Computers & Operations Research 133 (2021) 105383

Contents lists available at ScienceDirect

Computers and Operations Research

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

Solving the generalized multi-port container stowage planning problem by a matheuristic algorithm

Consuelo Parreño-Torres [a,*], Hatice Çalık [b], Ramon Alvarez-Valdes [c], Rubén Ruiz [d]

[a] Department of Applied Economics, Faculty of Economics and Business, University of Burgos, Plaza Infanta Doña Elena s/n, 09001, Burgos, Spain
[b] KU Leuven, Department of Computer Science, CODeS, Belgium
[c] Department of Statistics and Operations Research, Valencia University, Doctor Moliner 50, Burjassot, 46100, Valencia, Spain
[d] Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edificio 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, 46021, València, Spain

ARTICLE INFO

Keywords:
Container ship
Multiport stowage
Mathematical models
Matheuristics
Planning

ABSTRACT

We focus on a simplified container stowage planning problem where containers of different size and weight must be loaded and unloaded at multiple ports while maintaining the stability of the ship. We initially investigate how the difficulty in solving the problem changes with and without the consideration of container sizes and weight constraints. For this purpose, we provide integer programming formulations for the general problem as well as some special cases with identical container size and/or identical weights and evaluate their performance in randomly generated small- and medium-scale instances. We develop a matheuristic procedure, namely, an insert-and-fix heuristic, exploiting the special structure of the proposed formulations. The Insert-and-Fix method, in combination with a constructive algorithm that gives the solver an initial solution in each iteration, provides solutions with a low number of rehandles for instances with up to 5000 TEUs.

1. Introduction

Maritime transportation is one of the most energy-efficient modes of transportation. Seaborne trade constitutes a significant percentage of world trade. According to the 2019 review of the United Nations Conference on Trade and Development, world maritime trade volume reached 11 billion tonnes in 2018 and 793 million Twenty-foot Equivalent Units (TEUs) were handled in container ports worldwide (UNCTAD, 2019). The capacity of container vessels reached 20000 TEUs (Parreño-Torres et al., 2019), with some vessels exceeding that number launched in 2019. Considering such large volumes, maximizing efficiency in container handling is crucial to the global economy, as well as to the environment, since reducing unnecessary port operations results in a greatly reduced carbon footprint.

In order to minimize the number of loading/unloading operations at ports, it is essential to have an efficient stowage plan indicating the exact position of each container in the ship. Consider a ship that has to visit a set of ports in a given order to load and discharge containers. It starts its route by loading an initial set of containers at the first port, performs requested loading and unloading operations at the intermediary ports, and finally unloads all the remaining containers at the final port. The following data are given: (i) the number of containers to be loaded at each port, (ii) the port of load and port of discharge for each container, and thus (iii) the number of containers to be unloaded at each port.

Building a good stowage plan for a ship along the ports in its route is an NP-hard problem, with many decision variables that make it difficult to solve in practice. The main objectives are maximizing vessel utilization and minimizing operational costs. This second objective involves minimizing the number of rehandles and also minimizing the makespan of the cranes loading and unloading the containers. There are three main types of constraints. First, constraints related to container types. The most common containers are 20' and 40' containers, but there are also 45' containers, high cube containers, and other out-of-gauge (OOG) containers, whose positions are limited to certain positions in the ship. Reefer (refrigerated) containers must be placed in positions with power plugs. IMO containers (with dangerous goods of different types) must obey strict separation rules. Second, constraints related to the ship structure. The cargo space of a ship is divided into bays and each bay has an on-deck and an under-deck part, separated by a hatch-cover. Each part of a bay consists of a row of stacks, and each tier of a stack is a cell consisting of two slots in which one 40' or two 20' containers can be placed (see Fig. 1). Nevertheless, due to

* Correspondence to: Department of Applied Economics, Faculty of Economics and Business, University of Burgos, Plaza Infanta Doña Elena s/n, 09001, Burgos, Spain.
E-mail address: cparreno@ubu.es (C. Parreño-Torres).

https://doi.org/10.1016/j.cor.2021.105383
Received 7 April 2020; Received in revised form 13 May 2021; Accepted 14 May 2021
Available online 18 May 2021

