European Journal of Operational Research 302 (2022) 1063–1078

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Production, Manufacturing, Transportation and Logistics

A beam search algorithm for minimizing crane times in premarshalling problems

Consuelo Parreño-Torres [a,*], Ramon Alvarez-Valdes [a], Francisco Parreño [b]

[a] Department of Statistics and Operations Research, University of Valencia, Doctor Moliner 50, Burjassot, Valencia, 46100, Spain
[b] Department of Mathematics, University of Castilla-La Mancha, Albacete, Spain

ARTICLE INFO

Article history:
Received 15 June 2021
Accepted 24 January 2022
Available online 31 January 2022

Keywords:
Logistics
Container premarshalling
Crane time
Beam search

ABSTRACT

The premarshalling problem consists of sorting the containers placed in a bay of the container yard so that they can be retrieved in the order in which they will be required. We study the premarshalling problem with crane time minimization objective and develop a beam search algorithm, with some new elements adapted to the characteristics of the problem, to solve it. We propose various evaluation criteria, depending on the type of container movement, for its local evaluation; a new heuristic algorithm including local search for blue its global evaluation; and several new dominance rules. The computational study shows the contribution of each new element. The performance of the complete algorithm is tested on well-known benchmarks. The beam search algorithm matches all known optimal solutions, improves on the known suboptimal solutions, and obtains solutions for the largest instances, for which no solution had previously been found.

© 2022 The Author(s). Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license
(http://creativecommons.org/licenses/by-nc-nd/4.0/)

1. Introduction

Maritime transport accounts for more than 80% of international trade, with 11.08 billion tonnes transported in 2019 (UNCTAD, 2020). An increasing part of this volume corresponds to containerized transport. Containers are nowadays used for all kinds of goods, not only manufactured goods, but also fresh products. Container terminals handled 827 million TEUs (Twenty-foot Equivalent Unit) in 2020, with ports such as Shanghai moving more than 43 million TEUs (Statista, 2021).

The daily operation of container terminals faces increasing pressure from ship operators. On the one hand, vessel size is constantly increasing. The vessel HMM Algeciras can carry 23,964 TEUs and many similar vessels are being built and will be operational in the near future. On the other hand, the time a vessel spends in port is continuously decreasing, with a median of 0.97 days in 2019 (UNCTAD, 2020). To serve these large vessels with the required speed, terminals are continuously upgrading their facilities and machinery. On the seaside, they are installing larger quay cranes which can handle two 40-foot containers in one move, thus increasing the rate of moves per hour (Yue, Fan, & Ma, 2021). However, optimizing one subsystem is not effective if other subsystems remain deficient. In this respect, the container yard, in which containers are temporarily stored, is considered the bottleneck of the terminal. As storage space is limited and containers are stacked on top of each other, to retrieve a container that is not on top of a stack all the containers above it must be moved, resulting in unproductive and time-consuming movements. Therefore, an effective way to smooth out peak workloads once the ship arrives, and thus speed up the loading and unloading of vessels, is to use the yard cranes prior to the arrival of the ship to sort the container yard.

The problem of sorting the containers in the yard so that they will be available in the order in which they will be required is known as the Container Premarshalling Problem (CPMP). In its classical version the objective is to minimize the number of container moves required to sort the bay and several exact methods, and many heuristic and metaheuristic algorithms have been proposed to solve it. However, the number of moves does not correctly reflect the time required by the crane to sort the containers, which depends on their positions. Parreño-Torres, Alvarez-Valdes, Ruiz, & Tierney (2020) proposed a variant of the problem, the Container Premarshalling Problem with Crane Times, CPMPCT, in which the objective is to minimize the time the crane takes to sort the containers. The computational analysis carried out by the authors shows the gain in precision that can be obtained using this

* Corresponding author.
E-mail addresses: consuelo.parreno@uv.es (C. Parreño-Torres), ramon.alvarez@uv.es (R. Alvarez-Valdes), francisco.parreno@uclm.es (F. Parreño).

https://doi.org/10.1016/j.ejor.2022.01.038
0377-2217/© 2022 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license
(http://creativecommons.org/licenses/by-nc-nd/4.0/)

objective function, as opposed to the one that minimizes the number of moves, reducing the crane time by 24% in some cases.

In this paper, we address the CPMPCT and develop a beam search algorithm (Reddy et al., 1977). The basic beam search framework is embedded into an iterative procedure and problem-specific elements are added, such as considering different selection criteria for different types of container moves and developing an efficient heuristic algorithm, including local search, for the global evaluation phase. An extensive computational study on well-known benchmarks shows that the proposed beam search procedure matches the optimal solutions provided by the branch and bound algorithm of Parreño-Torres et al. (2020) and outperforms it on large instances.

The paper is structured as follows. The relevant literature is reviewed in Section 2. In Section 3, we formally describe the new problem of minimizing the crane times and show how these crane times are calculated. Section 4 introduces new dominance criteria which would be used to reduce the search. The beam search algorithm is presented in Section 5. The computational results are described in Section 6 and conclusions and future work are discussed in Section 7.

2. Literature review

Stacking problems, in which items are placed on top of each other and only the one on top is directly accessible, occur in many environments, not only in container terminals. Ge, Meng, Liu, Tang, & Zhao (2020) address the problem of stacking slabs in the steel industry and Maniezzo, Boschetti, & Gutjahr (2021) the problem of stacking boxes in a warehouse. As for the stacking problems in container terminals, which are the subject of this paper, a recent survey by Caserta, Schwarze, & Voß (2020) classifies them into three types: premarshalling, re-marshalling, and block relocation problem. The premarshalling problem is concerned with finding the shortest sequence of moves that sorts the containers within a bay according to a known retrieval sequence. It is solved for a single bay since moving the crane between bays is time consuming and can produce safety problems due to crane-crane or human-crane interactions. The re-marshalling problem is concerned with finding the minimum length sequence of moves for retrieving containers from a source bay and positioning them in a target bay. Finally, the block relocation problem aims to retrieve all the containers from a bay in the prescribed order while minimizing the number of rehandles. Caserta et al. (2020) review the relevant literature on the three problems, so here we will focus on premarshalling problems.

In the classical approach to the premarshalling problem, the purpose of the objective function has been to minimize the number of container moves required to sort the bay. Integer linear models have been proposed by Lee & Hsu (2007), van Brink & van der Zwaan (2014), de Melo da Silva, Toulouse, & Calvo (2018), and Parreño-Torres, Alvarez-Valdes, & Ruiz (2019). Other exact procedures are the A^* algorithm of Expósito-Izquierdo, Melián-Batista, & Moreno-Vega (2012), the iterative deepening A^* (IDA*) of Tierney, Pacino, & Voß (2017), and the branch and bound algorithms of Prandtstetter (2013), Zhang, Jiang, & Yun (2015), Tanaka & Tierney (2018), and Tanaka, Tierney, Parreño-Torres, Alvarez-Valdes, & Ruiz (2019). Many heuristic and metaheuristic algorithms have also been proposed, such as the heuristic algorithm of Caserta & Voß (2009), based on the Corridor Method, the neighborhood search algorithm of Lee & Chao (2009), or the heuristic tree search procedure of Bortfeldt & Forster (2012). More recently, Jovanovic, Tuba, & Voß (2017) have proposed a deterministic algorithm based on the randomized greedy procedure by Expósito-Izquierdo et al. (2012), Hottung & Tierney (2016) a biased random-key genetic algorithm, Wang, Jin, & Lim (2015), Wang, Jin, Zhang, & Lim (2017) propose target-guided procedures embedded in beam search algorithms, and Hottung, Tanaka, & Tierney (2020) a deep learning tree search procedure. Zweers, Bhulai, & van der Mei (2020) study an interesting version of the problem in which the time to perform the moves is limited, but even in this case the time is approximated by the number of moves.

However, Parreño-Torres et al. (2020) have shown that the number of moves does not accurately represent the time the crane needs to rearrange the bay. Therefore, they propose a new variant of the problem in which the objective is to minimize the crane time, in line with other studies in the closely related Block Relocation Problems in which crane time is considered in the objective function (Jovanovic, Tuba, & Voß, 2019; Lee & Lee, 2010; Lin, Lee, & Lee, 2015; da Silva Firmino, de Abreu Silva, & Times, 2019). The results reported by Parreño-Torres et al. (2020) indicate that the problem is more difficult to solve than the classical CPMP and their integer model and branch and bound algorithm do not perform well on the largest test instances in the literature. In this paper we propose a metaheuristic algorithm which obtains optimal or near-optimal solutions for all types of instances.

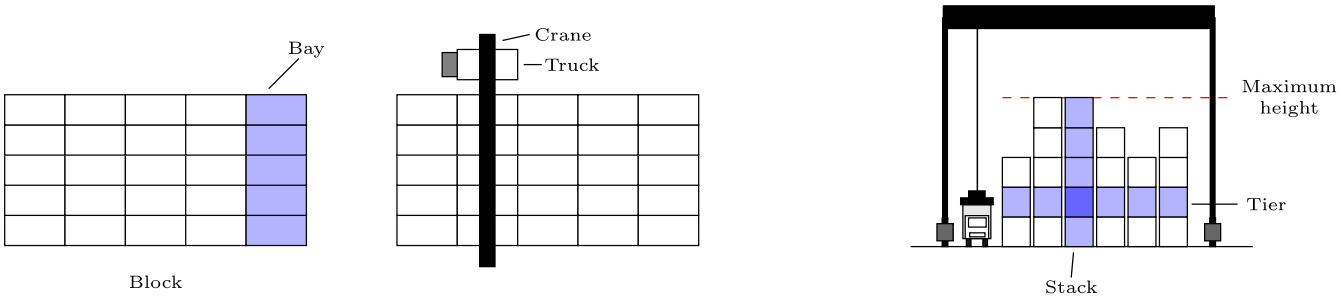
3. Problem description

The container yard is divided into several blocks consisting of parallel bays. Each bay has the same number of columns or stacks in which containers are piled up in several tiers. The maximum number of tiers or maximum number of containers each stack can hold is limited by the height of the cranes operating in the yard. As a consequence of stacking, in order to reach a specific container, it may be necessary to handle other containers above it. Fig. 1 shows a schematic representation of a container yard.

The container premarshalling problem with crane time minimization objective, CPMPCT, seeks to obtain the sequence of movements that requires the least crane time to arrange the containers located in a bay in the order in which they will be required later, considering that no container can leave the bay during that sequence.

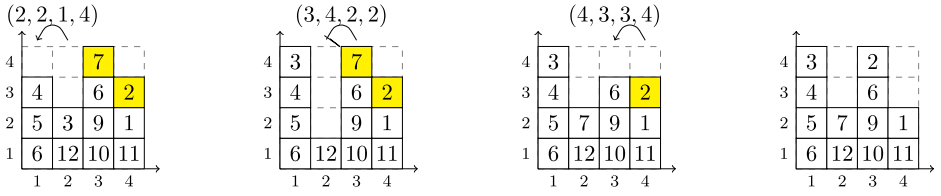
We assume the set $\mathcal{C}$ of all the containers placed in a bay to be the same size, therefore the bay can be seen as a matrix $T \times S$, where T represents the highest tier of the stacks and S the total number of stacks. Let $\mathcal{S}$ be the set of stacks in the bay and $\mathcal{T}$ the set of tiers. We describe each move by a quadruple (o, t, d, l), where $o, d \in \mathcal{S}$ and $t, l \in \mathcal{T}$. The pair (o, t) defines the origin of the move and (d, l) defines its destination. Therefore, a sequence of n moves is represented as $s = (o_1, t_1, d_1, l_1) \dots (o_n, t_n, d_n, l_n)$, $c_i \in \mathcal{C}$ being the container moved in (o_i, t_i, d_i, l_i). Since the order in which the containers leave the bay is known in advance, each container is assigned a number, its *group*, indicating the order in which it will be required. The set of container groups is denoted as $\mathcal{G} = \{1, \dots, G\}$, where containers in group 1 will be required first and those in group G last. The group of container c_i is indicated by function $group(c_i)$. The sequence s is a solution of the problem if the bay is sorted after the sequence. A bay is sorted if there are no blocking containers, i.e, containers blocking the removal of others. A container at a position (o, t) blocks the removal of others if its group is higher than that of any of the containers at lower positions, (o, t') such that $t' \leq t$. Fig. 2 shows an example of the solution of a premarshalling problem. The solution shown in the figure is optimal for both the CPMP and the CPMPCT. An alternative optimal solution for the CPMP is (2,2,1,4)(3,4,2,2)(4,3,2,3), which is not optimal for the CPMPCT.

We calculate the time needed for the crane to perform a movement following the procedure developed by Parreño-Torres et al. (2020). The crane specifications correspond to the Rubber Tyred Gantry (RTG) cranes used in the Noatum terminal in the port of Valencia in Spain. More specifically, they correspond to a



(a) Top view of two blocks. (b) Front view of a bay.

Fig. 1. Container yard scheme.



(a) Initial layout (b) Intermediate layout (c) Intermediate layout (d) Final layout

Fig. 2. Example solution $s = (2, 2, 1, 4)(3, 4, 2, 2)(4, 3, 3, 4)$ for the CPMPCT. Blocking containers are highlighted.

Konecranes RTG Transtainer 79. The authors distinguish between loaded/unloaded vertical and horizontal moves. The times are not proportional to the distances, as the acceleration of the crane is also taken into account. What is directly proportional to the tier in which the container is placed is the twistlock time, because the oscillation of the crane spreader depends on the length of the cable, due to the sway motion of the suspended load.

