = time(sol_1) \end{aligned}$$

The inequality follows from the fact that times are always positive and from the fact that we consider crane acceleration $((c^1_{s_ik_i} + c^1_{k_1k_{i+1}}) > c^1_{s_ik_{i+1}})$.

5. Integer programming models

We model the CPMPTC 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.

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

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

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

$$l_{skt} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1 \text{ there is a move from stack } s \text{ to stack } k; \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall\, s, k \in \mathcal{S}\colon s \neq k\ ; \quad \forall\, t \in \mathcal{T}\backslash\{T\};$$

$$u_{skt} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1 \text{ there is a move from stack } k \text{ and the} \\ & \text{destination stack of the previous move was stack } s; \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall\, s, k \in \mathcal{S}\colon s \neq k\ ; \quad \forall\, t \in \mathcal{T}\backslash\{T\};$$

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 $v_h = v^0_h + v^1_h$ to simplify notation.

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

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

The upper line of Eq. (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 second term corresponds to the movements of the loaded crane along the upper travel line from the stacks where the containers are initially located to their destination stacks. The last term of the first line corresponds to the horizontal moves performed to position the crane in the stacks where the next container to be moved is placed. With respect to the bottom line, the first term considers the vertical moves performed to reach the container and to hoist it up (in this case the container is also twistlocked). Finally, the vertical moves of the crane to release the container and position the spreader again on the upper travel line are considered in the second term of the bottom line.

The constraints of *IPCT* model are as follows.

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

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

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

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

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

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

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

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

$$\sum_{h=1}^{H} \sum_{p=1}^{P} z_{shpt} + \sum_{h=1}^{H} \sum_{p=1}^{P} w_{shpt+1} \leqslant 1 \quad \forall\, s \in \mathcal{S},\ \forall\, t \in \mathcal{T}\backslash\{T-1,\ T\} \tag{20}$$

$$\sum_{h=1}^{H} w_{shpt} + \sum_{h=1}^{H} z_{shpt+1} \leqslant 1 \quad \forall\, s \in \mathcal{S},\ \forall\, p \in \mathcal{P},\ \forall\, t \in \mathcal{T}\backslash\{T-1,\ T\} \tag{21}$$

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

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

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

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

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 hold 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. If positions (s, h) and $(s, h + 1)$ are occupied at point t, then no containers can be moved to those positions nor can be moved from position (s, h) between t and $t + 1$ points. Similarly, if a container is moved from position (s, h) between t and $t + 1$ points, then that position is occupied at point t, there are no containers on its top, and no containers can be loaded to that stack between t and $t + 1$. 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. Constraints (22) and (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. Finally, constraints (24) and (25) are valid constraints ensuring that neither a container of the lowest priority nor a container placed on the bottom tier of a stack will be moved on the last movement.

6. Branch and bound algorithm

6.1. Structure of the search tree

We propose a branch and bound algorithm in which the search strategy consists of repeatedly executing a limited depth version of a depth first search with increasing depth limits. It is known in computer science as an *iterative deepening search* and allows us to conduct a breadth-first search (BFS) in a depth-first way so as not to run out of memory. Algorithm 1 shows the pseudocode of our approach. Given the initial configuration of the bay, on line 2 the function LOWERBOUND() computes the lower bound for the number of moves lb. On line 3, the upper bound on the number of moves to solve the CPMPCT, u, is initialized to infinity. The procedure then tries on line 4 to obtain a feasible solution, that is, a solution without blocking containers, using the heuristic algorithm HEURISTIC() described in Section 6.2. If a feasible solution π is obtained, the upper bound for the number of moves, u, is updated by function UPPER (), according to Proposition 2. The function DFS() runs a DFS up to a depth l. The value l is initialized at lb on line 6. Unlike in the CPMP, even if the approach finds a feasible solution with l moves, the algorithm increases the depth by one (line 11) and runs the DFS again, as it is possible to obtain solutions with shorter crane times and more moves. Every time a better solution π is found, the upper bound u is updated. The algorithm stops once $l = u$, ensuring that the solution obtained is optimal. If a solution with $moves(\pi) = l$ is reached and the DFS with depth limit $l + 1$ does not improve it, the depth limit is directly updated to u (line 9).