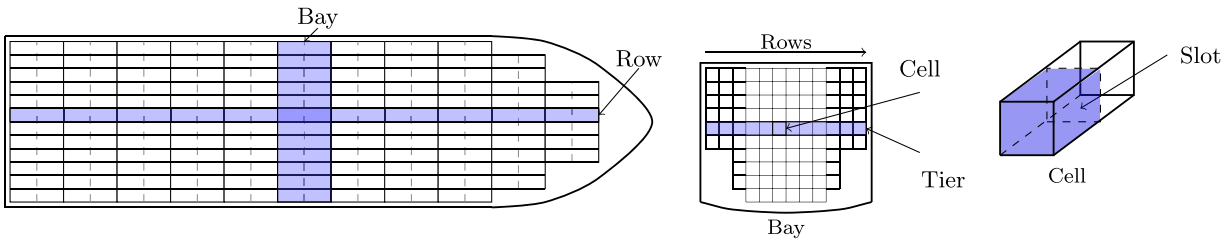

Fig. 1. Container ship scheme.

the structure of the ship, there may be some odd-slots in which only a 20' container can be placed. The weight and the height of each stack are limited. If a container has to be unloaded at a port, all containers on top of it in the same stack going to later ports must be unloaded and loaded again, causing *rehandles*. If the container is under deck, all containers on deck in the same bay must be rehandled. Below deck, the containers are secured by cell guides. On deck, they are secured by lashing rods, of limited strength, or lashing bridges. Containers on deck cannot block the line of sight from the pilot house to the sea in front of the ship. Third, constraints related to the sailing conditions. For a ship to be allowed to leave a port it must satisfy stability and stress forces constraints. There are also limits for the maximum and minimum draft and limits for the trim (inclination of the ship). A complete description of the technical requirements can be found in Jensen et al. (2018).

Optimizing the stowage plan taking into account all practical constraints is very hard. In the pioneering work of Botter and Brinati (1992), they considered all the constraints related to stability and other forces acting on a vessel and developed an integer linear model. They defined a binary variable for each container, position, and port and for every loading or unloading movement so as to ensure that the constraints are satisfied after each move, but this resulted in a huge number of variables, in the order of 10^9 even for small instances of 1000 TEUs and 6 ports and the model was not tested. Subsequently, two main strategies have been developed. One research line addresses a simplified problem, keeping its basic structure and progressively adding the most important constraints. The basic problem, where all the containers are of identical size and no additional constraints are present (stability, maximum weight, etc.), is referred to as the *Container Stowage Planning Problem* (CSPP) and it has been proven to be NP-Hard (Avriel et al., 2000). For the CSPP, several integer programming (IP) formulations and exact algorithms have been proposed, as well as heuristic approaches that can efficiently solve very large-scale instances. On the other hand, another line of research follows a hierarchical decomposition of the problem where container groups are first assigned to specific sections of the ship and then the exact positioning of each container is decided for each section. This paper aims to fill the gap between these two strategies by providing methods to solve the CSPP with container weight- and size-related constraints for instances with thousands of containers.

Our contribution is three-fold: (1) We introduce three generalizations of the CSPP, which come with several additional complexities: (i) the CSPP with 20' and 40' containers, container weights, and stability constraints, (ii) the CSPP with identical container sizes but different weights and stability constraints, and (iii) the CSPP with 20' and 40' containers where stacking constraints are present. (2) We provide an IP formulation for the CSPP, which outperforms the IP formulations proposed in the literature, and for the newly introduced generalizations. (3) Using the decomposable structure of these mathematical models, we develop a matheuristic approach, namely Insert-and-Fix (Wolsey, 1998), to solve the more general problem.

The remainder of this paper is organized as follows: Section 2 reviews the related papers from the literature. Section 3 describes the problems we study and defines the notation we use throughout the paper. Section 4 presents the mathematical models that we introduce for the aforementioned variants. Section 5 provides the details of the matheuristic algorithm that we develop. In this section, we also introduce a constructive procedure integrated into the insert-and-fix algorithm, which guarantees obtaining a feasible solution even when the solvers cannot provide one within the time limit. Section 6 details the computational experiments we conduct on our mathematical models and matheuristic algorithms. Finally, we present our conclusions in Section 7 and also provide some future research directions.

2. Related work

This section reviews relevant papers that consider container stowage planning where multiple ports are visited. So as not to distract the reader with a lengthy literature discussion, we exclude studies that focus only on single-port problems (see Larsen and Pacino, 2020 for an up-to-date review).

Although proposals to solve the problem can be traced back to Kessel (1977), the first mathematical formulation of the CSPP was provided by Avriel and Penn (1993) and revisited by Avriel et al. (1998). In addition to this formulation, Avriel and Penn (1993) introduce a *Whole Columns Heuristic Procedure* to solve the CSPP. Given a transportation matrix representing the demand between each pair of ports, the authors decompose it into two sub-matrices. The containers for the first sub-matrix are allocated through a simple procedure, whereas binary linear programming is used for the second sub-matrix. Avriel et al. (1998) extend this idea by using a dynamic slot-assignment scheme leading to a more efficient heuristic, which is referred to as the *Suspensory Heuristic Procedure*. The first metaheuristic approach by Dubrovsky et al. (2002) provides a genetic algorithm with an efficient solution-encoding scheme. This algorithm shows a similar performance to that of Avriel et al. (1998) on the CSPP instances tested and also introduces a ship stability constraint through a penalty function.