Given a solution $s = (o_1, t_1, d_1, l_1) \dots (o_n, t_n, d_n, l_n)$, the total time taken by the crane to perform the moves in solution s is calculated according to Eq. (1).

$$t(s) = h^0(o_1) + \sum_{i=2}^{n} h^0(d_{i-1}, o_i) + \sum_{i=1}^{n} v^1(t_i) + \sum_{i=1}^{n} h^1(o_i, d_i) + \sum_{i=1}^{n} v^0(l_i) \quad (1)$$

Consider the movement of a single container c_i from (o_i, t_i) to (d_i, l_i). First, the crane spreader moves horizontally along the upper path line from the destination stack of the previous move (d_{i-1}) to the corresponding stack (o_i); this time is represented by $h^0(d_{i-1}, o_i)$. It then moves down to reach the container in tier t_i, twistlocks it, and hoists it up to the upper travel line; this time is represented by $v^1(t_i)$. Once the container is at the upper travel line, the loaded crane moves horizontally from stack o_i to the destination stack (d_i), taking a time $h^1(o_i, d_i)$. Finally, the container is hoisted down to its new position in tier l_i and the spreader is positioned again at the upper travel line, all requiring a time $v^0(l_i)$. In addition to the moves required to change the positions of containers, the crane makes an initial move in which it moves horizontally along the upper travel line from its initial position, on the left of the first stack, to the origin of the first container to be moved. The time taken for this initial move is denoted as $h^0(o_1)$. A lower bound for the time spent to make a move, ϵ_t, consists of moving the crane to an adjacent stack, picking up a container from the topmost tier, and moving it to the topmost tier of an adjacent stack. Hence, $\epsilon_t := h^0(1, 2) + v^1(T) + h^1(2, 1) + v^0(T)$.

Let η be the total number of blocking containers, $\tau_1, \dots, \tau_\eta$, in the initial bay layout and t_{τ_i} the tier in which the blocking container τ_i is initially placed. We represent by η_t the lower bound for the time taken by the crane to retrieve and move them. The time needed to retrieve them is bounded below by considering that the unloaded crane moves horizontally η times to an adjacent stack, moves down to reach each of the blocking containers, twistlock it, and hoist it to the upper travel line. The time needed to move them to non-blocking positions is bounded below by considering that the loaded crane is moved horizontally η times to an adjacent stack, lowered to place each of the blocking containers in the highest tier to which it can be moved above non-blocking containers in the initial layout, and lifting the crane spreader to the upper travel line. Let us consider the example in Fig. 3(1a). Stacks 1, 2 and 3 each have one non-blocking container on tier 1 and two blocking containers on tiers 2 and 3. Overall, there are 6 blocking containers in the layout. To obtain a feasible solution, all blocking containers must be moved to non-blocking positions. Let us consider the move of the first blocking container. In the best case, the highest tier where it could be well placed above a non-blocking container is tier 2. Then there would be two non-blocking containers placed in one stack and one non-blocking container in the other two. The next blocking container could be moved to the stack with two non-blocking containers, on tier 3. Then, the highest tiers to which 2 of the remaining blocking containers could be moved to the same stack on tiers 4 and 5, and then, as this stack would be full, the last two blocking containers would go to tiers 2 and 3 in another stack. Therefore,

$$\eta_t = 6 \cdot \Big(h^0(1, 2) + h^1(1, 2)\Big) + \sum_{i=1}^{6} v^1(t_{\tau_i}) + \Big(2 \cdot v^0(2) + 2 \cdot v^0(3) + v^0(4) + v^0(5)\Big)$$

Let lb_m be the lower bound for the number of moves described in Tanaka et al. (2019). The first lower bound for the time spent by the crane proposed in Parreño-Torres et al. (2020), lb_t, is described in the following equation,

$$lb_t = \epsilon_t \cdot \Big(lb_m - \eta\Big) + \eta \cdot \Big(h^0(1, 2) + v^0(T) + h^1(1, 2)\Big) + \sum_{i=1}^{\eta} v^1(t_{\tau_i}) \quad (2)$$

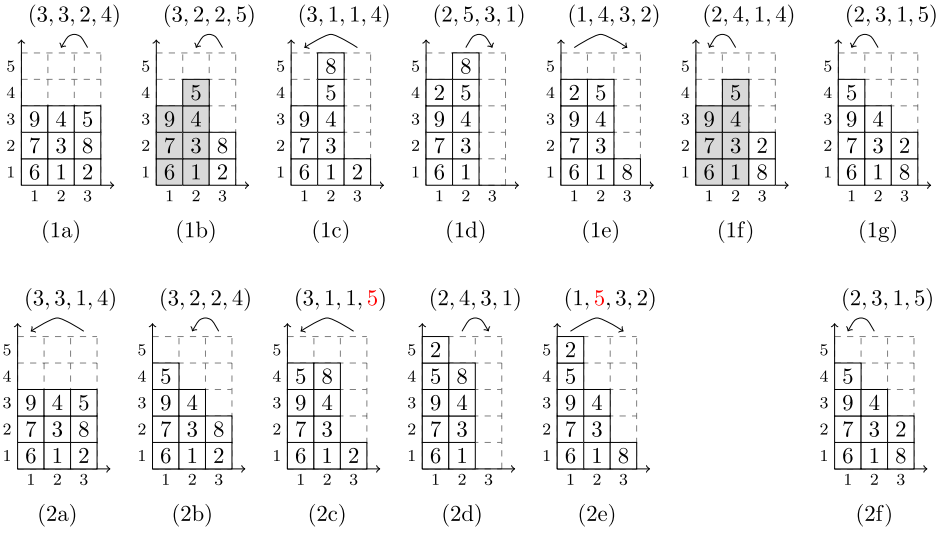


Fig. 3. Example of a sequence dominated by another, satisfying Proposition 3.

Table 1
Notation used for problem inputs.

$\mathcal{S}$	Set of stacks. $S := \|\mathcal{S}\|$ is the total number of stacks
$\mathcal{T}$	Set of tiers. $T := \|\mathcal{T}\|$ represents the highest tier of the stacks.
$\mathcal{G}$	Set of group values. $G := \|\mathcal{G}\|$ represents the highest group value and 1 the lowest group value.
$\mathcal{C}$	Set of containers. $C := \|\mathcal{C}\|$ is the total number of containers stored in the bay.
η	Number of blocking containers, $\tau_1, \ldots, \tau_\eta$, in the initial layout.
t_{τ_i}	Tier in which the blocking container τ_i is initially placed.
η_t	Lower bound for the time taken by the crane to move the blocking containers in the initial layout.
lb_m	Lower bound for the number of moves to solve the CPMP starting from the initial bay layout.
lb_t	Lower bound for the time taken by the crane to solve the CPMPCT from the initial bay layout.
ϵ_t	Lower bound for the time taken by the crane to perform a move.
$v^1(t)$	Time the crane takes to lower the spreader from the upper travel line to tier t of the container to be moved, twistlock the container and hoist it to the upper travel line.
$v^0(l)$	Time the crane takes to lower a container from the upper travel line to tier t, leave it and return the spreader to the upper travel line.
$h^1(o,d)$	Time taken by the crane to move a container along the upper travel line from stack o to stack d.
$h^0(d,o)$	Time taken by the crane to move the spreader (unloaded) from stack d to stack o.

Table 1 summarizes the notation already presented.

4. Dominance rules

The effectiveness of dominance rules in solving the CPMP has been well studied in the literature (Tanaka & Tierney, 2018; Tanaka et al., 2019; Tierney et al., 2017), showing a large reduction in the number of explored nodes when using tree search algorithms. In this section we review two dominance rules for the CPMPCT proposed in Parreño-Torres et al. (2020) and add nine new dominance rules.

Let δ be the Kronecker delta defined as $\delta_{x,y} = 0$ if $x \neq y$ and $\delta_{x,y} = 1$ if $x = y$. If y is a tuple $y = (y_1, y_2)$, then $\delta_{x,(y_1,y_2)} = 0$ if $x \neq y_1$ and $x \neq y_2$, and $\delta_{x,(y_1,y_2)} = 1$ if $x = y_1$ or $x = y_2$. Consider also the function $f(x_1, x_2, x_3, x_4)$ defined as follows:

$$f(x_1,x_2,x_3,x_4) = \begin{cases} x_1 - 1 & \text{If } x_2 = x_3 \\ x_1 + 1 & \text{If } x_2 = x_4 \\ x_1 & \text{Otherwise} \end{cases}$$

To formulate the proposed dominance rules, we consider the definition of stack invariant to a sequence of moves proposed by Tanaka et al. (2019).

Definition 1. A stack s is invariant to a sequence of moves $(o_1, t_1, d_1, l_1) \ldots (o_n, t_n, d_n, l_n)$, i.e, from move 1 to n, if its layout before and after the sequence of moves is the same and none of the containers placed in it have been moved by the sequence.

Let k be a stack invariant to the sequence $s = (o_1, t_1, d_1, l_1) \ldots (o_n, t_n, d_n, l_n)$. The maximum number of temporary containers that are simultaneously assigned to stack k during the sequence is:

$$t_k^{1,n}(s) := \max_{j \in \{1,\ldots,n\}} \left\{ \sum_{i=1}^{j} \left(\delta_{k,d_i} - \delta_{k,o_i} \right) \right\}$$

We represent as $n_{d_i}^n(s)$ the number of containers in stack d_i after the nth move of sequence s. In the propositions in this section, the first move of the sequences, (o_1, t_1, d_1, l_1), or the last move (o_n, t_n, d_n, l_n), or both, are not changed. They are included because the calculation of the crane time requires taking into account the crane movements from the previous position before the changes and to the next position after the changes.

4.1. Transitive movement dominance

This kind of dominance arises in sequences in which the same container is moved twice, once from stack a to stack b and then from stack b to stack c. Proposition 1 (Parreño-Torres et al., 2020) refers to the simplest case in which the same container undergoes two consecutive moves.

Proposition 1. *A sequence* $s_1 = (o_1, t_1, d_1, l_1) \ldots (o_i, t_i, d_i, l_i)(d_i, l_i, d_{i+1}, l_{i+1}) \ldots (d_n, l_n, d_n, l_n)$ *is dominated by the sequence* $s_2 = (o_1, t_1, d_1, l_1) \ldots (o_i, t_i, d_{i+1}, l_{i+1}) \ldots (d_n, l_n, d_n, l_n)$

However, more complex cases can be identified. Let us look at Fig. 3 illustrating Proposition 2 and consider the top-side sequence (1a)-(1g), $s_1 = (3,3,2,4)(3,2,2,5)$ (3,1,1,4)(2,5,3,1)(1,4,3,2) (2,4,1,4)(2,3,1,5), composed of 7 moves. The container with group value 5 is moved from stack 3 to stack 2 in move 1 and then from stack 2 to stack 1 in move 6. Stacks 1 and 2 are invariant to the sequence $s_1^* = (3,2,2,5)(3,1,1,4)(2,5,3,1)(1,4,3,2)$ (moves from 2 to 5) as can be seen in Fig. 3(1b) and (1f). Since the number of containers in stack 1 before the move (2,4,1,4) is 3, and there is only one temporary container assigned to stack 1 during s_1^*, this temporary container could still be assigned to stack 1 if it had one more container. Therefore, the same final configuration is obtained by sequence $s_2 = (3,3,1,4)(3,2,2,4)(3,1,1,5)(2,4,3,1)(1,5,3,2)(2,3,1,5)$ (Fig. 3(2a) and (2f)), as can be seen by comparing Fig. 3(1g) and (2f). Sequence s_2 in the example dominates sequence s_1, since the time taken by the crane to perform it is shorter. Note that if move (3,3,1,4) is made first (Fig. 3(2a)), instead of move (3,3,2,4) (Fig. 3(1a)), stack 2 will have one less container during the following moves (tiers highlighted in blue) and stack 1 will have one more container (tiers highlighted in red).

Proposition 2. *The sequence* $s_1 = (o_1,t_1,d_1,l_1)\dots(o_i,t_i,d_i,l_i)\dots(d_1,l_1,d_{n-1},l_{n-1})(o_n,t_n,d_n,l_n)$ *is dominated by* $s_2 = (o_1,t_1,d_{n-1},l_{n-1})\dots(o_i,t_i',d_i,l_i')\dots(o_{n-2},t_{n-2}',d_{n-2},l_{n-2}')(o_n,t_n,d_n,l_n)$ *in which* $t_i' := f(t_i,o_i,d_1,d_{n-1})$ *and* $l_i' := f(l_i,d_i,d_1,d_{n-1})$ *if the following conditions are satisfied:*

1. d_1 *and* d_{n-1} *are invariant from move 2 to* $n-2$ *in* s_1.
2. $n_{d_{n-1}}^{n-2}(s_1) + 1 + t_{d_{n-1}}^{2,n-2}(s_1) \le T$
3. $val_1 > val_2$, *where*

$$\begin{aligned} val_1 =& h^1(o_1,d_1) + h^0(d_1,o_2) + h^0(d_{n-2},d_1) + h^1(d_1,d_{n-1}) \\ &+ h^0(d_{n-1},o_n) + v^0(l_1) + v^1(l_1) \\ &+ \sum_{i=2}^{n-2}\left(\delta_{o_i,(d_1,d_{n-1})}\cdot v^1(t_i) + \delta_{d_i,(d_1,d_{n-1})}\cdot v^0(l_i)\right) \\ val_2 =& h^1(o_1,d_{n-1}) + h^0(d_{n-1},o_2) + h^0(d_{n-2},o_n) \\ &+ \sum_{i=2}^{n-2}\left(\delta_{o_i,(d_1,d_{n-1})}\cdot v^1(t_i') + \delta_{d_i,(d_1,d_{n-1})}\cdot v^0(l_i')\right) \end{aligned}$$

If the bay is sorted after $(d_1,l_1,d_{n-1},l_{n-1})$ *in sequence* s_1, *the same proposition holds without considering* (o_n,t_n,d_n,l_n) *in either* s_1 *or* s_2, *with* $h^0(d_{n-1},o_n) = h^0(d_{n-2},o_n) = 0$.