Algorithm 1. Branch and bound algorithm pseudocode.

```
1: function CTA(bay)
2:   lb ← LowerBound(bay)        ▷Lower bound for the number of moves
3:   u ← ∞                       ▷Upper bound for the number of moves
4:   π ← Heuristic(bay, ∅, ∅, u) ▷Feasible solution
5:   if π ≠ ∅ then u ← Upper(bay, π)
6:   l ← lb
7:   while l ⩽ u do
8:     DFS(bay, π, l, u)
9:     if l = moves(π) + 1 and l ≠ u then l ← u
10:    else
11:      l ← l + 1
12: return π
```

All possible moves are explored at each level of the DFS, 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^1_{ct}, exceeds the crane time of the best solution obtained so far. If LB^1_{ct} does not exceed it, we use LB^2_{ct}, but as its computational cost is higher, we only compute it if considering an approximation, in which all blocking containers could be moved to the bottom tier, the branch could be pruned. The DFS 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^1_{ct}. During the DFS, the heuristic algorithm HEURISTIC() is run at every node to try to obtain a feasible solution.

6.2. A heuristic algorithm for repairing non-completely arranged solutions

Given a non-completely arranged solution π', the best solution achieved so far π, and the upper bound for the number of moves

u, *Heuristic*() tries to obtain a solution better than the current one (see Algorithm 2 for the pseudocode). The heuristic makes BG[2] and GG moves such as those proposed in Tanaka and Tierney (2018), but also GB moves, stopping when the number of moves performed is greater than the upper bound for the number of moves or when the time spent to perform those moves plus a the lower bound for the time spent to perform a move t_{min} exceeds the best current solution (line 4).

The heuristic first selects the best BG move by function *BestBG*() on line 6 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. The smaller the difference, the smaller the number of containers that can be placed between the two containers.
2. The container with the highest priority. Therefore, the placement of this container does not affect other BG movements of the blocking containers of the current disposition of the bay.

When no further BG moves are possible, the heuristic selects the best GG move by *BestGG*() on line 7 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. The larger the difference, the larger the number of containers that can be placed on the top of both stacks.
2. The move with the highest priority of the new top of the origin stack. The larger it is, the more containers can be placed on top of it.
3. The move with the fewest number of containers in the origin stack. This allows the heuristic to empty a complete stack in later moves.

If a BG or GG move has been selected, it is applied and added to the partial solution on line 11 by *Apply*(). 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 on line 13. 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 by function TRY_GB() on line 9.

Algorithm 2. Heuristic pseudocode.

```
1: function Heuristic(bay, π′, π, u)
2:   M ← π′ ▷Partial solution
3:   while moves(M) < u and time(M) + t_min < time(π) do
4:     ▷u: Upper bound for the number of moves; t_min: Minimum time spent to perform a move
5:       m ← ∅
6:       m ← BestBG(bay)
7:       if m = ∅ then m ← BestGG(bay)
8:       if m = ∅ then
9:         flag ← Try_GB(M, bay)
10:      else
11:        M ← Apply(bay, M, m) ▷Make the move
12:        ifBlocks(M) = 0 and time(M) < time(π)then
13:          π ← M ▷Update the best solution so far
14:          return π
15:      if m = ∅ and flag = False then
16:    return ∅
17:  return π
```

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 Fig. 8 for a example in which a feasible solution is obtained by performing GB moves.

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

[2] According 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 not blocking. A *good-bad* (GB) move is a move in which a non-blocking container is moved to a stack where it blocks the retrieval of other containers.

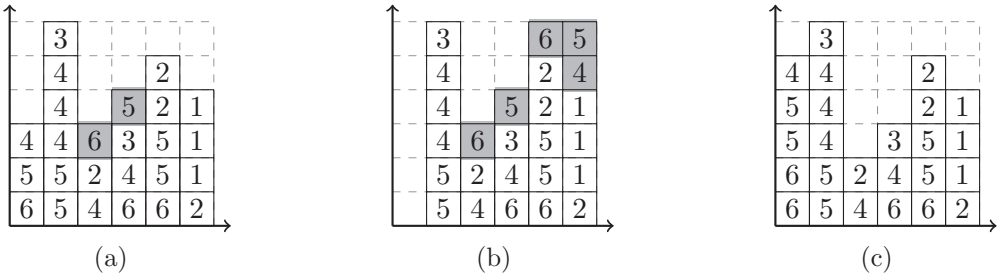


Fig. 8. Example in which a feasible solution is obtained by performing GB moves. Subfigure (a) shows a layout in which neither BG nor GG moves satisfying the heuristic criteria can be performed. Subfigure (b) shows the layout after emptying one stack. Finally, the bay is arranged by performing only BG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

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 s 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 asses 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).