Ding and Chou (2015) point out that it is always possible to assume that the ship is fully loaded when traveling between ports. The transportation matrices satisfying this property are referred to as *full* matrices and Ding and Chou (2015) present a method for generating such matrices. They further provide a new IP formulation and a heuristic method to solve the CSPP for full transportation matrices. This heuristic generates a stowage plan for each port without considering the transportation information for subsequent ports (hence no look-ahead strategy). The computational study by Ding and Chou (2015) on large-scale problem instances indicates that their algorithm performs better than the Suspensory Heuristic of Avriel et al. (1998) when the number of ports visited is large.

A mixed integer programming (MIP) formulation of the CSPP with an exponential number of variables is proposed by Roberti and Pacino (2018). The authors develop a column generation-based lower bounding procedure and combine it with a compact MIP model to solve instances with up to 10 ports and 5000 containers to optimality. Problem instances of this size remain unsolvable by the compact IP formulations available in the literature.

The state-of-the-art compact IP formulation and heuristic method for solving the CSPP have recently been provided by Parreño-Torres et al. (2019). This new IP formulation has fewer variables than the

formulations by Avriel et al. (1998) and Ding and Chou (2015). The authors propose several valid inequalities to strengthen its linear programming (LP) relaxation. The experimental study comparing the three formulations for the CSPP indicates that the new IP model outperforms the previous models by Avriel and Penn (1993) and Ding and Chou (2015) in terms of both solution quality and computing time. Moreover, Parreño-Torres et al. (2019) develop a Greedy Randomized Adaptive Search Procedure (GRASP) and compare it with the heuristic algorithms by Avriel et al. (1998) and Ding and Chou (2015) on medium- and large-scale instances that they generate. The GRASP provides very good solutions, of higher quality than those obtained by the previous algorithms, in less than five minutes for every instance tested.

Compared to other combinatorial optimization problems, the literature on CSPP is relatively scarce, and a standard definition of the problem and benchmarks have not yet really been established. For the basic CSPP, some IP models, exact, and heuristic methods can be found. However, the literature lacks methods to solve the CSPP with different container sizes and weights as well as stability constraints. Only the aforementioned paper by Botter and Brinati (1992) addresses the problem with all of its characteristics.

A common practice is to adopt a decomposition strategy in two phases, solving first a *Master Bay Planning Problem* (MBPP) or a multi-port MBPP (MP-MBPP) variant to allocate containers to different sections of the ship and then solving a *Slot Planning Problem* (SPP) to determine the exact position (slot) of the containers in each section. A decomposition of this kind makes it possible to solve the SPP independently for each section including constraints arising in practical problems. However, it does not guarantee an optimal solution for the complete problem. Among the studies utilizing this decomposition structure for multi-port stowage planning problems, Wilson and Roach (1999) derive a method based on Tabu Search (TS). Kang and Kim (2002) iteratively solve the MP-MBPP via a greedy heuristic and the SPP via a tree search method, Pacino et al. (2011) combine an IP formulation for the MP-MBPP and a constraint programming approach for the SPP, and Ambrosino et al. (2015a, 2017) provide an MIP formulation for the MP-MBPP and a heuristic for the SPP. Ambrosino et al. (2015b) developed a Relax-and-Fix matheuristic approach, which solves at each iteration relaxations of the integer model progressively fixing subsets of binary variables, obtaining good results for very large instances of the MP-MBPP.

Other relevant papers include a variant of the CSPP where the reshuffling cost is not identical for all ports (Zhang et al., 2018) and an MP-MBPP variant with weight uncertainty (Li et al., 2018). To the best of our knowledge, the combined CSPP with different container sizes, weights, and stability constraints has not been studied in the literature.

3. Problem description

Container stowage plans use a bay-row-tier coordinate system to identify the position of a container aboard the ship. The bays represent the transverse sections of the ship and are numbered from bow (or forward) to stern (or aft). The rows run the length of the ship, with the starboard rows having odd numbers and the port rows having even numbers. The last coordinate is the tier or layer of the containers. Each position defined by these three coordinates is called a *cell* and it can contain two 20' or one 40' container. We can use a fourth coordinate to identify each *slot* with capacity for one 20' container, as shown in Fig. 1.