The idea of the next proposition is analogous: there is a container moved twice, from stack a to b and then from b to c. Here the origin stack of the first move of this container (stack a) and the stack to which the container is temporarily moved (stack b) are invariant to the sequence of moves between them, but the destination stack of the second move of the container (stack c) is not. The container can be moved directly from a to c after the sequence to which a and b are invariant. Starting from the initial bay layout in Fig. 3(1a), in sequence $s_1 =$ (1,3,3,4)**(1,2,2,4)**(3,4,1,2)(3,3,2,5)(1,2,3,3)(2,5,3,4)**(2,4,3,5)** a container is moved twice in moves 2 and 7 (in bold). It is dominated by $s_2 = (1,3,3,4)(3,4,1,3)(3,3,2,4)(1,3,3,3)(2,4,3,4)$ **(1, 2, 3, 5)** in which the container is moved once at the end of the sequence. However, in sequence s_2, move 2 simply reverses move 1, so s_2 is dominated in turn by (3,3,2,4)(1,3,3,3)(2,4,3,4)(1,2,3,5).

Proposition 3. *The sequence* $s_1 = (o_1,t_1,d_1,l_1)(o_2,t_2,d_2,l_2)\dots(o_i,t_i,d_i,l_i)\dots(d_2,l_2,d_n,l_n)$ *is dominated by sequence* $s_2 = (o_1,t_1,d_1,l_1)(o_3,t_3',d_3,l_3')\dots(o_i,t_i',d_i,l_i')\dots(o_2,t_2,d_n,l_n)$ *where* $t_i' := f(t_i,o_i,d_2,o_2)$ *and* $l_i' := f(l_i,d_i,d_2,o_2)$ *if the following conditions are satisfied:*

1. o_2 *and* d_2 *are invariant to moves 3 to* $n-1$ *in* s_1.
2. $n_{o_2}^{n-1}(s_1) + 1 + t_{o_2}^{3,n-1}(s_1) \le T$
3. $val_1 > val_2$, *where*

$$\begin{aligned} val_1 =& h^0(d_1,o_2) + h^1(o_2,d_2) + h^0(d_2,o_3) + h^0(d_{n-1},d_2) \\ &+ h^1(d_2,d_n) + v^0(l_2) + v^1(l_2) \\ &+ \sum_{i=3}^{n-1}\left(\delta_{o_i,(o_2,d_2)}\cdot v^1(t_i) + \delta_{d_i,(o_2,d_2)}\cdot v^0(l_i)\right) \\ val_2 =& h^0(d_1,o_3) + h^0(d_{n-1},o_2) + h^1(o_2,d_n) \\ &+ \sum_{i=3}^{n-1}\left(\delta_{o_i,(o_2,d_2)}\cdot v^1(t_i') + \delta_{d_i,(o_2,d_2)}\cdot v^0(l_i')\right) \end{aligned}$$

The proposition also holds if there is no move (o_1,t_1,d_1,l_1) *before* (o_2,t_2,d_2,l_2) *and the crane starts from its initial position, with* $h^0(d_1,o_2) = h^0(o_2)$ *and* $h^0(d_1,o_3) = h^0(o_3)$.

The last proposition in this section refers to sequences in which a container is moved twice, from stack a to stack b and then from b to c, but neither the origin stack of the first container move (stack a) nor the destination stack of the second container move (stack b) are invariant to the sequence between these moves. The container cannot be moved just once, but there could be another temporary stack c' to which the container could be moved, instead of b, with a shorter crane time. For instance, suppose there is a fourth empty stack in the bay layout shown in Fig. 3(1a). Sequence (3,3,4,1)(1,3,3,3)(4,1,1,3) moves a container twice, from stack 3 to stack 4 in move 1 and then from stack 4 to stack 1 in move 3. It is dominated by (3,3,2,4)(1,3,3,3)(2,4,1,3), in which the container in stack 3 is also moved twice, but first to stack 2 and then from stack 2 to stack 1. The final layout is the same but is reached in less time in the second sequence.

Proposition 4. *The sequence* $s_1 = (o_1,t_1,d_1,l_1)\dots(o_i,t_i,d_i,l_i)\dots(d_1,l_1,d_n,l_n)$ *is dominated by* $s_2 = (o_1,t_1,d,l')\dots(o_i,t_i',d_i,l_i')\dots(d,l',d_n,l_n)$ *where* $t_i' := f(t_i,o_i,d_1,d)$ *and* $l_i' := f(l_i,d_i,d_1,d)$ *if the following conditions are satisfied:*

1. d_1 *and* d *are invariant from move 2 to* $n-1$ *in* s_1.
2. $n_d^{n-1}(s_1) + 1 + t_d^{2,n-1}(s_1) \le T$
3. $val_1 > val_2$, *where*

$$\begin{aligned} val_1 =& h^1(o_1,d_1) + h^0(d_1,o_2) + h^0(d_{n-1},d_1) + h^1(d_1,d_n) \\ &+ v^0(l_1) + v^1(l_1) + \sum_{i=2}^{n-1}\left(\delta_{o_i,(d_1,d)}\cdot v^1(t_i) + \delta_{d_i,(d_1,d)}\cdot v^0(l_i)\right) \\ val_2 =& h^1(o_1,d) + h^0(d,o_2) + h^0(d_{n-1},d) + h^1(d,d_n) + \\ &+ v^0(l') + v^1(l') + \sum_{i=2}^{n-1}\left(\delta_{o_i,(d_1,d)}\cdot v^1(t_i') + \delta_{d_i,(d_1,d)}\cdot v^0(l_i')\right) \end{aligned}$$

4.2. Unrelated movement dominance

This kind of dominance arises in sequences in which one (or more) of the moves can be made in a different order, without altering the final layout of the bay and requiring less crane time. The simplest case involves moving a container from stack a to stack b and then another container from stack c to stack d. If stacks a, b, c, and d are all different and the sequence in which first the container is moved from c to d and then other container is moved from a to b takes less crane time, the former sequence is dominated by the latter. This simple case, and other more complex ones, are identified in Propositions 5 and 6. On the one hand, Proposition 5 identifies the dominance that occurs when a move can be made in an earlier position in the sequence. For instance, if Fig. 4a represents the initial bay layout, sequence (3,4,2,5)(3,3,2,6)(1,4,4,5)(3,2,4,6) is

(a) Layout to illustrate Propositions 5 and 6.

(b) Layout to illustrate Propositions 8 to 11.

Fig. 4. Bay layout examples.

dominated by (1,4,4,5)(3,4,2,5)(3,3,2,6)(3,2,4,6) in which the move that was previously the third is now the first. On the other hand, Proposition 6 identifies the dominance that occurs when a move can be made at a later position in the sequence. For instance, considering again the initial layout of a bay in Fig. 4a, the first move in the sequence (4,4,2,5)(1,4,4,4)(1,3,3,5)(4,4,3,6)(4,3,1,3) can be performed in fourth place, leading to the same layout and taking less crane time. Therefore, that sequence is dominated by the sequence (1,4,4,5)(1,3,3,5)(4,5,3,6)(4,4,2,5)(4,3,1,3).

Proposition 5. *The sequence* $s_1 = (o_1, t_1, d_1, l_1) \dots (o_i, t_i, d_i, l_i) \dots (o_n, t_n, d_n, l_n)$ *is dominated by* $s_2 = (o_1, t_1, d_1, l_1)(o_{n-1}, t_{n-1}, d_{n-1}, l_{n-1})(o_2, t'_2, d_2, l'_2) \dots (o_i, t'_i, d_i, l'_i) \dots (o_{n-2}, t'_{n-2}, d_{n-2}, l'_{n-2})(o_n, t_n, d_n, l_n)$ *where* $t'_i := f(t_i, o_i, o_{n-1}, d_{n-1})$ *and* $l'_i := f(l_i, d_i, o_{n-1}, d_{n-1})$ *if the following conditions are satisfied:*

1. o_{n-1} *and* d_{n-1} *are invariant from move 2 to* $n-2$ *in* s_1.
2. $n^{n-2}_{d_{n-1}}(s_1) + 1 + t^{2,n-2}_{d_{n-1}}(s_1) \le T$
3. $val_1 > val_2$, *where*

$$val_1 = h^0(d_1, o_2) + h^0(d_{n-2}, o_{n-1}) + h^0(d_{n-1}, o_n) + \sum_{i=2}^{n-2} \left(\delta_{o_i,(o_{n-1},d_{n-1})} \cdot v^1(t_i) + \delta_{d_i,(o_{n-1},d_{n-1})} \cdot v^0(l_i) \right)$$

$$val_2 = h^0(d_1, o_{n-1}) + h^0(d_{n-1}, o_2) + h^0(d_{n-2}, o_n) + \sum_{i=2}^{n-2} \left(\delta_{o_i,(o_{n-1},d_{n-1})} \cdot v^1(t'_i) + \delta_{d_i,(o_{n-1},d_{n-1})} \cdot v^0(l'_i) \right)$$

The proposition also holds if there is no move (o_1, t_1, d_1, l_1) *before* (o_2, t_2, d_2, l_2) *and the crane starts from its initial position, with* $h^0(d_1, o_2) = h^0(o_2)$ *and* $h^0(d_1, o_{n-1}) = h^0(o_{n-1})$. *Moreover, if the bay is sorted after move* $(o_{n-1}, t_{n-1}, d_{n-1}, l_{n-1})$ *in sequence* s_1, *the same proposition holds without considering* (o_n, t_n, d_n, l_n) *either in* s_1 *or in* s_2, *with* $h^0(d_{n-1}, o_n) = h^0(d_{n-2}, o_n) = 0$.

Proposition 6. *The sequence* $s_1 = (o_1, t_1, d_1, l_1) \dots (o_i, t_i, d_i, l_i) \dots (o_n, t_n, d_n, l_n)$ *is dominated by* $s_2 = (o_1, t_1, d_1, l_1)(o_3, t'_3, d_3, l'_3) \dots (o_i, t'_i, d_i, l'_i) \dots (o_{n-1}, t'_{n-1}, d_{n-1}, l'_{n-1})(o_2, t_2, d_2, l_2)(o_n, t_n, d_n, l_n)$ *where* $t'_i := f(t_i, o_i, d_2, o_2)$ *and* $l'_i := f(l_i, d_i, d_2, o_2)$ *if the following conditions are satisfied:*

1. o_2 *and* d_2 *are invariant from move 3 to* $n-1$ *in* s_1.
2. $n^{n-1}_{o_2}(s_1) + 1 + t^{3,n-1}_{o_2}(s_1) \le T$
3. $val_1 > val_2$, *where*

$$val_1 = h^0(d_1, o_2) + h^0(d_2, o_3) + h^0(d_{n-1}, o_n) + \sum_{i=3}^{n-1} \left(\delta_{o_i,(o_2,d_2)} \cdot v^1(t_i) + \delta_{d_i,(o_2,d_2)} \cdot v^0(l_i) \right)$$

$$val_2 = h^0(d_1, o_3) + h^0(d_{n-1}, o_2) + h^0(d_2, o_n) + \sum_{i=3}^{n-1} \left(\delta_{o_i,(o_2,d_2)} \cdot v^1(t'_i) + \delta_{d_i,(o_2,d_2)} \cdot v^0(l'_i) \right)$$

The proposition also holds if there is no move (o_1, t_1, d_1, l_1) *before* (o_2, t_2, d_2, l_2) *and the crane starts from its initial position, with* $h^0(d_1, o_2) = h^0(o_2)$ *and* $h^0(d_1, o_3) = h^0(o_3)$. *Moreover, if the bay is sorted after move* $(o_{n-1}, t_{n-1}, d_{n-1}, l_{n-1})$ *in sequence* s_1, *the same proposition holds without considering* (o_n, t_n, d_n, l_n) *either in* s_1 *or in* s_2, *with* $h^0(d_{n-1}, o_n) = h^0(d_2, o_n) = 0$.

4.3. Same group movement dominance

This kind of dominance arises in sequences in which two containers in the same group are relocated in two different moves. A simple case is described in Proposition 7 (Parreño-Torres et al., 2020), in which a container with group value g is moved first from one stack a to another stack b, and in the next move, a container with the same group value g is moved to stack a from another stack c. The same layout is obtained with a lower crane time by moving a container with group value g directly from c to b.

Proposition 7. *A sequence* $s_1 = (o_1, t_1, d_1, l_1) \dots (o_i, t_i, d_i, l_i)(o_{i+1}, t_{i+1}, o_i, d_i) \dots (o_n, t_n, d_n, l_n)$ *such that* $group(c_i) = group(c_{i+1})$ *is dominated by sequence* $s_2 = (o_1, t_1, d_1, l_1) \dots (o_{i-1}, t_{i-1}, d_{i-1}, l_{i-1})(o_{i+i}, t_{i+1}, d_i, l_i)(o_{i+2}, t_{i+2}, d_{i+2}, l_{i+2}) \dots (o_n, t_n, d_n, l_n)$

Propositions 8 and 9 generalize this for cases where there is an intermediate sequence between movements from a to b and from c to a. If stacks c and a are both invariant to the intermediate sequence, the containers temporarily assigned to a during that intermediate sequence still fit with one more container, and the total time taken by the crane is lower, it is better to move a container with group value g from stack c to b and then perform the intermediate sequence (see Proposition 8). For example, consider the initial bay layout in Fig. 4b, with $a = 2$, $b = 5$, and $c = 3$. The sequence (2,6,5,4)(4,2,5,5)(3,4,2,6)(4,1,3,4) is dominated by the sequence (3,4,5,4)(4,2,5,5)(4,1,3,4). If stacks a and b are invariant, the containers temporarily assigned to a during that intermediate sequence still fit with one more container, and the total time taken by the crane is lower, it is better to perform the intermediate sequence and then to move a container with group value g from stack c to b, (see Proposition 9). Considering again the initial layout in Fig. 4b, with $a = 1$, $b = 4$, and $c = 3$, the sequence (4,2,5,4)(1,6,4,2)(3,4,5,5)(3,3,1,6)(2,6,3,3) is dominated by the sequence (4,2,5,4)(3,4,5,5)(3,3,4,2)(2,6,3,3).