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 s. On the other hand, BZ dataset has the shortest average running time, 129.77 s and is followed by EMM dataset with an average of 248.23 s.

In the instances studied in Table 3, the number of moves to solve the CPMP equals the number of moves to solve the CPMPCT. This happens because we only consider the instances solved optimally by *IPS6*, and the average number of moves is relatively small (about 8 moves). However, Fig. 3 in Section 3.3 shows four cases that belong to data sets from the literature, and in two of them the number of movements in the optimal solution for the CPMP is less than that of the CPMPCT. This situation is more frequent as the size of the bay grows and the number of movements needed to reorganize it increases.

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

Table 3
Performance of *IPCT* model on the instances optimally solved by *IPS6* on the EMM, ZJY, BZ, and CV datasets grouped by number of tiers (H) and number of stacks (S).

				IPCT					IPS6		IPCT vs. IPS6	
						Avg.			Avg.			
Dataset	H	S	#Inst.	#Opt.	#Feas.	Moves	CTime	CPU(s)	Moves	CTime	AVRPD	MAXRPD
EMM	4	4	150	150	150	4.76	472.93	251.90	4.76	477.46	1.13	21.29
	4	7	126	115	124	5.77	605.08	185.89	5.77	630.02	3.75	14.66
	4	10	115	79	109	6.29	670.03	332.02	6.29	715.00	6.28	15.30
		Total	391	344	383	5.45	562.37	248.23	5.45	583.01	3.19	21.29
ZJY	4	6	20	10	17	8.40	878.60	740.87	8.40	912.82	3.48	9.41
	4	7	17	11	16	9.36	987.85	533.23	9.36	1012.90	2.62	8.91
	4	8	18	9	17	7.33	794.97	839.89	7.33	813.80	2.61	5.54
	4	9	17	4	13	8.00	846.07	265.92	8.00	880.24	3.90	4.45
	5	6	9	1	8	11.00	1261.70	2017.25	11.00	1263.11	0.11	0.11
		Total	81	35	71	8.46	898.66	683.26	8.46	925.10	2.94	9.41
BZ	4	3	120	120	120	3.69	374.34	0.62	3.69	376.93	0.54	7.68
	4	5	120	120	120	7.58	444.76	5.00	7.58	457.37	2.61	14.15
	4	7	120	117	120	4.38	535.70	100.70	4.38	562.20	4.74	18.51
	4	9	120	98	119	8.23	600.71	200.94	8.23	644.29	6.96	24.03
	6	3	120	116	119	5.26	917.30	43.50	5.26	924.27	0.79	9.09
	6	5	120	95	100	8.52	1023.02	237.54	8.52	1054.03	2.87	18.69
	6	7	104	64	92	5.92	1050.33	391.55	5.92	1111.04	5.42	13.83
	6	9	93	32	83	8.31	1018.88	439.58	8.31	1106.15	7.69	15.10
		Total	893	762	873	6.09	686.69	129.77	6.09	712.45	3.38	24.03
CV	5	3	40	40	40	8.79	984.27	201.16	8.79	994.09	0.91	4.24
	5	4	38	34	37	8.41	973.75	342.69	8.41	992.59	2.12	12.66
	5	5	32	21	29	8.38	987.73	817.42	8.38	1018.80	3.15	7.91
	5	6	23	8	15	8.00	956.86	1057.91	8.00	985.91	2.76	5.34
	5	7	11	1	5	8.00	940.13	868.36	8.00	1010.72	6.98	6.98
	5	8	7	0	2	–	–	–	–	–	–	–
	6	4	9	5	7	9.60	1170.06	861.35	9.60	1179.32	0.79	1.74
	6	5	2	1	1	11.00	1360.83	3032.89	11.00	1391.82	2.23	2.23
	6	6	1	0	0	–	–	–	–	–	–	–
		Total	163	110	136	8.58	991.15	486.68	8.58	1009.93	1.91	12.66

The column "#Inst." represents the total number of instances and columns "#Opt." and "#Feas." show the number of optimal and feasible solutions obtained. The "Moves" and "CTime" columns represent the average number of moves and average time spent by the crane on the instances solved to optimality and the "CPU(s)" columns the average running time (in seconds) spent by each model. The columns "AVRPD" and "MAXRPD" show the average of the relative percentage deviation and the maximum relative percentage deviation in the optimally solved instances. The notation "–" is used when the averages cannot be calculated.