We consider a ship that visits several ports where container loading and unloading operations are carried out. The boat is empty before the route starts and also when the route is completed. The capacity and structure of the ship are known and we are also provided with the details of the containers to be loaded at each port, with their weight and size and their port of discharge. We thus have to decide the stowage plan of the ship along the whole route in order to minimize the total number of rehandles. Rehandles are required at a port for unloading containers placed below other containers that do not have to be unloaded at this port and for loading containers in slots below other existing containers. A feasible solution must satisfy the following rules:

(a) No container can hang in the air. If a slot has a container, the slots below it must also have containers.
(b) 20' containers cannot be loaded on top of 40' containers, but one 40' container can be stacked on top of two 20' containers ("russian stacks" or "mixed stacks").
(c) A cell must be either empty or full (both slots occupied).
(d) Stability constraints are expressed in terms of the longitudinal center of gravity (LCG) and the vertical center of gravity (VCG).

Let us consider a ship with capacity Ω TEUs traveling along a trade route consisting of N ports. The ship has B bays, where each bay b contains R^b rows, and each row r of bay b has L^{bc} cells with 2 slots each. Let $\mathcal{B}$ be the index set of the bays, $\mathcal{R}^b$ the index set of the rows in bay b, $\mathcal{L}^{bc}$ the index set of the tiers in row r of bay b, and $\mathcal{X}$ the set of the slots at each cell. The total capacity of the ship can be calculated as $\Omega = \sum_{b\in\mathcal{B}}\sum_{c\in\mathcal{R}^b}\sum_{l\in\mathcal{L}^{bc}}\sum_{x\in\mathcal{X}}(b\times r\times l\times x)$. We consider two sizes of containers S, 20' and 40', and three different weight classes $\mathcal{M}$, depending on whether the containers are light, medium, or heavy. Table 1 shows the notation used in the following sections.

Since the slots of a cell must be both occupied or both empty, the number of empty slots after loading operations at each port has to be even.

4. Integer programming models

In this section we present a *BSW* model, to address the problem considering all the assumptions presented in Section 3, *BW* and *BS* models that address the problem by relaxing some of the assumptions, and finally a *Base* model to address the basic problem.

4.1. A new model with 20' and 40' containers, considering container weights and stability constraints: the BSW model

We first present the *BSW* model, which involves five sets of binary variables: two sets to identify the position of 20' and 40' containers at each port, t and f; two variable sets to identify those containers that are unproductively moved, τ and φ; and a last variable set to identify empty cells, e.

$$t^m_{ij}(p,x) = \begin{cases} 1, & \text{if one 20' container of class } m \text{ with destination port } j \\ & \text{is stowed in slot (p,x) after the loading operations at} \\ & \text{port } i \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

$$\forall i \in \mathcal{O};\quad \forall j \in \mathcal{D}^i;\quad \forall m \in \mathcal{M};\quad \forall p \in \mathcal{P};\quad \forall x \in \mathcal{X}$$

$$f^m_{ij}(p) = \begin{cases} 1, & \text{if one 40' container of class } m \text{ with destination port } j \\ & \text{is stowed in cell } p \text{ after the loading operations at} \\ & \text{port } i \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

$$\forall i \in \mathcal{O};\quad \forall j \in \mathcal{D}^i;\quad \forall m \in \mathcal{M};\quad \forall p \in \mathcal{P};$$

$$\tau_i(p,x) = \begin{cases} 1, & \text{if one 20' container stowed in slot (p,x) at port } i-1 \\ & \text{is relocated at port } i \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

$$\forall i \in \mathcal{O};\quad \forall p \in \mathcal{P};\quad \forall x \in \mathcal{X}$$

Table 1
Notation employed throughout the paper.