Proposition 8. *The sequence* $s_1 = (o_1, t_1, d_1, l_1)(o_2, t_2, d_2, l_2) \dots (o_i, t_i, d_i, l_i) \dots (o_{n-1}, t_{n-1}, o_2, t_2)(o_n, t_n, d_n, l_n)$ *such that* $group(c_2) = group(c_{n-1})$ *is dominated by sequence* $s_2 = (o_1, t_1, d_1, l_1)(o_{n-1}, t_{n-1}, d_2, l_2) \dots (o_i, t'_i, d_i, l'_i) \dots (o_{n-2}, t'_{n-2}, d_{n-2}, l'_{n-2})(o_n, t_n, d_n, l_n)$ *where* $t'_i := f(t_i, o_i, o_{n-1}, o_2)$ *and* $l'_i := f(l_i, d_i, o_{n-1}, o_2)$ *if the following conditions are satisfied:*

1. o_2 *and* o_{n-1} *are invariant from move 3 to* $n-2$ *in* s_1.
2. $n^{n-2}_{o_2}(s_1) + 1 + t^{3,n-2}_{o_2}(s_1) \le T$

3. $val_1 > val_2$, where

$$val_1 = h^0(d_1, o_2) + h^1(o_2, d_2) + h^0(d_{n-2}, o_{n-1}) + h^1(o_{n-1}, o_2) + h^0(o_2, o_n) + v^0(t_2) + v^1(t_2) + \sum_{i=3}^{n-2}\left(\delta_{o_i,(o_2,o_{n-1})} \cdot v^1(t_i) + \delta_{d_i,(o_2,o_{n-1})} \cdot v^0(l_i)\right)$$

$$val_2 = h^0(d_1, o_{n-1}) + h^1(o_{n-1}, d_2) + h^0(d_{n-2}, o_n) + \sum_{i=3}^{n-2}\left(\delta_{o_i,(o_2,o_{n-1})} \cdot v^1(t'_i) + \delta_{d_i,(o_2,o_{n-1})} \cdot v^0(l'_i)\right)$$

The proposition also holds if there is no move (o_1, t_1, d_1, l_1) before (o_2, t_2, d_2, l_2) and the crane starts from its initial position, with $h^0(d_1, o_2) = h^0(o_2)$ and $h^0(d_1, o_{n-1}) = h^0(o_{n-1})$. Moreover, if the bay is sorted after move $(o_{n-1}, t_{n-1}, o_2, t_2)$ in sequence s_1, the same proposition holds without considering (o_n, t_n, d_n, l_n) either in s_1 or in s_2, with $h^0(o_2, o_n) = h^0(d_{n-2}, o_n) = 0$.

Proposition 9. *The sequence $s_1 = (o_1, t_1, d_1, l_1)(o_2, t_2, d_2, l_2)\dots(o_i, t_i, d_i, l_i)\dots(o_{n-1}, t_{n-1}, o_2, t_2)(o_n, t_n, d_n, l_n)$ such that $group(c_2) = group(c_{n-1})$ is dominated by sequence $s_2 = (o_1, t_1, d_1, l_1)(o_3, t'_3, d_3, l'_3)\dots(o_i, t'_i, d_i, l'_i)\dots(o_{n-1}, t_{n-1}, d_2, l_2)(o_n, t_n, d_n, l_n)$ where $t'_i := f(t_i, o_i, d_2, o_2)$ and $l'_i := f(l_i, d_i, d_2, o_2)$ if the following conditions are satisfied:*

1. *o_2 and d_2 are invariant from move 3 to $n-2$ in s_1.*
2. *$n^{n-2}_{o_2}(s_1) + 1 + t^{3,n-2}_{o_2}(s_1) \le T$*
3. *$val_1 > val_2$, where*

$$val_1 = h^0(d_1, o_2) + h^1(o_2, d_2) + h^0(d_2, o_3) + h^1(o_{n-1}, o_2) + h^0(o_2, o_n) + v^0(t_2) + v^1(t_2) + \sum_{i=3}^{n-2}\left(\delta_{o_i,(o_2,d_2)} \cdot v^1(t_i) + \delta_{d_i,(o_2,d_2)} \cdot v^0(l_i)\right)$$

$$val_2 = h^0(d_1, o_3) + h^1(o_{n-1}, d_2) + h^0(d_2, o_n) + \sum_{i=3}^{n-2}\left(\delta_{o_i,(o_2,d_2)} \cdot v^1(t'_i) + \delta_{d_i,(o_2,d_2)} \cdot v^0(l'_i)\right)$$

The proposition also holds if there is no move (o_1, t_1, d_1, l_1) before (o_2, t_2, d_2, l_2) and the crane starts from its initial position, with $h^0(d_1, o_2) = h^0(o_2)$ and $h^0(d_1, o_3) = h^0(o_3)$. Moreover, if the bay is sorted after move $(o_{n-1}, t_{n-1}, o_2, t_2)$ in sequence s_1, the same proposition holds without considering (o_n, t_n, d_n, l_n) either in s_1 or in s_2, with $h^0(o_2, o_n) = h^0(d_2, o_n) = 0$.

The following propositions consider the movement of containers with the same group value in two distinct moves, that is, a container with group value g is moved from a to b and, after a sequence of moves, another container with group value g is moved from c to d. If the origin stacks of the moves, a and c, are invariant to the intermediate sequence and the containers that are temporarily allocated to a still fit if there is one more container, the origin stacks can be swapped, resulting in the same layout. Therefore, the sequence that takes less crane time dominates the other (see Proposition 10). For example, considering the layout in Fig. 4b and $a = 3$, $b = 5$, $c = 2$, and $d = 5$, the sequence (3,4,5,4)(4,2,5,5)(1,6,4,2)(2,6,5,6) is dominated by the sequence (2,6,5,4)(4,2,5,5)(1,6,4,2)(3,4,5,6). Similarly, if the destination stacks of the moves, b and d, are invariant to the intermediate sequence and the containers that are temporarily allocated in d still fit if there is one more container, the destination stacks can be swapped, resulting in the same layout. Therefore, the sequence that takes less crane time dominates the other sequence (see Proposition 11). For example, considering the layout in Fig. 4b and $a = 4$, $b = 5$, $c = 3$, and $d = 2$, the sequence (2,6,5,4)(4,2,5,5)(1,6,4,2)(3,4,2,6)(4,2,1,6) is dominated by the sequence (2,6,5,4)(4,2,2,6)(1,6,4,2)(3,4,5,5)(4,2,1,6).

Proposition 10. *The sequence $s_1 = (o_1, t_1, d_1, l_1)(o_2, t_2, d_2, l_2)\dots(o_i, t_i, d_i, l_i)\dots(o_n, t_n, d_n, l_n)$ such that $group(c_2) = group(c_n)$ is dominated by sequence $s_2 = (o_1, t_1, d_1, l_1)(o_n, t_n, d_2, l_2)\dots(o_i, t'_i, d_i, l'_i)\dots(o_2, t_2, d_n, l_n)$ where $t'_i := f(t_i, o_i, o_n, o_2)$ and $l'_i := f(l_i, d_i, o_n, o_2)$ if the following conditions are satisfied:*

1. *o_2 and o_n are invariant from move 3 to $n-1$ in s_1.*
2. *$n^{n-1}_{o_2}(s_1) + 1t^{3,n-1}_{o_2}(s_1) \le T$*
3. *$val_1 > val_2$, where*

$$val_1 = h^0(d_1, o_2) + h^1(o_2, d_2) + h^0(d_{n-1}, o_n) + h^1(o_n, d_n) + \sum_{i=3}^{n-1}\left(\delta_{o_i,(o_2,o_n)} \cdot v^1(t_i) + \delta_{d_i,(o_2,o_n)} \cdot v^0(l_i)\right)$$

$$val_2 = h^0(d_1, o_n) + h^1(o_n, d_2) + h^0(d_{n-1}, o_2) + h^1(o_2, d_n) + \sum_{i=3}^{n-1}\left(\delta_{o_i,(o_2,o_n)} \cdot v^1(t'_i) + \delta_{d_i,(o_2,o_n)} \cdot v^0(l'_i)\right)$$

The proposition also holds if there is no move (o_1, t_1, d_1, l_1) before (o_2, t_2, d_2, l_2) and the crane starts from its initial position, with $h^0(d_1, o_2) = h^0(o_2)$ and $h^0(d_1, o_n) = h^0(o_n)$.

Proposition 11. *The sequence $s_1 = (o_1, t_1, d_1, l_1)\dots(o_i, t_i, d_i, l_i)\dots(o_{n-1}, t_{n-1}, d_{n-1}, l_{n-1})\ (o_n, t_n, d_n, l_n)$ such that $group(c_1) = group(c_{n-1})$ is dominated by the sequence $s_2 = (o_1, t_1, d_{n-1}, l_{n-1})\dots(o_i, t'_i, d_i, l'_i)\dots(o_{n-1}, t_{n-1}, d_1, l_1)(o_n, t_n, d_n, l_n)$ where $t'_i := f(t_i, o_i, d_1, d_{n-1})$ and $l'_i := f(l_i, d_i, d_1, d_{n-1})$ if the following conditions are satisfied:*

1. *d_1 and d_{n-1} are invariant from move 2 to $n-2$ in s_1.*
2. *$n^{n-2}_{d_{n-1}}(s_1) + 1 + t^{2,n-2}_{d_{n-1}}(s_1) \le T$*
3. *$val_1 > val_2$, where*

$$val_1 = h^1(o_1, d_1) + h^0(d_1, o_2) + h^1(o_{n-1}, d_{n-1}) + h^0(d_{n-1}, o_n) + \sum_{i=2}^{n-2}\left(\delta_{o_i,(d_1,d_{n-1})} \cdot v^1(t_i) + \delta_{d_i,(d_1,d_{n-1})} \cdot v^0(l_i)\right)$$

$$val_2 = h^1(o_1, d_{n-1}) + h^0(d_{n-1}, o_2) + h^1(o_{n-1}, d_1) + h^0(d_1, o_n) + \sum_{i=2}^{n-2}\left(\delta_{o_i,(d_1,d_{n-1})} \cdot v^1(t'_i) + \delta_{d_i,(d_1,d_{n-1})} \cdot v^0(l'_i)\right)$$

If the bay is sorted after move $(o_{n-1}, t_{n-1}, d_{n-1}, l_{n-1})$ in sequence s_1, the same proposition holds without considering (o_n, t_n, d_n, l_n) either in s_1 or in s_2, with $h^0(d_{n-1}, o_n) = h^0(d_1, o_n) = 0$.

5. A beam search-based algorithm

The Beam Search algorithm (BS) uses breadth-first search (BFS) to explore the solution tree. Unlike standard BFS, the BS algorithm only keeps a reduced set of β promising nodes at each level, where β is known as beam width. The remaining nodes are permanently discarded. Consequently, the path to the optimal solution could be pruned by discarding the node leading to it, in which case the optimality of the best solution found is not guaranteed. The selection of nodes is usually done in two steps. First, a fast local evaluation selects a subset of nodes from among all those generated at a given level. Then a more time-consuming global evaluation selects from this subset the β promising nodes to keep at this level.

Beam search algorithms have been very successful in related problems in which solutions are represented by sequences of moves, such block relocation (Bacci, Mattia, & Ventura, 2019) or cutting and packing problems (Libralesso & Fontan, 2021; Parreño, Alonso, & Alvarez-Valdes, 2020). For other variants of premarshalling problems, similar tree search structures are the basis

Table 2
Main notation used throughout the algorithms in Section 5.

β	Beam width.
π	Best solution found.
$u(\pi)$	Upper bound on the number of moves to solve the CPMPCT.
s^w	Sequence performed to reach node w.
$m(s)$	Number of moves in sequence s.
$t(s)$	Time taken by the crane to perform sequence s.
$\eta(w)$	Number of blocking containers in the layout of node w.
$lb_m(w)$	Lower bound for the number of moves to solve the CPMP from the layout of node w.
$lb_t(w)$	Lower bound for the time taken by the crane to solve the CPMPCT from the layout of node w.
$CPU_{\max}$	Maximum computing time.

of recent high-performing algorithms (Hottung et al., 2020). In this section, we describe the proposed algorithm based on the beam search structure, along with the node comparison criteria and the global evaluation algorithm used. Table 2 summarizes the main notation used throughout the section.

5.1. Algorithm description

Algorithm 1 shows the pseudocode of our proposed algorithm. It iteratively runs a beam search algorithm (lines 11 to 19), increasing the beam width β at each iteration. The algorithm ends when the time limit is reached or an optimal solution is found. A solution is optimal if during one iteration of the beam search, the number of nodes generated at each depth is less than or equal to the current beam width.