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 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 s to 0.03 s and from 817.42 s to 0.15 s, 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 s for any of the datasets. In addition, an optimal solution is found for around 80% of the instances in the three datasets.

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.

Table 4
Comparing *CTA* vs. IPCT in the EMM, ZJY, BZ and CV instances to evaluate the performance of the mathematical model *IPCT* in the previous section grouped by number of tiers (H) and number of stacks (S).

				#Opt.		#Feas.			Avg.		Avg. CPU(s)	
Dataset	H	S	#Inst.	IPCT	CTA	IPCT	CTA	#Both	Moves	CTime	IPCT	CTA
EMM	4	4	150	150	150	150	150	150	4.76	472.93	251.90	0.27
	4	7	126	115	125	124	126	115	5.77	605.08	185.89	0.79
	4	10	115	79	83	109	115	76	6.15	654.80	294.54	107.00
		Total	391	344	358	383	391	341	5.41	558.03	239.14	24.23
ZJY	4	6	20	10	20	17	20	10	8.40	878.60	740.87	0.41
	4	7	17	11	17	16	17	11	9.36	987.85	533.23	1.32
	4	8	18	9	17	17	18	9	7.33	794.97	839.89	16.90
	4	9	17	4	9	13	17	4	8.00	846.07	265.92	18.27
	5	6	9	1	9	8	9	1	11.00	1261.70	2017.25	6.26
		Total	81	35	72	71	81	35	8.46	898.66	683.26	7.14
BZ	4	3	120	120	120	120	120	120	3.69	374.34	0.62	0.01
	4	5	120	120	120	120	120	120	4.38	444.76	5.00	0.02
	4	7	120	117	119	120	120	117	5.26	535.70	100.70	4.85
	4	9	120	98	98	119	120	96	5.81	590.17	135.02	65.75
	6	3	120	116	120	119	120	116	7.58	917.30	43.50	0.02
	6	5	104	95	102	100	104	94	8.16	1014.21	211.95	19.80
	6	7	96	64	56	92	96	53	7.76	962.17	125.93	201.11
	6	9	93	32	21	83	93	21	7.29	918.22	267.80	502.67
		Total	893	762	756	873	893	737	5.90	664.38	85.06	40.65
CV	5	3	40	40	40	40	40	40	8.78	984.27	201.16	0.01
	5	4	38	34	38	37	38	34	8.41	973.75	342.69	0.03
	5	5	32	21	32	29	32	21	8.38	987.73	817.42	0.15
	5	6	23	8	23	15	23	8	8.00	956.86	1057.91	0.54
	5	7	11	1	11	5	11	1	8.00	940.13	868.36	0.52
	5	8	7	0	6	2	7	0	–	–	–	–
	6	4	9	5	9	7	9	5	9.60	1170.06	861.35	0.07
	6	5	2	1	2	1	2	1	11.00	1360.83	3032.89	1.13
	6	6	1	0	1	0	1	0	–	–	–	–
		Total	163	110	162	136	163	110	8.58	991.15	486.68	0.10

The column "#Inst." gives the total number of instances tested at each group, the columns "#Opt." and "#Feas." represent the number of optimal and feasible solutions obtained by each approach, and the column "#Both" provides the number of instances in which both of them reached the optimal solution. The "Moves", "CTime" and "Avg. CPU(s)" columns represent the average number of moves, average time spent by the crane, and the average running time (in seconds). The notation "–" is used when the averages cannot be calculated.

Table 5
Performance of *CTA* on EMM, ZJY, and BZ datasets grouped by number of tiers (H) and stacks (S).