Ω	Container ship capacity, which is expressed in terms of the total number of slots.
$\mathcal{N}$	Index set of ports. $\mathcal{N} = \{1, \dots, N\}$
$\mathcal{O}$	Ports where loading operations can occur. $\mathcal{O} = \{1, \dots, N-1\} \subseteq \mathcal{N}$
$\mathcal{O}^r$	Ports where rehandles can occur. $\mathcal{O}^r = \{2, \dots, N-1\} \subseteq \mathcal{O} \subseteq \mathcal{N}$
$\mathcal{D}^i$	Set of possible destination ports for containers loaded at port i. $\mathcal{D}^i = \{i+1, \dots, N\}$
$\mathcal{B}$	Index set of bays. $\mathcal{B} = \{1, \dots, B\}$
$\mathcal{R}^b$	Index set of rows of bay b. $\mathcal{R}^b = \{1, \dots, R^b\}$
$\mathcal{L}^{br}$	Index set of tiers of bay b, row r. $\mathcal{L}^{br} = \{1, \dots, L^{br}\}$
$\mathcal{P}$	Set of cells. $\mathcal{P} = \{p = (b,r,l) : b \in \mathcal{B}, r \in \mathcal{R}^b, l \in \mathcal{L}^{br}\}$
$\mathcal{P}^-$	Set of cells. $\mathcal{P}^- = \{p = (b,r,l) : b \in \mathcal{B}, r \in \mathcal{R}^b, l \in \mathcal{L}^{br} \setminus \{L^{br}\}\}$. If $p = (b,r,l) \in \mathcal{P}^-$, then $p' = (b,r,l+1)$.
$\mathcal{X}$	Index set of slots. $\mathcal{X} = \{1,2\}$
$\mathcal{M}$	Set of weight classes (light, medium and heavy).
$\mathcal{T}^\alpha$	Set of $\alpha \in \{20,40\}$ feet containers.
$\mathcal{T}_i^\alpha$	Set of $\alpha \in \{20,40\}$ feet containers loaded at port i.
T_{ij}^α	Number of $\alpha \in \{20,40\}$ feet containers going from port i to port j.
$T_{ij}^{\alpha m}$	Number of $\alpha \in \{20,40\}$ feet containers of weight class m going from port i to port j.
E_i	Number of empty cells after loading operations at port i.
W_i	Total weight on board after loading operations at port i.
w_m^α	Weight of an $\alpha \in \{20,40\}$ feet container of class m.
d_b	Distance from a cell in bay b to the central longitudinal position of the ship. Distances to the center of bays located on the bow side are represented by positive values while those of bays located on the stern side are represented by negative values.
d_b^x	Distance from slot x in bay b to the central longitudinal position of the ship. Distances to the center of slots in bays located on the bow side are represented by positive values while those of slots in bays located on the stern side are represented by negative values.
d_l	Distance from tier l to the centerline of the ship. Distances of tiers above the centerline are represented by positive values while those of tiers below the centerline are represented by negative values.
$G-, G+$	Longitudinal center of gravity limits. Maximum deviation distance of the LCG from 0, that is, the central longitudinal position of the ship.
$L-, L+$	Vertical center of gravity limits. Maximum deviation distance of the VCG from 0, that is, the centerline of the ship.

$$\varphi_i(p) = \begin{cases} 1, & \text{if one 40' container stowed in cell } p \text{ at port } i-1 \text{ is relocated at port } i \\ 0, & \text{otherwise} \end{cases} \quad (4)$$
$$\forall i \in \mathcal{O}; \quad \forall p \in \mathcal{P};$$

$$e_i(p) = \begin{cases} 1, & \text{if cell } p \text{ is empty after the loading operations at port } i \\ 0, & \text{otherwise} \end{cases} \quad (5)$$
$$\forall i \in \mathcal{O}; \quad \forall p \in \mathcal{P};$$

With these variables, the objective function of the *BSW* model (6) seeks to minimize the total number of rehandles. We weight the 20' and 40' rehandles alike, but a penalty could be applied to the 40' ones by adding a coefficient greater than one to the φ variables.

$$\text{Min} \sum_{i \in \mathcal{O}^r} \sum_{p \in \mathcal{P}} \left(\sum_{x \in \mathcal{X}} \tau_i(p,x) + \varphi_i(p) \right) \quad (6)$$

The *BSW* model constraints are as in Box I.

For the sake of clarity, constraints can be divided into two groups: *Rehandles* and *Storage*. The *Rehandles* group identifies the unproductive moves at each port and involves constraints (7) to (10). When a cell is occupied, before unloading operations at port i, by a 40' container whose destination is not that port and it is no longer there after the loading operations, there has been an unproductive rehandle in that cell by constraints (7). Likewise for 20' containers and slots by constraints (8). If a container going to port i occupies a given cell at that port, then either the cells above it are occupied by containers going to that port or the containers in those positions will be relocated, by constraints (9). Similarly, if a container is relocated at port i, either the upper slot is occupied by a container going to that port or this container is also relocated. Since constraints (9) combine rehandles of 20' and 40' containers, it could happen that one rehandle of a 40' container is incorrectly used, without any 40' container being involved, when two rehandles of 20' containers are needed. Therefore, constraints (10) ensure that rehandles of 40' containers in cell p at port i can only occur if there is one 40' container not going to this port stowed in that cell before unloading operations at port i and the cell is not occupied by 20' containers or by a 40' container with destination port i.