Algorithm 1 Pseudocode of the algorithm based on beam search.

```
1: function BeamSearch(root)
2:     π ← ∅                                          ▷ Best solution found
3:     β ← S                                          ▷ Initial beam width
4:     opt ← false    ▷ Flag to indicate if the optimal solution has been reached
5:     u ← 1.5 · lb_m                                 ▷ Upper bound on the number of moves.
6:     while opt = false and time < CPU_max do
7:         depth ← 1                                  ▷ Depth
8:         opt ← true
9:         list.push_back(root)
10:        sublist ← ∅
11:        while list ≠ ∅ and time < CPU_max do
12:            w ← list.front()
13:            list.pop_front()
14:            sublist ← Descendants(depth, w, sublist, π, u)
15:                    ▷ All non-dominated neighbours from node w
16:            if list ≠ ∅ then
17:                depth ← depth + 1
18:                list ← GlobalEval(sublist, β, π, depth, u)
19:                if list.size() = β then opt ← false
20:                sublist ← ∅
21:        if π = ∅ then u ← 1.5 · u
22:        β ← 1.5 · β
23:    if time ≥ CPU_max then opt ← false
24: return π
```

The beam width is initially set to the number of stacks S in the bay (line 3) and increased 1.5 times on each iteration (line 21). This increment is in line with what was reported by Libralesso & Fontan (2021). Throughout the algorithm, a parameter u is used to limit the maximum number of levels to explore. Initially, u is set to 1.5 times the lower bound on the number of moves at the root node *root* (line 5). As this initial value is only an estimation, it must be increased if no possible solution is found (line 20), and is updated to a valid bound for the number of moves as soon as a feasible solution is obtained. Moreover, the value of u is also updated each time a better feasible solution is obtained to reduce the solution space. Let $t(s^\pi)$ be the time taken by the crane to perform sequence s^π, η the number of blocking containers in the initial bay layout, η_t a lower bound for the time taken by the crane to move them, and ϵ_t a lower bound for the time taken to perform a move. Eq. (3) provides a valid upper bound on the number of moves to solve the CPMPCT as shown in Parreño-Torres et al. (2020).

$$u := u(\pi) = \left\lfloor \frac{t(s^\pi) - \eta^t + \eta \cdot \epsilon_t}{\epsilon_t} \right\rfloor \tag{3}$$

Each of the beam search iterations starts from the *root* node, which corresponds to the initial layout of the bay (line 9). The Descendants() function generates all non-dominated descendants of node w at line 14. A node is a descendant of another node if it is obtained after the movement of a single container. Moreover, a node $\bar{w}$ is dominated if one of the following conditions holds:

1. At least one of the criteria described in Section 4 is satisfied. Thus, the sequence of movements performed to reach the node is dominated by another sequence leading to the same layout and requiring less crane time.
2. The lower bound for the number of moves to solve the CPMP at node $\bar{w}$ plus the number of moves to reach that node is greater than the current upper bound on the number of moves: $lb^m(\bar{w}) + m(s^{\bar{w}}) > u$.

Non-dominated nodes are sorted into four groups in *sublist* depending on the type of the last move performed: bad-bad (BB), good-bad (GB), good-good (GG), or bad-good (BG). This classification was introduced by Bortfeldt (2004). Bad-bad moves are those in which a blocking container moves to a stack in which it also blocks the removal of others. Bad-good moves refer to moves in which a blocking container is moved to a stack in which it is no longer blocking. Similarly, good-bad and good-good moves involve moving of a non-blocking container to a position where it blocks the removal of others and to a position where it does not, respectively. Different tie-breaking criteria are used for sorting the nodes within each group as described in Section 5.2.

Once all of the descendants at *depth* have been generated in *sublist*, we move to the global evaluation in line 17. It is carried out by the GlobalEval() function, which runs a heuristic algorithm to obtain feasible solutions as well as a local search algorithm to improve them. This function returns the most promising β nodes stored in *list*. Algorithms 2, 3, and 4 show the pseudocode of the global evaluation, the heuristic algorithm, and the local search. They all are further described in Section 5.3. If the number of nodes contained in *list* after line 17 equals β, the optimality of the solution reached cannot be guaranteed and therefore flag *opt* is set to *false* in line 18. List *sublist* is cleared at line 19 as it will be used again to store the descendants of the beam nodes at *depth*, that is, the descendants of the (at most) β nodes stored in *list*. If the beam search ends and no solution has been found, the upper bound on the number of moves is increased in line 20. If

Algorithm 2 Global evaluation algorithm.

```
1: function GlobalEval(sublist, β, π, depth, u)
2:     list ← ∅
3:     β_prior ← α% · β
4:     count ← 0
5:     while count < β_prior and sublist ≠ ∅ do
6:         for i in {BG,GG,BB,GB} do
7:             w̄ = sublist[i].front() ▷ Take the first element of each sublist
8:             if depth + lb_m(w̄) ≤ u then
9:                 count ← count + 1
10:                w̄.priority ← true
11:                w̄.identifier ← count
12:                w̄.heur ← Heuristic(w̄, π, d)
13:                list ← InsertNode(list, β, w̄)
14:            sublist[i].pop_front()
15:    for i in {BG,GG,BB,GB} do
16:        count ← 0
17:        for w̄ = sublist[i].begin() to sublist[i].end() do ▷ Take all the elements of each sublist
18:            if depth + lb_m(w̄) ≤ u then
19:                count ← count + 1
20:                w̄.priority ← false
21:                w̄.identifier ← count
22:                w̄.heur ← Heuristic(w̄, π, u, d)
23:                list ← InsertNode(list, β, w̄)
24: return list
```

Algorithm 3 A heuristic algorithm to obtain feasible solutions.

```
1: function Heuristic(w̄, π, u, depth) ▷ Partial solution
2:     if π = ∅ then c ← depth else c ← 0
3:     flag ← true
4:     s ← s^w̄ and w ← w̄
       ▷ Let mt/mm be the maximum improvement in time/number of moves achieved by the local search.
5:     while t(s) + ε_t · η(w) − mt < t(s^π) and m(s) + η(w) − mm − c < u and flag = true do
6:         if ∃ (o, t, d, l) in BG moves thenAdd (o, t, d, l) to s and Update w and flag ← true
7:         else if ∃ (o, t, d, l) in GG moves then Add (o, t, d, l) to s and Update w and flag ← true
8:         else EmptyStack(w) Add moves to s and Update w and flag ← true
9:         if η(w) = 0 then
10:            s ← LS(s, d)
11:            flag ← false
12:            if t(s) < t(s^π) then Update(π) ▷ Update the best solution so far
13:    if η(w) ≠ 0 then
14:        t ← ∞
15:    else
16:        t ← t(s)
17: return t
```

neither the time limit nor the optimal solution has been reached, a new beam search algorithm is run with an increased beam width updated in line 21.

5.2. Node comparison criteria

Non-dominated descendants of the beam nodes are divided into four groups according to the type of the last move made: BB, BG, GG, and GB. We consider 11 different tie-breaking criteria to order the nodes. They are evaluated one by one in order until the tie is broken. Within each group, only the appropriate criteria are used to evaluate the type of move performed in it. Table 3 shows all the criteria defined, the order in which they are evaluated, and the groups in which they are (✓) and are not used (-).

Algorithm 4 Improvement phase.

```
1: function LS(s, depth)
2:     s_0 ← ∅ ▷ Empty sequence
3:     for i = depth + 1 to m(s) do
4:         Let (o_i, t_i, d_i, l_i) be movement i of sequence s
5:         s_0 ← Add movement (o_i, t_i, d_i, l_i) to sequence s_0
6:         better ← true
7:         while better do
8:             better ← false
9:             if Direct dominance then better ← true and Modify sequence s_0
10:            if better = false and Proposition 5 or 11 then better ← true and Modify sequence s_0
11:            if better = false and Proposition 6, 8, 9, or 10 then better ← true and Modify sequence s_0
12:            if better = false and Proposition 2 then better ← true and Modify sequence s_0
13:            if better = false and Proposition 3 then better ← true and Modify sequence s_0
14:            if better = false and Proposition 4 then better ← true and Modify sequence s_0
15:    s ← s_0
16:    for i = m(s) to 1 do
17:        Let (o_i, t_i, d_i, l_i) be movement i of sequence s
18:        if ∃ d not in o_{i+1}, ..., o_{m(s)}, in which the container is well placed at tier l with a lower crane time then
19:            Replace (o_i, t_i, d_i, l_i) by (o_i, t_i, d, l) in sequence s
20: return s
```

1. *Previous movement type*. In the BB group, first the nodes whose last two moves are BB, with identical origin stack. In the GB group, first the nodes whose last two moves are BB or GB, with identical origin stack. The underlying idea is that if one container is moved from one stack s to another where it blocks the removal of others, it is in order to free a lower position in stack s so that other containers can be correctly allocated. Once such a move has been made from s, it is convenient to continue with this type of move until the required position is freed, even if the move does not reduce the lower bound.
2. *Lower bound for the moves.* First, the nodes with the lowest lower bound for the number of moves.
3. *Containers to unload from the origin stack.* First, the nodes with the lowest number of containers to be unloaded from the origin stack s such that it would be possible to place a container with group g in this stack, where g is the highest group between the topmost containers in the blocking stacks and the blocking containers in stack s. This criterion applies only to BB and GB and is considered for the explanation given for criterion 1. The fewer containers that need to be removed from one stack to free up the desired position, the better.
4. *Containers in the origin stack.* First, the nodes with the lowest number of containers in the origin stack. This criterion applies only to BB and GB and is considered for the explanation given for criterion 1. The fewer containers the stack has, the more containers will fit above the position to be freed.
5. *Containers placed upside down.* First, the nodes in which a container is moved to a stack in which it is placed on top of a container with a lower group value (these containers are said to

Table 3
Tie-breaking criteria to order the nodes. A tick indicates that the criterion applies.

Order	Criterion	BB	GB	BG	GG
1	Previous move type	✓	✓	-	-
2	Lower bound for the number of moves	✓	✓	✓	✓
3	Containers to unload from the origin stack	✓	✓	-	-
4	Containers in the origin stack	✓	✓	-	-
5	Container placed upside down	✓	✓	-	-
6	Maximum blocking group value of the origin stack	✓	-	✓	-
7	Gap topmost destination - topmost origin	-	-	-	✓
8	Gap topmost destination - container moved	-	✓	-	-
9	Topmost container in the origin stack	-	✓	-	✓
10	Lower bound for the time spent by the crane	✓	✓	✓	✓
11	Crane time spent so far	✓	✓	✓	✓

be placed "upside down"). If two containers are upside down, they are accessible to be resorted correctly in another stack. This criterion applies only to BB and GB, where the container will block the removal of others in the destination stack.

6. *Maximum group value of a blocking container in the origin stack.* First, the nodes with the highest group value of a blocking container. This criterion applies only to BB and BG where the container being moved is a blocking container.
7. *Gap between the topmost destination and the topmost origin.* First, the nodes with the smallest difference between the group value of the topmost container in the destination stack before the move and the group value of the topmost container in the origin stack. This criterion applies to GG. The smaller the difference between consecutive containers in a well-sorted stack, the better.
8. *Gap between the topmost destination and the container being moved.* First, the nodes with the smallest difference between the group value of the topmost container in the destination stack before the move and the group value of the container being moved. This criterion applies to BG; the smaller the difference, the greater the flexibility to include new containers.
9. *Topmost container in the origin stack.* First, the nodes with the largest container group value of the topmost container in the origin stack. This criterion applies to GB and GG; the larger the value, the wider the range of group values that could be accepted on top.
10. *Lower bound for the time spent by the crane.* First, the nodes with the lowest lower bound for the time required by the crane to rearrange the bay.
11. *Crane time spent so far.* First, the nodes with the lowest crane time taken by the crane to reach the node.

5.3. Global evaluation approach

The global evaluation phase is carried out using the GlobalEval() function. Our global evaluation basically consists in completing the sequence of moves of a node until a feasible solution is found, using the HEURISTIC() function, with two specific features. On the one hand, a given percentage of nodes belonging to each class is kept to ensure diversity. On the other, the heuristic algorithm does not go on indefinitely, but stops as soon as it becomes clear that a good solution will not be found.

The GlobalEval() function is outlined in Algorithm 2. Let *sublist* be a vector of four lists, each containing the nodes of one group (BG, GG, BB, or GB). This function goes through the nodes in each list and evaluates those that are not dominated by number of moves or by crane time (see lines 8 and 18). The first nodes of each group will be added to the beam's node list until α% of the beam is occupied (see lines 5 to 14). The remaining (100-α)% will be occupied by any of the remaining nodes. To discriminate between the nodes that are in the α% and the remaining nodes in lines 10 and 20, a *priority* variable is used. We assign each node an identifier with the order in which it is added on line 11 and with its position in *sublist*[*i*] on line 21. Next, we try to obtain a feasible solution using the Heuristic() function described in Section 5.3.1. The time it takes for the crane to rearrange the bay is stored in $\bar{w}$.*heur* in lines 12 and 23. This value is unbounded if a feasible solution has not been found. Using InsertNode(), node $\bar{w}$ is inserted in *list* where the nodes are ordered according to the following criteria, which are evaluated in order until ties are broken:

1. The highest priority value.
2. The lowest lower bound for the number of moves to solve the CPMP.
3. The lowest value returned by Heuristic().
4. The lowest position it occupies within its corresponding list in *sublist*.
5. The lowest crane time it takes for the crane to reach the node, plus the lower bound for the time the crane still requires to rearrange the bay.

The InsertNode() function returns only the β most promising nodes, discarding the remaining ones.

5.3.1. A heuristic algorithm to obtain a feasible solution

Algorithm 3 shows the pseudocode of the heuristic used in the global evaluation. It makes moves starting from the partial solution $\bar{w}$ for as long as the following three conditions are satisfied:

1. There still are blocking containers.
2. The estimate of the time required to obtain a feasible solution does not exceed the crane time of the best solution obtained so far. This estimate is calculated by adding the time taken by the crane to reach the current node and the lower bound for the time the crane still needs to rearrange the bay, and subtracting the maximum improvement in crane time achieved so far by the local search. This subtraction makes it possible to obtain solutions that are initially worse than the best known solution but that could be improved later by the local search.
3. The estimate of the number of moves required to obtain a feasible solution is less than the upper bound on the number of moves to solve the CPMPCT. The estimate is calculated by adding the number of moves involved in w and the number of blocking containers at the current node, and subtracting the maximum reduction in the number of moves resulting from the local search. If no feasible solution has yet been found, we add the depth of the node being evaluated to the upper bound, to give the heuristic more leeway to obtain a solution.

The Heuristic() function makes BG moves first (line 6), in the following order of preference:

1. The move with the smallest difference in group value between the container at the top of the destination stack and the container moved.

2. The move with the largest blocking group of containers in the origin stack.
3. The move requiring the shortest crane time.

If no more BG moves are possible, it starts making GG moves (line 7), according to the following order of preference:

1. The move with the largest difference in group value between the new top of the origin stack and the top of the destination stack before the move. Only moves where the difference is at least one are considered.
2. The move in which the priority of the new top of the origin stack is highest.
3. The move requiring the shortest crane time.