					Avg. Opt.				Avg. Feas.		
Dataset	H	S	#Inst.	#Opt.	Moves	CTime	CPU(s)	#Feas.	Moves	CTime	CPU(s)
EMM	4	4	225	150	4.76	472.93	0.27	150	4.76	472.93	0.27
	4	7	225	125	6.32	657.64	27.13	150	8.64	905.64	87.25
	4	10	225	83	6.52	692.36	197.32	150	11.51	1222.22	395.29
		Total	675	358	5.71	588.30	55.33	450	8.30	866.93	160.94
ZJY	4	6	20	20	10.15	1053.42	16.92	20	10.15	1053.42	16.92
	4	7	20	18	10.22	1071.74	23.39	20	10.70	1122.24	21.24
	4	8	20	17	9.18	976.58	238.74	20	9.85	1053.99	202.94
	4	9	20	10	10.20	1078.95	585.43	20	11.80	1266.02	295.57
	5	6	20	15	13.00	1448.46	488.19	20	14.05	1582.43	366.21
		Total	100	80	10.50	1118.48	224.94	100	11.31	1215.62	180.58
BZ	4	3	120	120	3.69	374.34	0.01	120	3.69	374.34	0.01
	4	5	120	120	4.38	444.76	0.02	120	4.38	444.76	0.02
	4	7	120	119	5.37	546.53	12.55	120	5.43	553.47	12.50
	4	9	120	98	5.92	600.48	94.52	120	7.13	724.87	128.94
	5	3	120	120	7.88	949.12	0.03	120	7.88	949.12	0.03
	5	5	120	105	8.96	1102.35	47.29	120	10.06	1233.84	41.40
	5	7	120	56	8.02	996.30	265.96	120	12.38	1528.31	265.03
	5	9	120	21	7.29	918.22	502.67	120	15.40	1932.41	814.76
		Total	960	759	6.16	694.19	54.25	960	8.29	967.64	157.84

The column "#Inst." gives the total number of instances tested in each group and the columns "#Opt." and "#Feas." represent the number of optimal and feasible solutions obtained. The "Moves", "CTime" and "CPU(s)" columns represent the average number of moves, average time spent by the crane and the average running time (in seconds), respectively.

Table 6
Performance of *CTA* on CV dataset grouped by number of tiers (H) and number of stacks (S).

				Avg. Opt.				Avg. Feas.		
H	S	#Inst.	#Opt.	Moves	CTime	CPU(s)	#Feas.	Moves	CTime	CPU(s)
5	3	40	40	8.78	984.27	0.01	40	8.78	984.27	0.01
5	4	40	40	9.03	1043.19	0.11	40	9.03	1043.19	0.11
5	5	40	40	10.15	1187.69	37.69	40	10.15	1187.69	37.69
5	6	40	35	10.77	1270.00	204.94	40	11.33	1338.96	179.33
5	7	40	25	11.48	1374.62	610.68	40	12.85	1531.04	384.34
5	8	40	11	11.00	1326.97	985.54	40	13.53	1633.31	273.04
6	4	40	35	15.26	1830.44	61.12	40	15.83	1897.71	53.76
6	5	40	9	14.56	1789.99	78.93	40	17.95	2218.00	18.54
6	6	40	3	12.67	1600.40	921.00	40	19.33	2405.58	76.17
6	7	40	0	–	–	–	40	21.90	2765.90	181.16
7	4	40	6	18.17	2221.35	178.50	40	23.55	2933.43	597.13
7	5	40	1	15.00	1876.20	434.54	38	24.92	3208.36	373.53
7	6	40	0	–	–	–	37	29.49	3925.59	1288.49
7	7	40	0	–	–	–	36	31.56	4282.63	1327.28
7	8	40	0	–	–	–	36	35.94	4973.35	1780.95
7	9	40	0	–	–	–	32	39.44	5588.67	1596.88
7	10	40	0	–	–	–	30	41.10	5864.61	1758.06
8	6	40	0	–	–	–	2	36.50	5252.07	1346.71
8	10	40	0	–	–	–	3	59.33	9134.43	1632.72
	Total	760	245	11.14	1315.10	171.07	654	21.07	2737.96	546.25

The column "#Inst." gives the total number of instances tested at each group and the columns "#Opt." and "#Feas." represent the number of optimal and feasible solutions obtained. The "Moves", "CTime" and "CPU(s)" columns represent the average number of moves, average time spent by the crane and the average running time (in seconds). The notation "–" is used when the averages cannot be calculated.

8. Conclusions

We propose the pre-marshalling problem considering crane time minimization objective (CPMPCT). 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.

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 will consider more complex heuristic procedures involving other types of movements such as BB moves, although that would require a careful new study given the fact that the runtime will certainly increase. Another line of research wouldintegrate 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.

CRediT authorship contribution statement

Consuelo Parreño-Torres: Conceptualization, Methodology, Software, Validation, Data curation, Writing - original draft. **Ramon Alvarez-Valdes:** Supervision, Methodology, Conceptualization, Funding acquisition, Writing - review & editing. **Rubén Ruiz:** Conceptualization, Resources, Funding acquisition, Writing - review & editing. **Kevin Tierney:** Conceptualization, Funding acquisition, Writing - review & editing.