The *Storage* group ensures proper stowage and involves constraints (11) to (21). Constraints (11)–(14) specify the number of containers of each size and type to be loaded in each port. Constraints (11) and (13) do so for the 20' containers and (12) and (14) for the 40' containers. Upon leaving port i, the number of containers to be unloaded at the next port $i+1$ equals the total number of containers going to port $i+1$ loaded since the beginning of the route, by constraints (11) and (12). By constraints (13) and (14), the number of containers to be unloaded at any port after $i+1$, upon leaving port $i+1$, equals the number of containers on board the ship when it leaves port i plus those that will be loaded at $i+1$. Constraints (15) set the number of empty cells. Constraints (16) ensure that a cell stows at most one 40' or one 20' container in its first slot. This is also satisfied for the second slot by constraints (17), which also ensure that a cell stows either two 20' containers or none. Constraints (17) and (18) prevent 20' containers being stacked on top of 40' containers, and no containers can hang because of constraints (19). Following the approximation described in Pacino et al. (2011), ship stability is related to the position of the center of gravity. Constraints (20) and (21) limit the position of the longitudinal center of gravity LCG and of the vertical center of gravitiy VCG within given thresholds.

4.2. A new model considering container weights and stability constraints: the BW model

To consider the stability constraints but only one container size, we present the *BW* model. It uses the same definition for f, φ, and e variables as before. Since it only deals with 40' containers, variables defining positions and rehandles of 20' containers and the stacking and occupancy assumptions described in Section 3, items b and c, are no longer considered.

Rehandles :

$$f^m_{i-1j}(p) - f^m_{ij}(p) \leq \varphi_i(p) \qquad \forall i \in \mathcal{O}^r,\ \forall j \in \mathcal{D}^i,\ \forall m \in \mathcal{M},\ \forall p \in \mathcal{P} \tag{7}$$

$$t^m_{i-1j}(p,x) - t^m_{ij}(p,x) \leq \tau_i(p,x) \qquad \forall i \in \mathcal{O}^r,\ \forall j \in \mathcal{D}^i,\ \forall m \in \mathcal{M},\ \forall p \in \mathcal{P},\ \forall x \in \mathcal{X} \tag{8}$$

$$\sum_{m\in\mathcal{M}} \left(t^m_{i-1i}(p,x) - t^m_{i-1i}(p',x) + f^m_{i-1i}(p) - f^m_{i-1i}(p') \right) + \tau_i(p,x) + \varphi_i(p) \leq \tau_i(p',x) + \varphi_i(p') + e_{i-1}(p') \quad \forall i \in \mathcal{O}^r,\ \forall p \in \mathcal{P}^-,\ \forall x \in \mathcal{X} \tag{9}$$

$$\varphi_i(p) + \sum_{m\in\mathcal{M}} \Big(\sum_{j\in\mathcal{D}^{i-1}} t^m_{i-1j}(p,1) + f^m_{i-1i}(p) \Big) \leq 1 \qquad \forall i \in \mathcal{O} \setminus \{1\},\ \forall p \in \mathcal{P} \tag{10}$$

Storage :

$$\sum_{p\in\mathcal{P}} \sum_{x\in\mathcal{X}} t^m_{i(i+1)}(p,x) = \sum_{k=1}^{i} T^{20m}_{k(i+1)} \qquad \forall i \in \mathcal{O},\ \forall m \in \mathcal{M} \tag{11}$$

$$\sum_{p\in\mathcal{P}} f^m_{i(i+1)}(p) = \sum_{k=1}^{i} T^{40m}_{k(i+1)} \qquad \forall i \in \mathcal{O},\ \forall m \in \mathcal{M} \tag{12}$$

$$\sum_{p\in\mathcal{P}} \sum_{x\in\mathcal{X}} t^m_{ij}(p,x) = \sum_{p\in\mathcal{P}} \sum_{x\in\mathcal{X}} t^m_{(i+1)j}(p,x) - T^{20m}_{(i+1)j} \qquad \forall i \in \mathcal{O} \setminus \{N-1\},\ \forall j \in \mathcal{D}^{i+1},\ \forall m \in \mathcal{M} \tag{13}$$

$$\sum_{p\in\mathcal{P}} f^m_{ij}(p) = \sum_{p\in\mathcal{P}} f^m_{(i+1)j}(p) - T^{40m}_{(i+1)j} \qquad \forall i \in \mathcal{O} \setminus \{N-1\},\ \forall j \in \mathcal{D}^{i+1},\ \forall m \in \mathcal{M} \tag{14}$$

$$\sum_{p\in\mathcal{P}} e_i(p) = E_i \qquad \forall i \in \mathcal{O} \tag{15}$$

$$\sum_{j\in\mathcal{D}^i} \sum_{m\in\mathcal{M}} \left(t^m_{ij}(p,1) + f^m_{ij}(p) \right) + e_i(p) = 1 \qquad \forall i \in \mathcal{O},\ \forall p \in \mathcal{P} \tag{16}$$