If neither BG moves nor GG moves are possible, the heuristic tries to empty the stack from which the least number of containers have to be removed to free a position in order to allocate the blocking container with maximum group value (line 8). During stack emptying, the destination of the relocated container is selected according to the following preferences:

1. The stacks in which the container will be upside down.
2. The stacks with the highest topmost container.
3. The stacks to which the movement involves the shortest crane time

If a stack cannot be emptied according to these criteria, the heuristic ends without obtaining a feasible solution. If the algorithm ends with a feasible solution, the local search algorithm $LS()$ described in Section 5.3.2 is applied to improve it (line 10), updating mt and mm values whenever necessary. If the improved solution is better than π, the best solution found, π, is updated as well as the upper bound on the number of moves to solve the CPMPCT (line 12).

5.3.2. An algorithm to improve feasible solutions

Algorithm 4 shows the pseudocode of the improvement phase used in the heuristic algorithm. First, it replaces dominated sequences in the solution with sequences that dominate them, considering Propositions 1 to 11 in lines 9 to 14. Direct dominance refers to Propositions 1 and 7, and to Propositions 5 and 6 with $n = 4$. These direct dominances apply to two consecutive movements, so they are quicker to check and are therefore checked first. Since the solution up to the $depth$-th move has been built taking into account the dominance rules, we check the rules for the moves added by the heuristic, that is, for moves from $depth + 1$ to $m(s)$. Next, the improvement phase tries to replace the last destination stack of each container by another stack in which the container is also well placed but which involves less crane time (lines 16–19). We only consider as candidate stacks those that are not the origin stacks of a subsequent movement. If the candidate stack is the destination stack of a subsequent movement, the group of the container moved must be larger than the group of the container moved in the next move involving that stack. In addition, it must be ensured that all containers entering the stack during subsequent movements fit into the stack height.

6. Computational results

In order to test the performance of the Beam Search based algorithm, BS, as well as the effect of each of the elements of which it is composed, an extensive computational analysis has been carried out, comparing the results of the BS algorithm with the results obtained by the branch and bound algorithm proposed by Parreño-Torres et al. (2020), CTA. We coded the algorithms in C++ and executed them on virtual machines with 4 virtual processors and 16 Gigabytes of RAM each. The virtual machines ran Windows 10 Enterprise 64 bits. Virtual machines were run in an OpenStack virtualization platform supported by several blade servers, each with two 18-core Intel Xeon Gold 5220 processors running at 2.2 GHz and 384 Gigabytes of RAM.

6.1. Test instances

We focus on five well-known datasets from the literature: on the one hand, the three datasets from van Brink & van der Zwaan (2014) (BZ dataset), Expósito-Izquierdo et al. (2012) (EMM dataset), and Zhang et al. (2015) (ZJY dataset), in which the branch and bound algorithm by Parreño-Torres et al. (2020) obtains a feasible or optimal solution in every instance, and on the other, the datasets from Bortfeldt & Forster (2012) (BF dataset) and Caserta & Voß (2009) (CV dataset), which contain the most difficult instances in the literature. The BF dataset is not considered in Parreño-Torres et al. (2020), and from the CV dataset only instances of up to 8 tiers are considered, that is, 760 instances out of the total, 840. The specific details of each dataset are as follows:

BZ dataset. This includes 960 instances with a number of tiers ranging from 4 to 6, with 3, 5, 6, and 9 stacks, with container fill rates of 50% and 70%, and with 2, 3, or 6 container groups.

EMM dataset. As the original instances from Expósito-Izquierdo et al. (2012) were lost, we consider those re-generated by Tierney et al. (2017), which use a similar distribution. This includes 450 instances with 4 tiers and 4, 7, or 10 stacks, filled to 50% or 75%. The number of container groups ranges from 2 to 8.

ZJY dataset. This dataset comprises 100 instances, with 4 tiers and 6, 7, 8, or 9 stacks, and with 5 tiers and 6 stacks. There may be several containers with the same group, and the stacks are usually filled up to $|T| - 1$ tiers. Containers of 10 different groups are considered.

CV dataset. This dataset has 840 instances divided into 21 categories of 40 instances each, containing 5 tiers and 3 to 8 stacks, 6 tiers and 5 to 7 stacks, 7 tiers and 4 to 9, 8 tiers and 6 or 10 stacks, and 12 tiers and 8 or 10 stacks. All the containers have different group values and stacks are filled to the same height with the two highest tiers empty.

BF dataset. This dataset contains 681 instances which are divided into 37 categories, 32 belonging to subset BF, with 20 instances each, and 5 belonging to LC, 4 with 10 instances and the remainder with 1 instance. The BF categories have 5 or 8 tiers and 16 or 20 stacks and the LC categories have 5 or 6 tiers and 10 or 12 stacks. There can be multiple containers with the same group value.

In total, 3031 instances were used to test the proposed algorithm. Moreover, to avoid overfitting these datasets, we created a training set consisting of 152 randomly selected instances in the BF and CV datasets, as these contain the most complex instances. We denote the training set as TS throughout this section.

6.2. Assessing the effect of each element on the proposed algorithm, BS.

In order to set the parameters of the proposed BS algorithm, as well as to assess the effect that each of the elements included has on its performance, we test different variants of the algorithm, shown in Table 4, over the training set. Taking the full BS algorithm as a reference, in BS^0 and BS^1 the dominance rules are removed, completely in BS^0 and partially, leaving only the direct dominance rules, in BS^1. BS-NH does not include the heuristic used in the global evaluation and in BS-NLS only the local search phase is removed. The last four variants only change the percentage of nodes, α%, which are given priority because they are the best in each sublist. In algorithm BS, $\alpha = 10$; in the variants, this value is changed

Table 4
Algorithm variants directly compared on the training set.

Algorithm	Dominance	α	HEURISTIC()	LS()
BS	All	10	✓	✓
BS^0	None	10	✓	✓
BS^1	Direct	10	✓	✓
BS-NH	All	10	✗	✓
BS-NLS	All	10	✓	✗
$BS_{0\%}$	All	0	✓	✓
$BS_{30\%}$	All	30	✓	✓
$BS_{50\%}$	All	50	✓	✓
$BS_{100\%}$	All	100	✓	✓

to 0, 30, 50, and 100. The algorithms are compared with BS in terms of their average relative percentage deviation (AVRPD). For each algorithm *ALG* and each instance *I*, the relative percentage deviation is calculated as $\frac{ALG(I)-BS(I)}{BS(I)} * 100$.

We first run a beam-search-based algorithm considering only the criteria common to all groups and without the global evaluation. This simple version of BS fails to find a solution in 11.1% percent of the instances tested and finds none at all in any of the instances of the training set with 12 tiers. Moreover, in those where it does find a solution, it remains at an AVRPD of 3.15% from the full BS algorithm. Next, we run the variants contained in Table 4. Table 5 shows their AVRPD from our final algorithm on the training set grouped by number of tiers and by number of stacks. First, we check whether applying the dominance rules described in Section 4 has any effect. The solutions provided by the algorithm without the dominance rules, BS^0, have an AVRPD of 0.77%, and those provided by the algorithm using only direct dominance, BS^1 (Propositions 1 and 2 together with Proposition 5 with $n = 2$, Propositions 10 and 11 with $n = 3$, and Proposition 6 (or 7) with $n = 4$) have an AVRPD of 0.22% from the BS algorithm. Thus, the use of dominance rules contributes to finding better solutions. If we remove the heuristic for obtaining feasible solutions described in Section 5.3.1 from the algorithm, BS-NH, the AVRPD increases to 4.65%, reaching an AVRPD of around 18% in the instances with 12 tiers. If only the improvement phase described in Section 5.3.2 is removed, the solutions also become worse, with an AVRPD of 0.96%. The last element we compare in the table is the inclusion of different selection criteria depending on the type of movement. If the same criteria are used on all candidate nodes when selecting the β most promising, taking $\alpha = 0$, the algorithm's performance worsens, with an AVRPD of 0.31%. On the other hand, if we increase the percentage of selected nodes that are the best in each subgroup, the algorithm performance also worsens. The last three columns in Table 5 show that $\alpha = 30$ and $\alpha = 50$ produce very similar results to those obtained with the reference value of $\alpha = 10$, but they are clearly worse if the value is increased to 100.

6.3. Comparing the results obtained by BS with the results obtained by the CTA algorithm

We compare here the results obtained by the full Beam Search algorithm, BS, considering time limits of 60, 300, and 3600 seconds, with those obtained by the branch and bound algorithm, CTA, run on the same machines with a fixed time limit of 3600 seconds. Table 6 shows the results on the 1510 instances of the BZ, EMM, and ZJY datasets grouped by number of tiers and number of stacks. The first three columns show the number of tiers and stacks and the number of instances in each class. The rest of the table is divided into two parts. Columns 4–11 show the results for the instances solved optimally by CTA and columns 12–19 those for the instances not optimally solved. Columns *CTA* and *BS* provide the average crane times and columns *AVRPD* the average percentage deviations (average of the crane times provided by BS minus those of CTA divided by the crane time provided by CTA).

Table 5
Average percentage relative deviation (AVRPD) from the BS algorithm proposed.

				Dominances		Heuristic		Groups			
	$\|\mathcal{T}\|$	$\|\mathcal{S}\|$	#I	BS^0	BS^1	BS-NH	BS-NLS	$BS_{0\%}$	$BS_{30\%}$	$BS_{50\%}$	$BS_{100\%}$
BF dataset											
	5	16	16	0.31	0.06	4.89	1.36	0.24	0.11	0.14	1.59
	5	20	16	1.22	0.38	7.35	2.32	0.48	0.45	0.43	1.69
	8	16	16	0.73	-0.12	4.90	1.23	-0.24	0.11	0.28	1.27
	8	20	16	0.24	0.31	8.79	1.35	0.51	-0.28	-0.13	1.17
Tot/Avg			64	0.63	0.16	6.48	1.57	0.25	0.10	0.18	1.43
CV dataset											
	5	3	4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	5	4	4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	5	5	4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	5	6	4	0.00	0.00	0.06	0.00	0.00	0.00	0.00	0.00
	5	7	4	0.14	0.08	0.69	0.16	0.00	0.00	0.00	-0.07
	5	8	4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	6	4	4	0.00	0.00	1.28	0.00	0.00	0.00	0.00	0.77
	6	5	4	0.30	0.00	0.63	0.00	0.00	0.00	0.00	0.52
	6	6	4	0.41	-0.13	1.14	0.00	-0.03	0.00	0.00	0.91
	6	7	4	0.81	0.05	1.23	-0.09	-0.49	-0.61	0.00	0.82
	6	12	4	0.06	-0.10	1.97	-0.28	-0.07	-0.09	-0.36	0.72
	7	4	4	1.25	0.00	5.08	0.26	0.00	0.00	0.00	3.19
	7	5	4	0.21	0.15	5.63	0.00	1.05	0.81	-0.36	1.24
	7	6	4	2.93	0.79	1.97	1.83	-0.23	0.17	0.35	2.12
	7	7	4	-0.06	0.88	2.81	1.23	0.02	0.02	0.04	1.35
	7	8	4	1.71	0.68	3.13	1.38	0.12	-0.15	-0.07	1.72
	7	9	4	2.39	1.30	5.52	2.06	0.23	1.13	0.66	2.56
	7	10	4	1.36	0.24	3.35	1.29	0.20	-0.37	1.06	1.66
	8	6	4	0.66	-0.85	1.17	-0.53	0.46	-1.11	-1.23	-0.35
	8	10	4	1.35	-0.55	1.84	2.02	-0.82	-0.97	-0.94	0.84
	12	6	4	1.40	-0.02	22.12	0.23	4.43	0.46	-0.80	0.28
	12	10	4	2.37	3.32	13.24	1.68	3.04	1.24	2.12	2.97
Tot/Avg			88	0.79	0.27	3.31	0.51	0.36	0.02	0.02	0.97
	Tot/Avg		152	0.72	0.22	4.65	0.96	0.31	0.05	0.09	1.16

Table 6
Comparing the BS algorithm with CTA algorithm on BZ, EMM, and ZJY datasets.

					60 seconds		300 seconds		3600 seconds				60 seconds		300 seconds		3600 seconds	
$\|\mathcal{T}\|$	$\|\mathcal{S}\|$	#I	#O	CTA	BS	AVRPD	BS	AVRPD	BS	AVRPD	#SF	CTA	BS	AVRPD	BS	AVRPD	BS	AVRPD
BZ dataset																		
4	3	120	120	374	374	0.00	374	0.00	374	0.00	0	-	-	-	-	-	-	-
4	5	120	120	445	445	0.00	445	0.00	445	0.00	0	-	-	-	-	-	-	-
4	7	120	119	547	547	0.00	547	0.00	547	0.00	1	1380	1380	0.00	1380	0.00	1380	0.00
4	9	120	98	600	600	0.00	600	0.00	600	0.00	22	1279	1280	0.06	1280	0.06	1279	-0.01
6	3	120	120	949	949	0.00	949	0.00	949	0.00	0	-	-	-	-	-	-	
6	5	120	105	1102	1102	0.00	1102	0.00	1102	0.00	15	2154	2158	0.15	2154	0.00	2154	0.00
6	7	120	56	996	996	0.00	996	0.00	996	0.00	64	1994	1999	0.17	1996	0.08	1994	0.02
6	9	120	21	918	918	0.00	918	0.00	918	0.00	99	2148	2140	-0.32	2136	-0.44	2134	-0.52
Tot/Avg		960	759	694	694	0.00	694	0.00	694	0.00	201	2000	1998	-0.09	1995	-0.19	1994	-0.25
EMM dataset																		
4	4	150	150	473	473	0.00	473	0.00	473	0.00	0	-	-	-	-	-	-	-
4	7	150	125	658	658	0.00	658	0.00	658	0.00	25	2146	2169	1.11	2163	0.82	2159	0.64
4	10	150	83	692	692	0.00	692	0.00	692	0.00	67	1879	1888	0.37	1884	0.18	1880	0.06
Tot/Avg		450	358	588	588	0.00	588	0.00	588	0.00	92	1951	1965	0.57	1960	0.36	1956	0.22
ZJY dataset																		
4	6	20	20	1053	1053	0.00	1053	0.00	1053	0.00	0	-	-	-	-	-	-	-
4	7	20	18	1072	1072	0.00	1072	0.00	1072	0.00	2	1577	1583	0.35	1578	0.08	1578	0.08
4	8	20	17	977	977	0.00	977	0.00	977	0.00	3	1493	1494	0.11	1493	0.00	1493	0.00
4	9	20	10	1079	1079	0.00	1079	0.00	1079	0.00	10	1453	1453	0.00	1453	0.00	1453	0.00
5	6	20	15	1448	1449	0.03	1448	0.00	1448	0.00	5	1984	1984	0.00	1984	0.00	1984	0.00
Tot/Avg		100	80	1118	1119	0.01	1118	0.00	1118	0.00	20	1604	1605	0.05	1604	0.01	1604	0.01
Tot/Avg		1510	1197	691	691	0.00	691	0.00	691	0.00	313	1960	1963	0.12	1960	-0.01	1958	-0.10

Table 7
Comparing the BS and CTA algorithm on CV dataset.