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.

Appendix A. Proof of Proposition 6

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

$$t1 = c^1_{s_i k_i} + c^0_{k_i s_{i+1}} + c^1_{s_{i+1} s_i} = t^{rl}(distance^r(s_i, k_i)) + t^{ru}(distance^r(k_i, s_{i+1})) + t^{rl}(distance^r(s_{i+1}, s_i))$$
$$t2 = c^0_{s_i s_{i+1}} + c^1_{s_{i+1} k_i} + c^0_{k_i s_i} = t^{rl}(distance^r(s_i, s_{i+1})) + t^{ru}(distance^r(s_{i+1}, k_i)) + t^{rl}(distance^r(k_i, s_i))$$

To address this proof, we rewrite Eq. (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, so we can substitute the acceleration $a^{r\beta}$ of Eq. (2) with its corresponding expression provided by Eq. (1), $a^{r\beta} = (vmax^{r\beta})^2/2d^r$.

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

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 the three stacks in non-decreasing order, we get six cases. Nevertheless we just have to consider three of them because $distance^r(s, k) = distance^r(k, s)$:

Case 1. $s_i < k_i < s_{i+1}$, Fig. 9b.
Case 2. $s_i < s_{i+1} < k_i$, Fig. 9c.
Case 3. $k_i < s_i < s_{i+1}$, Fig. 9d.

In Fig. 9b–d, 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 $t2$.

Case 1. $s_i < k_i < s_{i+1}$, Fig. 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: $x_1, x_2, x_3 < 2d^r$

$$t_1 = \frac{2\sqrt{2d^r x_1}}{vmax^{rl}} + \frac{2\sqrt{2d^r x_2}}{vmax^{ru}} + \frac{2\sqrt{2d^r x_3}}{vmax^{rl}},\ t_2 = \frac{2\sqrt{2d^r x_1}}{vmax^{ru}} + \frac{2\sqrt{2d^r x_2}}{vmax^{rl}} + \frac{2\sqrt{2d^r x_3}}{vmax^{ru}},$$

$$\begin{aligned} t_1 - t_2 &= \frac{2\sqrt{2d^r x_1}\, vmax^{ru} + 2\sqrt{2d^r x_2}\, vmax^{rl} + 2\sqrt{2d^r x_3}\, vmax^{ru}}{vmax^{rl} vmax^{ru}} \\ &\quad - \frac{2\sqrt{2d^r x_1}\, vmax^{rl} + 2\sqrt{2d^r x_2}\, vmax^{ru} + 2\sqrt{2d^r x_3}\, vmax^{rl}}{vmax^{rl} vmax^{ru}} \\ &= \frac{(vmax^{ru} - vmax^{rl})(2\sqrt{2d^r x_1} + 2\sqrt{2d^r x_3} - 2\sqrt{2d^r x_2})}{vmax^{rl} vmax^{ru}} \end{aligned}$$

The difference $t1 - t2$ can be described as a function of three terms $t1 - t2 = (term_1 term_3)/term_2$, where $term_1 = (vmax^{ru} - vmax^{rl})$, $term_2 = vmax^{ru} vmax^{rl}$, and in this case $term_3 = 2\sqrt{2d^r x_1} + 2\sqrt{2d^r x_3} - 2\sqrt{2d^r x_2}$, but this structure appears in all cases, just changing the expression of $term_3$, depending on the lengths of segments x_1, x_2, and x_3 for choosing the appropriate expressions for the times. We want to prove 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. In this first case:

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

The first inequality is satisfied since $x_3 > x_2$, 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 $x_3 \geqslant 2d^r$

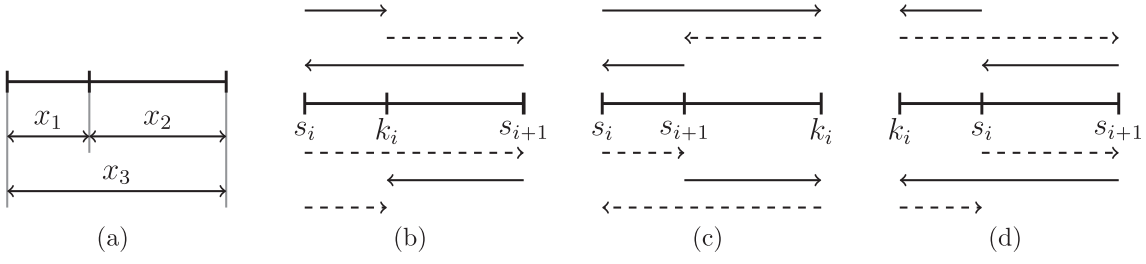


Fig. 9. Given three stacks at distances x_1, x_2, and x_3 (Fig. 9a), Fig. 9b–d show the different relative positions that mu.st be addressed.