$$\sum_{j\in\mathcal{D}^i} \sum_{m\in\mathcal{M}} \left(t^m_{ij}(p,2) - t^m_{ij}(p,1) \right) = 0 \qquad \forall i \in \mathcal{O},\ \forall p \in \mathcal{P} \tag{17}$$

$$\sum_{j\in\mathcal{D}^i} \sum_{m\in\mathcal{M}} \left(t^m_{ij}(p',2) - t^m_{ij}(p,2) \right) \leq 0 \qquad \forall i \in \mathcal{O},\ \forall p \in \mathcal{P}^- \tag{18}$$

$$e_i(p) - e_i(p') \leq 0 \qquad \forall i \in \mathcal{O},\ \forall p \in \mathcal{P}^- \tag{19}$$

$$G^- \leq \frac{\sum_{j\in\mathcal{D}^i} \sum_{m\in\mathcal{M}} \sum_{p=(b,c,l)\in\mathcal{P}} \left(\sum_{x\in\mathcal{X}} d^x_b w^{20}_m t^m_{ij}(p,x) + d_b w^{40}_m f^m_{ij}(p) \right)}{W_i} \leq G^+ \qquad \forall i \in \mathcal{O} \tag{20}$$

$$L^- \leq \frac{\sum_{j\in\mathcal{D}^i} \sum_{m\in\mathcal{M}} \sum_{p=(b,c,l)\in\mathcal{P}} d_l \left(\sum_{x\in\mathcal{X}} w^{20}_m t^m_{ij}(p,x) + w^{40}_m f^m_{ij}(p) \right)}{W_i} \leq L^+ \qquad \forall i \in \mathcal{O} \tag{21}$$

Box I.

The *BW* model is:

$$\text{Min} \sum_{i\in\mathcal{O}^r} \sum_{p\in\mathcal{P}} \varphi_i(p) \tag{22}$$

s.t.

$$f^m_{i-1j}(p) - f^m_{ij}(p) \leq \varphi_i(p) \qquad \forall i \in \mathcal{O}^r,\ \forall j \in \mathcal{D}^i,\ \forall m \in \mathcal{M},\ \forall p \in \mathcal{P} \tag{23}$$

$$\sum_{m\in\mathcal{M}} \left(f^m_{i-1i}(p) - f^m_{i-1i}(p') \right) + \varphi_i(p) \leq \varphi_i(p') + e_i(p') \quad \forall i \in \mathcal{O}^r,\ \forall p \in \mathcal{P}^- \tag{24}$$

$$\sum_{p\in\mathcal{P}} f^m_{i(i+1)}(p) = \sum_{k=1}^{i} T^{40m}_{k(i+1)} \qquad \forall i \in \mathcal{O},\ \forall m \in \mathcal{M} \tag{25}$$

$$\sum_{p\in\mathcal{P}} f^m_{ij}(p) = \sum_{p\in\mathcal{P}} f^m_{(i+1)j}(p) - T^{40m}_{(i+1)j} \quad \forall i \in \mathcal{O} \setminus \{N-1\},\ \forall j \in \mathcal{D}^{i+1},\ \forall m \in \mathcal{M} \tag{26}$$

$$\sum_{p\in\mathcal{P}} e_i(p) = E_i \qquad \forall i \in \mathcal{O} \tag{27}$$

$$\sum_{j\in\mathcal{D}^i} \sum_{m\in\mathcal{M}} f^m_{ij}(p) + e_i(p) = 1 \qquad \forall i \in \mathcal{O},\ \forall p \in \mathcal{P} \tag{28}$$

$$e_i(p) - e_i(p') \leq 0 \qquad \forall i \in \mathcal{O},\ \forall p \in \mathcal{P}^- \tag{29}$$

$$G^- \leq \frac{\sum_{j\in\mathcal{D}^i} \sum_{m\in\mathcal{M}} \sum_{p=(b,c,l)\in\mathcal{P}} d_b w^{40}_m f^m_{ij}(p)}{W_i} \leq G^+ \qquad \forall i \in \mathcal{O} \tag{30}$$

$$L^- \leq \frac{\sum_{j\in\mathcal{D}^i} \sum_{m\in\mathcal{M}} \sum_{p=(b,c,l)\in\mathcal{P}} d_l w^{40}_m f^m_{ij}(p)}{W_i} \leq L^+ \qquad \forall i \in \mathcal{O} \tag{31}$$

The formulation is similar to that of the previous model but without considering the loading of 20' containers. The objective function (22) minimizes the number of rehandles. Constraints (23)–(24) identify them and constraints (25)–(27) identify the number of containers for each destination and in each weight class that must be on board when leaving each port as well as the number of empty slots. Constraints (28)–(29) enforce the rules regarding stowage: each slot holds at most one container and no containers can hang in the air. Finally, constraints (30)–(31) handle the stability of the ship.