						All			60 seconds		300 seconds		3600 seconds			60s	300s	3600s
$\|\mathcal{T}\|$	$\|\mathcal{S}\|$	$\|\mathcal{C}\|$	#I	#O	CTA	AVRPD	#SF	CTA	BS	AVRPD	BS	AVRPD	BS	AVRPD	#NSF	BS	BS	BS
5	3	9	40	40	984	0.00	-	-	-	-	-	-	-	-	-	-	-	-
5	4	12	40	40	1043	0.00	-	-	-	-	-	-	-	-	-	-	-	-
5	5	15	40	40	1188	0.00	-	-	-	-	-	-	-	-	-	-	-	-
5	6	18	40	35	1270	0.00	5	1822	1822	0.00	1822	0.00	1822	0.00	-	-	-	-
5	7	21	40	25	1375	0.00	15	1792	1795	0.18	1794	0.11	1792	0.03	-	-	-	-
5	8	24	40	11	1327	0.00	29	1750	1751	0.10	1750	0.04	1750	0.03	-	-	-	-
6	4	16	40	35	1830	0.00	5	2369	2372	0.15	2372	0.15	2369	0.00	-	-	-	-
6	5	20	40	9	1790	0.00	31	2342	2345	0.10	2343	0.02	2342	0.00	-	-	-	-
6	6	24	40	3	1600	0.00	37	2471	2488	0.68	2483	0.44	2476	0.18	-	-	-	-
6	7	28	40	0	-	-	40	2766	2799	1.16	2788	0.79	2779	0.47	-	-	-	-
7	4	20	40	6	2221	0.00	34	3059	3102	1.36	3091	1.01	3082	0.72	-	-	-	-
7	5	25	40	1	1876	0.00	37	3244	3331	2.70	3301	1.80	3286	1.30	2	3533	3533	3533
7	6	30	40	0	-	-	37	3926	3977	1.47	3953	0.89	3919	-0.01	3	4377	4376	4344
7	7	35	40	0	-	-	36	4283	4346	1.59	4308	0.75	4281	0.13	4	4912	4879	4793
7	8	40	40	0	-	-	36	4973	4979	0.25	4948	-0.37	4910	-1.10	4	5391	5336	5330
7	9	45	40	0	-	-	32	5589	5565	-0.10	5544	-0.48	5495	-1.34	8	5688	5636	5620
7	10	50	40	0	-	-	30	5865	5894	0.48	5816	-0.78	5785	-1.32	10	6668	6566	6527
8	6	36	40	0	-	-	2	5252	5616	7.04	5495	4.91	5495	4.91	38	5673	5634	5561
8	10	60	40	0	-	-	3	9134	8560	-5.58	8497	-6.31	8221	-9.08	37	8708	8621	8496
12	8	80	40	0	-	-	-	-	-	-	-	-	-	-	40	17,098	16,678	16,414
12	10	100	40	0	-	-	-	-	-	-	-	-	-	-	40	24,781	24,357	24,030
Tot/Avg			840	245	1315	0.00	409	3590	3616	0.87	3594	0.37	3573	-0.09	186	12,831	12,614	12,442

The instances in these datasets are considered simple in the literature, since state-of-the-art branch and bound algorithms for the standard CPMP can solve all of them optimally in a few seconds, and CTA solves 79.2% of them optimally for the CPMPCT. Although the margin for improvement is small, it is observed that the BS algorithm is able to reach the optimal solution in all the instances solved to optimality by CTA considering a time limit of 300 seconds and in all but two by considering a limit of 60 seconds. In addition, in the instances not optimally solved by CTA, BS cuts the crane time relative to CTA by 0.01% with a time limit of 300 seconds and 0.10% with one of 3600 seconds, although the crane time is slightly worse if BS runs for only 60 seconds.

Let us now focus on the most challenging datasets. We start with CV; the results can be seen in Table 7. The first four columns show the characteristics of the instances: tiers, stacks, containers, and number of instances in each group. The rest of the table is divided into three parts. Columns 5 to 7 compare CTA and BS on the instances optimally solved by CTA. Their number is shown in column *#O*, the average optimal value in column *CTA* and the AVRPD of BS relative to CTA in column *AVRPD*. The CTA algorithm solves 245 out of the 840 instances to optimality. The algorithm's performance worsens as the number of tiers increases. For these 245 instances, BS reaches all the optimal solutions even with a time limit of 60 seconds. The second part of the table, columns 8–15, shows the 409 instances for which CTA obtains a solution but does not reach optimality. For these instances, the average percentage deviation is 0.87% running BS with a time limit of 60 seconds, 0.37% running it with 300 seconds, and -0.09% with 3600 seconds. The third part of the table, columns 16–19, corresponds to the 186 instances for which CTA does not find a solution. BS finds a solution

Table 8
Comparing the BS algorithm with CTA algorithm on BF dataset.

							60 seconds		300 seconds		3600 seconds			60s	300s	3600s
Set	$\|\mathcal{T}\|$	$\|\mathcal{S}\|$	$\|\mathcal{C}\|$	#I	#SF	CTA	BS	AVRPD	BS	AVRPD	BS	AVRPD	#NSF	BS	BS	BS
BF1	5	16	48	20	20	3422	3300	-3.53	3289	-3.88	3274	-4.31	0	-	-	-
BF2	5	16	48	20	20	4300	4163	-3.17	4134	-3.85	4114	-4.30	0	-	-	-
BF3	5	16	48	20	20	3436	3344	-2.67	3333	-2.99	3321	-3.32	0	-	-	-
BF4	5	16	48	20	20	4321	4155	-3.80	4140	-4.15	4120	-4.61	0	-	-	-
BF5	5	16	64	20	20	4907	4801	-2.18	4774	-2.72	4745	-3.32	0	-	-	-
BF6	5	16	64	20	20	5859	5858	-0.01	5817	-0.70	5787	-1.21	0	-	-	-
BF7	5	16	64	20	20	5049	5016	-0.63	4989	-1.16	4954	-1.87	0	-	-	-
BF8	5	16	64	20	20	6059	5998	-0.98	5945	-1.86	5905	-2.51	0	-	-	-
BF9	8	16	77	20	19	8193	7952	-2.90	7907	-3.45	7872	-3.89	1	8863	8631	8631
BF10	8	16	77	20	20	9329	9044	-3.08	8987	-3.67	8949	-4.07	0	-	-	-
BF11	8	16	77	20	20	8364	8117	-2.86	8070	-3.40	8025	-3.94	0	-	-	-
BF12	8	16	77	20	20	9425	9111	-3.33	9064	-3.83	9006	-4.44	0	-	-	-
BF13	8	16	103	20	4	11,460	11,048	-3.59	11,000	-4.00	10,884	-5.02	16	11,214	11,071	10,977
BF14	8	16	103	20	0	-	-	-	-	-	-	-	20	14,155	13,936	13,762
BF15	8	16	103	20	1	11,125	11,227	0.92	11,227	0.92	10,856	-2.42	19	11,469	11,302	11,131
BF16	8	16	103	20	0	-	-	-	-	-	-	-	20	14,347	14,042	13,803
BF17	5	20	60	20	20	4410	4245	-3.71	4223	-4.22	4198	-4.78	0	-	-	-
BF18	5	20	60	20	20	5443	5259	-3.34	5233	-3.82	5197	-4.49	0	-	-	-
BF19	5	20	60	20	20	4402	4265	-3.10	4239	-3.69	4197	-4.65	0	-	-	-
BF20	5	20	60	20	20	5445	5286	-2.90	5262	-3.34	5233	-3.88	0	-	-	-
BF21	5	20	80	20	18	6283	6164	-1.87	6123	-2.52	6080	-3.21	2	6077	6067	6016
BF22	5	20	80	20	20	7590	7414	-2.30	7342	-3.24	7276	-4.11	0	-	-	-
BF23	5	20	80	20	20	6222	6104	-1.84	6040	-2.88	5990	-3.68	0	-	-	-
BF24	5	20	80	20	20	7621	7520	-1.31	7451	-2.23	7370	-3.29	0	-	-	-
BF25	8	20	96	20	20	10,247	9877	-3.60	9829	-4.07	9759	-4.75	0	-	-	-
BF26	8	20	96	20	20	11,836	11,393	-3.75	11,314	-4.42	11,253	-4.93	0	-	-	-
BF27	8	20	96	20	20	10,263	9957	-2.94	9917	-3.34	9833	-4.15	0	-	-	-
BF28	8	20	96	20	20	12,000	11,566	-3.62	11,477	-4.35	11,403	-4.96	0	-	-	-
BF29	8	20	128	20	4	14,847	13,950	-6.04	13,875	-6.56	13,587	-8.48	16	14,326	14,075	13,856
BF30	8	20	128	20	1	16,893	16,594	-1.77	16,303	-3.50	15,928	-5.71	19	17,682	17,250	16,958
BF31	8	20	128	20	2	14,485	14,097	-2.55	13,901	-3.94	13,634	-5.78	18	14,282	14,047	13,919
BF32	8	20	128	20	0	-	-	-	-	-	-	-	20	17,620	17,254	17,036
LC1	5	10	35	1	1	1702	1702	0.00	1702	0.00	1702	0.00	0	-	-	-
LC2a	6	12	50	10	10	2770	2737	-1.10	2731	-1.34	2722	-1.64	0	-	-	-
LC2b	6	12	50	10	10	4994	4933	-1.25	4900	-1.92	4876	-2.39	0	-	-	-
LC3a	6	12	54	10	10	2844	2811	-1.18	2802	-1.46	2801	-1.51	0	-	-	-
LC3b	6	12	54	10	9	5395	5388	-0.12	5362	-0.58	5345	-0.90	1	5659	5518	5518
Tot/Avg				681	529	6779	6595	-2.54	6554	-3.12	6508	-3.74	152	14,268	14,004	13,814

in all of these instances. With a time limit of 60 seconds, the average crane time is 12,831 seconds, and it is reduced by 1.7% with 300 and by 3.0% with 3600 seconds.

Table 8 compares BS and CTA on the 681 BF test instances. In this case, since CTA does not solve any of them optimally, the table is divided into two parts. Columns 6–13 show the results of the instances for which a solution is found by CTA. The number of such instances appears in column *#SF* and then the average value obtained by CTA in 3600 seconds and by BS running for 60, 300, and 3600 seconds. BS reduces the average crane time by 2.54% with a time limit of 60 seconds, by 3.12% with 300 seconds, and by 3.74% with 3600 seconds. Columns 13–14 show the results for the instances in which CTA does not find a solution. The number of these instances appears in column *#NSF*, 152 in total. BS finds a solution for all of them. The average crane time for these instances is 14,268 seconds, considering a time limit of 60 seconds. This time is reduced by 1.85% when running for 600 seconds, and by 3.18% if it runs for 3600 seconds.

As in previous datasets, the algorithm keeps improving the solutions if the running time increases. In the iterative process, the beam width is enlarged, more nodes are considered at each level and this makes it possible to obtain solutions that were discarded when the beam width was smaller.

7. Conclusions

Premarshalling problems are gaining greater importance due to the increasing size of vessels and the pressure to reduce their stay in port by minimizing the time needed for loading and unloading operations. We have addressed the premarshalling problem with the objective of minimizing the time the yard crane takes to rearrange a bay and have developed a beam search algorithm.

Beam search algorithms, in particular, and tree search structures, in general, are appropriate when solutions consist of a sequence of moves that produces a feasible solution step by step. The main challenge for a beam search algorithm is to avoid discarding moves that apparently do not produce an immediate improvement, for instance moves emptying a stack, but that could lead to good solutions later in the tree. We have developed two mechanisms. On the one hand, we use various criteria to evaluate the four types of moves, BB, BG, GB, and GG. On the other hand, we designed a simple but powerful heuristic for the global evaluation. In addition, we have developed a new set of dominance rules which help to identify and eliminate nodes that cannot lead to better solutions.

The extensive computational study on several literature benchmarks shows that the beam search algorithm is able to obtain all the optimal solutions identified by the existing branch and bound algorithm, improves on the known suboptimal solutions, and obtains good solutions for the largest instances in which the branch and bound algorithm could not reach a feasible solution.

The ideas developed for this variant of the premarshalling problem can be applied to other variants such as the time-limited case studied by Zweers et al. (2020), and also to related problems, such as the block relocation problem.

Acknowledgments

This work has been partially supported by the Spanish Ministry of Science, Innovation, and Universities, project RTI2018-094940-B-100, partially financed with FEDER funds, by the Junta de Comunidades de Castilla-La Mancha, project SBPLY/17/180501/000282, and by the Junta de Comunidades de Castilla y León, project BU056P20, partially financed with FEDER funds.