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

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 $x_3 \geqslant 2d^r$ and that $x_1 > 0$.

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

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 $x_1, x_3 \geqslant 2d^r$.

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: $x_1, x_2, x_3 \geqslant 2d^r$.

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

Case 3. $k_i < s_i < s_{i+1}$, Fig. 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 $\sqrt{x_1} + \sqrt{x_2} \geqslant \sqrt{x_1 + x_2}$. Therefore $term_3 > 0$.

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

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 $x_3 \geqslant 2d^r$, the following inequalites are satisfied:

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

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

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 $x_1, x_3 \geqslant 2d^r$. Proof analogous to the previous scenario.

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

The last scenario follows from $x_3 = x_1 + x_2$ and $2d^r > 0$, then $term3 = x_1 + x_2 + 2d^r - x_3 = 2d^r > 0$.

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.

References

Bacci, T., Mattia, S., Ventura, P., 2019. The bounded beam search algorithm for the block relocation problem. Comput. Oper. Res. 103, 252–264. https://doi.org/10.1016/j.cor.2018.11.008.

Bortfeldt, A., Forster, F., 2012. A tree search procedure for the container pre-marshalling problem. Eur. J. Oper. Res. 217, 531–540. https://doi.org/10.1016/j.ejor.2011.10.005.

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. Springer, Berlin Heidelberg. volume 8737 of Lecture Notes in Computer Science, pp. 798–809. https://doi.org/10.1007/978-3-662-44777-2_66.

Caserta, M., Schwarze, S., Voß, S., 2011. Container rehandling at maritime container terminals, in: Böse, J. (Ed.), Handbook of Terminal Planning. Springer, New York. volume 49 of Operations Research/Computer Science Interfaces Series, pp. 247–269. https://doi.org/10.1007/978-1-4419-8408-1_13.

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. https://doi.org/10.1007/978-3-642-01129-0_89.

Expósito-Izquierdo, C., Melián-Batista, B., Moreno-Vega, M., 2012. Pre-marshalling problem: heuristic solution method and instances generator. Exp. Syst. Appl. 39, 8337–8349. https://doi.org/10.1016/j.eswa.2012.01.187.

Hottung, A., Tanaka, S., Tierney, K., 2020. Deep learning assisted heuristic tree search for the container pre-marshalling problem. Comput. Oper. Res. 113, 104781. https://doi.org/10.1016/j.cor.2019.104781.

Hottung, A., Tierney, K., 2016. A biased random-key genetic algorithm for the container pre-marshalling problem. Comput. Oper. Res. 75, 83–102. https://doi.org/10.1016/j.cor.2016.05.011.

Jovanovic, R., Tanaka, S., Nishi, T., Voß, S., 2019a. A grasp approach for solving the blocks relocation problem with stowage plan. Flexible Serv. Manuf. J. 31, 702–729. https://doi.org/10.1007/s10696-018-9320-3.

Jovanovic, R., Tuba, M., Voß, S., 2017. A multi-heuristic approach for solving the pre-marshalling problem. CEJOR 25, 1–28. https://doi.org/10.1007/s10100-015-0410-y.

Jovanovic, R., Tuba, M., Voß, S., 2019b. An efficient ant colony optimization algorithm for the blocks relocation problem. Eur. J. Oper. Res. 274, 78–90. https://doi.org/10.1016/j.ejor.2018.09.038.

Lee, Y., Chao, S.L., 2009. A neighborhood search heuristic for pre-marshalling export containers. Eur. J. Oper. Res. 196, 468–475. https://doi.org/10.1016/j.ejor.2008.03.011.

Lee, Y., Hsu, N.Y., 2007. An optimization model for the container pre-marshalling problem. Comput. Oper. Res. 34, 3295–3313. https://doi.org/10.1016/j.cor.2005.12.006.

Lee, Y., Lee, Y., 2010. A heuristic for retrieving containers from a yard. Comput. Oper. Res. 37, 1139–1147. https://doi.org/10.1016/j.cor.2009.10.005.

Lehnfeld, J., Knust, S., 2014. Loading, unloading and premarshalling of stacks in storage areas: survey and classification. Eur. J. Oper. Res. 239, 297–312. https://doi.org/10.1016/j.ejor.2014.03.011.
Lin, D.Y., Lee, Y.J., Lee, Y., 2015. The container retrieval problem with respect to relocation. Transp. Res. Part C 52, 132–143. https://doi.org/10.1016/j.trc.2015.01.024.
Parreño-Torres, C., Alvarez-Valdes, R., Ruiz, R., 2019. Integer programming models for the pre-marshalling problem. Eur. J. Oper. Res. 274, 142–154. https://doi.org/10.1016/j.ejor.2018.09.048.
Prandtstetter, M., 2013. A dynamic programming based branch-and-bound algorithm for the container pre-marshalling problem. Technical Report. Technical report, AIT Austrian Institute of Technology.
Quispe, K.E.Y., Lintzmayer, C.N., Xavier, E.C., 2018. An exact algorithm for the blocks relocation problem with new lower bounds. Comput. Oper. Res. 99, 206–217. https://doi.org/10.1016/j.cor.2018.06.021.
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. Eur. J. Oper. Res. 276, 40–56. https://doi.org/10.1016/j.ejor.2018.05.004.
da Silva Firmino, A., de Abreu Silva, R., Times, V., 2019. A reactive GRASP metaheuristic for the container retrieval problem to reduce crane's working time. J. Heurist. 25, 141–173. https://doi.org/10.1007/s10732-018-9390-0.
Tanaka, S., Mizuno, F., 2018. An exact algorithm for the unrestricted block relocation problem. Comput. Oper. Res. 95, 12–31. https://doi.org/10.1016/j.cor.2018.02.019.
Tanaka, S., Tierney, K., 2018. Solving real-world sized container pre-marshalling problems with an iterative deepening branch-and-bound algorithm. Eur. J. Oper. Res. 264, 165–180. https://doi.org/10.1016/j.ejor.2017.05.046.
Tanaka, S., Tierney, K., Parreño-Torres, C., Alvarez-Valdes, R., Ruiz, R., 2019. A branch and bound approach for large pre-marshalling problems. Eur. J. Oper. Res. 278, 211–225. https://doi.org/10.1016/j.ejor.2019.04.005.
Tanaka, S., Voß, S., 2019. An exact algorithm for the block relocation problem with a stowage plan. Eur. J. Oper. Res. 279, 767–781. https://doi.org/10.1016/j.ejor.2019.06.014.
Tierney, K., Pacino, D., Voß, S., 2017. Solving the pre-marshalling problem to optimality with A∗ and IDA∗. Flexible Serv. Manuf. J. 29, 223–259. https://doi.org/10.1007/s10696-016-9246-6.
UNCTAD, 2018. United Nations Conference on Trade and Development (UNCTAD) Review of Maritime Transport. United Nations. <https://unctad.org/en/PublicationsLibrary/rmt2018_en.pdf>.
Wang, N., Jin, B., Lim, A., 2015. Target-guided algorithms for the container pre-marshalling problem. Omega 53, 67–77. https://doi.org/10.1016/j.omega.2014.12.002.
Wang, N., Jin, B., Zhang, Z., Lim, A., 2017. A feasibility-based heuristic for the container pre-marshalling problem. Eur. J. Oper. Res. 256, 90–101. https://doi.org/10.1016/j.ejor.2016.05.061.
Wilmsmeier, G., Spengler, T., 2016. Energy consumption and container terminal efficiency. Bull. FAL 6, 1–10 https://repositorio.cepal.org/handle/11362/40928.
Zhang, R., Jiang, Z.Z., Yun, W.Y., 2015. Stack pre-marshalling problem: a heuristic-guided branch-and-bound algorithm. Int. J. Industr. Eng. 22, 509–523.
Zhang, R., Liu, S., Kopfer, H., 2016. Tree search procedures for the blocks relocation problem with batch moves. Flexible Serv. Manuf. J. 28, 397–424. https://doi.org/10.1007/s10696-015-9229-z.
Zhu, H., Ji, M., Guo, W., Wang, Q., Yang, Y., 2019. Mathematical formulation and heuristic algorithm for the block relocation and loading problem. Naval Res. Logist. 66, 333–351. https://doi.org/10.1002/nav.21843.