Appendix A. Proofs of the dominance rules provided in Section 4

Proof of Proposition 2 Since d_1 and d_{n-1} are invariant to the sequence $(o_2, t_2, d_2, l_2) \dots (o_{n-2}, t_{n-2}, d_{n-2}, l_{n-2})$ of s_1, their layout before and after it is the same, and none of the containers placed in d_1 or in d_{n-1} before (or after) the sequence are handled during the sequence. Moreover, the containers that are temporarily placed on stack d_{n-1} during the sequence from move 2 to $n-2$ still fit if d_{n-1} has one more container. In that case, the same final layout is obtained by the feasible sequence s_2 in which the container c_1, which is moved first from o_1 to d_1 and then from d_1 to d_{n-1} in sequence s_1, is moved directly from stack o_1 to d_{n-1} prior to the sequence of moves $(o_2, t'_2, d_2, l'_2) \dots (o_{n-2}, t'_{n-2}, d_{n-2}, l'_{n-2})$ in sequence s_2. The primes indicate that the tiers from which a container is taken or in which it is left in the sequence from move 2 to $n-2$ of s_1 are modified so that moves involving stack d_1 are made at a lower tier and moves involving stack d_{n-1} are made at a higher tier. Since s_1 and s_2 lead to the same layout, s_1 is dominated by s_2 if the time taken by the crane to carry out the sequence is longer, which is satisfied if condition 3 is satisfied.

Proof of Proposition 3 Since o_2 and d_2 are invariant to the sequence $(o_3, t_3, d_3, l_3) \dots (o_{n-1}, t_{n-1}, d_{n-1}, l_{n-1})$ of s_1 and the containers that are temporarily placed on stack o_2 during that sequence (from move 3 to $n-1$) still fit if o_2 has one more container, the same final layout is obtained by the feasible sequence s_2 in which container c_2, which is moved first from o_2 to d_2 and then from d_2 to d_n in sequence s_1, is directly moved from stack o_2 to d_n after the sequence of moves $(o_3, t'_3, d_3, l'_3) \dots (o_{n-1}, t'_{n-1}, d_{n-1}, l'_{n-1})$. The primes indicate that the tiers from which a container is taken or in which it is left in the sequence from move 3 to $n-1$ of s_1 are modified, so that moves involving stack d_2 are made at a lower tier and moves involving stack o_2 are made at a higher tier. Since s_1 and s_2 lead to the same layout, s_1 is dominated by s_2 if the time spent by the crane in carrying out the sequence is longer, which is satisfied if condition 3 is satisfied.

Proof of Proposition 4 Since stacks d_1 and d are invariant to the sequence $(o_2, t_2, d_2, l_2) \dots (o_{n-1}, t_{n-1}, d_{n-1}, l_{n-1})$ of s_1 and the containers that are temporarily placed on stack d during that sequence (from move 2 to $n-1$) still fit if d has one more container, the same final layout is obtained by the feasible sequence s_2. Whereas container c_1 is moved first from stack o_1 to d_1 and then from d_1 to d_n in sequence s_1, container c_1 is temporarily moved to stack d instead of to stack d_1 in sequence s_2. The primes indicate that the tiers from which a container is taken or in which it is left in the sequence from move 2 to $n-1$ of s_1 are modified so that now moves involving stack d_1 are made at a lower tier and moves involving stack d are made at a higher tier. Since s_1 and s_2 lead to the same layout, s_1 is dominated by s_2 if the time spent by the crane in carrying out the sequence is longer, which is satisfied if condition 3 is satisfied.

Proof of Proposition 5 Since o_{n-1} and d_{n-1} are invariant to the sequence $(o_2, t_2, d_2, l_2) \dots (o_{n-2}, t_{n-2}, d_{n-2}, l_{n-2})$ in s_1 and the containers that are temporarily placed on stack d_{n-1} during that sequence still fit if d_{n-1} has one more container, the same final layout is obtained by moving container c_{n-1} from o_{n-1} to d_{n-1} in second position. The primes indicate that the tiers from which a container is taken or in which it is left in the sequence from move 2 to $n-2$ of s_1 are modified so that now moves involving stack o_{n-1} are made at a lower tier and moves involving stack d_{n-1} are made at a higher tier. Since s_1 and s_2 lead to the same layout, s_1 is dominated by s_2 if the time spent by the crane in carrying out the sequence is longer, which is satisfied if condition 3 is satisfied.

Proof of Proposition 6 Since o_2 and d_2 are invariant to the sequence $(o_2, t_2, d_2, l_2) \dots (o_{n-1}, t_{n-1}, d_{n-1}, l_{n-1})$ in s_1 and the containers that are temporarily placed on stack o_2 during that sequence still fit if o_2 has one more container, the same final layout is obtained by moving container c_2 from stack o_2 to stack d_2 in penultimate position, that is, just before the move (o_n, t_n, d_n, l_n). The primes indicate that the tiers from which a container is taken or in which it is left in the sequence from move 2 to $n-1$ of s_1 are modified so that now moves involving stack d_2 are made at a lower tier and moves involving stack o_2 are made at a higher tier. Since s_1 and s_2 lead to the same layout, s_1 is dominated by s_2 if the time spent by the crane in carrying out the sequence is longer, which is satisfied if condition 3 is satisfied.

Proof of Proposition 8 A container c_2 of group value $g = group(c_2)$ is first moved from stack o_2 to d_2 and, after a sequence of moves, another container c_{n-1} of the same group value $group(c_{n-1}) = group(c_2) = g$ is moved from o_{n-1} to stack o_2 in s_1. Since o_2 and o_{n-1} are invariant to the sequence $(o_3, t_3, d_3, l_3) \dots (o_{n-2}, t_{n-2}, d_{n-2}, l_{n-2})$ in s_1 and the containers that are temporarily placed on stack o_2 during that sequence still fit if o_2 has one more container, the same final layout is obtained by moving container c_{n-1} in second position directly from stack o_{n-1} to stack d_2. The primes indicate that the tiers from which a container is taken or in which it is left in the sequence from move 3 to $n-2$ of s_1 are modified so that now moves involving stack o_{n-1} are made at a lower tier and moves involving stack o_2 are made at a higher tier. Since s_1 and s_2 lead to the same layout, s_1 is dominated by s_2 if the time spent by the crane in carrying out the sequence is longer, which is satisfied if condition 3 is met.

Proof of Proposition 9 A container c_2 of group value $g = group(c_2)$ is first moved from stack o_2 to d_2 and, after a sequence of moves, another container c_{n-1} of the same group value $group(c_{n-1}) = group(c_2) = g$ is moved from o_{n-1} to stack o_2 in s_1. Since o_2 and d_2 are invariant to the sequence $(o_3, t_3, d_3, l_3) \dots (o_{n-2}, t_{n-2}, d_{n-2}, l_{n-2})$ in s_1 and the containers that are temporarily placed on stack o_2 during that sequence still fit if o_2 has one more container, the same final layout is obtained by moving container c_2 of group value $group(c_2)$ in penultimate position, that is, just before moving (o_n, t_n, d_n, l_n), directly from stack o_{n-1} to stack d_2. The primes indicate that the tiers from which a container is taken or in which it is left in the sequence from move 3 to $n-2$ of s_1 are modified so that now moves involving stack d_2 are made at a lower tier and moves involving stack o_2 are made at a higher tier. Since s_1 and s_2 lead to the same layout, s_1 is dominated by s_2 if the time taken by the crane to carry out the sequence is longer, which is satisfied if condition 3 is met.

Proof of Proposition 10 Two containers of the same group value are moved in (o_2, t_2, d_2, l_2) and in (o_n, t_n, d_n, l_n). Since o_2 and o_n are invariant to the sequence $(o_3, t_3, d_3, l_3) \dots (o_{n-1}, t_{n-1}, d_{n-1}, l_{n-1})$ in s_1 and the containers that are temporarily placed on stack o_2 during that sequence still fit if o_2 has one more container, the same final layout is obtained by moving container c_2 in penultimate position, that is, just before the move (o_n, t_n, d_n, l_n), to stack d_n, and moving container c_n in second position to stack d_2. The primes indicate that the tiers from which a container is taken or in which it is left in the sequence from move 3 to $n-1$ of s_1 are modified so that now moves involving stack o_n are made at a lower tier and moves involving stack o_2 are made at a higher tier. Since s_1 and s_2 lead to the same layout, s_1 is dominated by s_2 if the time

take by the crane to perform the sequence is longer, which is satisfied if condition 3 is met.

Proof of Proposition 11 Two containers of the same group value are moved in (o_1, t_1, d_1, l_1) and in $(o_{n-1}, t_{n-1}, d_{n-1}, l_{n-1})$. Since d_2 and d_n are invariant to the sequence $(o_2, t_2, d_2, l_2) \dots (o_{n-2}, t_{n-2}, d_{n-1}, l_{n-1})$ in s_1 and the containers that are temporarily placed on stack d_{n-1} during that sequence still fit if d_{n-1} has one more container, the same final layout is obtained by moving container c_1 to stack d_{n-1} and container c_{n-1} to stack d_1. The primes indicate that the tiers from which a container is taken or in which it is left in the sequence from move 2 to $n-2$ of s_1 are modified so that now moves involving stack d_1 are made at a lower tier and moves involving stack d_{n-1} are made at a higher tier. Since s_1 and s_2 lead to the same layout, s_1 is dominated by s_2 if the time spent by the crane in carrying out the sequence is longer, which is satisfied if condition 3 is met.

References

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

Bortfeldt, A. (2004). A heuristic for the container pre-marshalling problem. In *Proceedings of the 3rd international conference on computer and IT applications in the martime industry (compit'04)* (pp. 419–429).

Bortfeldt, A., & Forster, F. (2012). A tree search procedure for the container premarshalling problem. *European Journal of Operational Research, 217*(3), 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 A. Schulz, & D. Wagner (Eds.), *Proceedings of the Algorithms–ESA 2014*. In *Lecture Notes in Computer Science: vol. 8737* (pp. 798–809). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-44777-2_66.

Caserta, M., Schwarze, S., & Voß, S. (2020). Container rehandling at maritime container terminals: A literature update. In J. Böse (Ed.), *Handbook of terminal planning*. In *Operations Research/Computer Science Interfaces Series: vol. 64* (pp. 343–382). Springer Nature. https://doi.org/10.1007/978-3-030-39990-0_16.

Caserta, M., & Voß, S. (2009). A corridor method-based algorithm for the premarshalling problem. In M. Giacobini, A. Brabazon, S. Cagnoni, G. A. Di Caro, A. Ekárt, A. I. Esparcia-Alcázar, … P. Machado (Eds.), *Applications of evolutionary computing* (pp. 788–797). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-01129-0_89.

Expósito-Izquierdo, C., Melián-Batista, B., & Moreno-Vega, M. (2012). Premarshalling problem: Heuristic solution method and instances generator. *Expert Systems with Applications, 39*(9), 8337–8349. https://doi.org/10.1016/j.eswa.2012.01.187.

Ge, P., Meng, Y., Liu, J., Tang, L., & Zhao, R. (2020). Logistics optimisation of slab pre-marshalling problem in steel industry. *International Journal of Production Research, 58*, 4050–4070. https://doi.org/10.1080/00207543.2019.1641238.

Hottung, A., Tanaka, S., & Tierney, K. (2020). Deep learning assisted heuristic tree search for the container pre-marshalling problem. *Computers & Operations Research, 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. *Computers & Operations Research, 75*, 83–102. https://doi.org/10.1016/j.cor.2016.05.011.

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. https://doi.org/10.1007/s10100-015-0410-y.

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. 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. *European Journal of Operational Research, 196*(2), 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. *Computers & Operations Research, 34*(11), 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. *Computers & Operations Research, 37*(6), 1139–1147. https://doi.org/10.1016/j.cor.2009.10.005.

Libralesso, L., & Fontan, F. (2021). An anytime tree search algorithm for the 2018 roadef/euro challenge glass cutting problem. *European Journal of Operational Research, 291*(3), 883–893.

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

Maniezzo, V., Boschetti, M., & Gutjahr, W. (2021). Stochastic premarshalling of block stacking warehouses. *Omega, 102*, 102336. https://doi.org/10.1016/j.omega.2020.102336.

Parreño, F., Alonso, M., & Alvarez-Valdes, R. (2020). Solving a large cutting problem in the glass manufacturing industry. *European Journal of Operational Research, 287*, 378–388. https://doi.org/10.1016/j.ejor.2020.05.016.

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. https://doi.org/10.1016/j.ejor.2018.09.048.

Parreño-Torres, C., Alvarez-Valdes, R., Ruiz, R., & Tierney, K. (2020). Minimizing crane times in pre-marshalling problems. *Transportation Research Part E: Logistics and Transportation Review, 137*, 101917.

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.

Reddy, D. R., et al. (1977). *Speech understanding systems: A summary of results of the five-year research effort* p. 138. Department of Computer Science. Camegie-Mell University, Pittsburgh, PA.

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. 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. *Journal of Heuristics, 25*(2), 141–173. https://doi.org/10.1007/s10732-018-9390-0.

Statista (2021). Container shipping - statistics & facts. Accessed = 2021-05-14https://www.statista.com/topics/1367/container-shipping/#dossierSummary.

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. 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. *European Journal of Operational Research, 278*(1), 211–225. https://doi.org/10.1016/j.ejor.2019.04.005.

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. https://doi.org/10.1007/s10696-016-9246-6.

UNCTAD (2020). Review of maritime transport 2020. In *Proceedings of the United Nations Conference on Trade and Development*. https://unctad.org/webflyer/review-maritime-transport-2020

Wang, N., Jin, B., & Lim, A. (2015). Target-guided algorithms for the container premarshalling 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. *European Journal of Operational Research, 256*(1), 90–101. https://doi.org/10.1016/j.ejor.2016.05.061.

Yue, L., Fan, H., & Ma, M. (2021). Optimizing configuration and scheduling of double 40 ft dual-trolley quay cranes and AGVs for improving container terminal services. *Journal of Cleaner Production, 292*, 126019. https://doi.org/10.1016/j.jclepro.2021.126019.

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.

Zweers, B., Bhulai, S., & van der Mei, R. (2020). Pre-processing a container yard under limited available time. *Computers and Operations Research, 123*, 105045. https://doi.org/10.1016/j.cor.2020.105045.