4.3. A new model with 20' and 40' containers and no stability constraints: the BS model

We now present the *BS* model, which considers two container sizes, 20' and 40' containers, and size-related constraints but no weight-related constraints such as stability constraints. It uses the definition of the τ and φ variables in the *BSW* model. With respect to variables

describing the position of 20' and 40' containers, the super-index m for the weight class is no longer needed, as follows:

$$t_{ij}(p,x) = \begin{cases} 1, & \text{if one 40' container with destination port } j \text{ is stowed in slot (p,x) after the loading operations at port } i \\ 0, & \text{otherwise} \end{cases} \quad (32)$$

$\forall i \in \mathcal{O}; \quad \forall j \in D^i; \quad \forall p \in \mathcal{P}; \quad \forall x \in \mathcal{X}$

$$f_{ij}(p) = \begin{cases} 1, & \text{if one 40' container with destination port } j \text{ is stowed in cell } p \text{ after the loading operations at port } i \\ 0, & \text{otherwise} \end{cases} \quad (33)$$

$\forall i \in \mathcal{O}; \quad \forall j \in D^i; \quad \forall p \in \mathcal{P};$

As container weights are not included in this model, we can assume, without loss of generality, that the ship is full after loading operations at each port i by adding E_i containers of 40' from port i to port $i+1$. These containers will not produce rehandles and will be assigned to the upper positions. So, in this model we do not need to declare e variables and we are considering $F_{i,i+1}$ as $F_{i,i+1} + E_i \quad \forall i \in \mathcal{O}$. See Eq. (34) to Eq. (45) in Box II.

The objective function (34) minimizes the number of rehandles of 20' and 40' containers. Constraints (35)–(38) identify the unproductive moves of 20' and 40' containers at each port and constraints (39)–(45) ensure that all containers are properly loaded and stored in a similar way to the *BSW* model but without considering the stability constraints.

4.4. A base model with 40' containers and no stability constraints: the base model

We finally present the *Base* model, which addresses the problem by considering a single container size (40') and no stability constraints. It uses the definition of the f and φ variables in the *BS* model. Again, we do not need to declare e variables considering $F_{i,i+1}$ as $F_{i,i+1} + E_i \, \forall i \in \mathcal{O}$. We can formulate the problem as follows:

$$\text{Min} \sum_{i \in \mathcal{O}^r} \sum_{p \in \mathcal{P}} \varphi_i(p,x) \quad (46)$$

s.t.

$$f_{i-1j}(p) - f_{ij}(p) \leq \varphi_i(p) \qquad \forall i \in \mathcal{O}^r, \ \forall j \in D^i, \ \forall p \in \mathcal{P} \quad (47)$$

$$f_{i-1i}(p,x) + \varphi_i(p,x) \leq f_{i-1i}(p',x) + \varphi_i(p',x) \qquad \forall i \in \mathcal{O}^r, \ \forall p \in \mathcal{P}^- \quad (48)$$

$$\sum_{p \in \mathcal{P}} f_{i(i+1)}(p) = \sum_{k=1}^{i} T^{40}_{k(i+1)} \qquad \forall i \in \mathcal{O} \quad (49)$$

$$\sum_{p \in \mathcal{P}} f_{ij}(p) = \sum_{p \in \mathcal{P}} f_{(i+1)j}(p) - T^{40}_{(i+1)j} \qquad \forall i \in \mathcal{O} \setminus \{N-1\}, \ \forall j \in D^{i+1} \quad (50)$$

$$\sum_{j \in D^i} f_{ij}(p) = 1 \qquad \forall i \in \mathcal{O}, \ \forall p \in \mathcal{P} \quad (51)$$

The objective function (46) minimizes the number of rehandles along the ship's route. Constraints (47) and (48) identify these unproductive moves. Constraints (49)–(50) set the number of containers to be loaded at each port. Finally, constraints (51) ensure that each cell is occupied by one container.

5. An insert-and-fix matheuristic to solve the generalized *CSPP*

The *BSW* model requires a large number of variables and constraints and solvers such as CPLEX do not provide satisfactory results for real-size instances. The structure of the model, in which the variables can be naturally divided into groups corresponding to the ports on the route, suggests the use of matheuristics such as Insert-and-Fix (Wolsey, 1998), in which, at each step, the variables corresponding to a new port are defined as binary and variables which were binary in previous steps are fixed. Matheuristics of this kind, and others of similar structure such as Relax-and-Fix or Fractional-Relax-and-Fix, have been widely used in lot-sizing and scheduling problems arising in foundries (de Araujo et al., 2008), animal feed plants (Toso et al., 2009), or flat-panel display industries (Lee and Lee, 2020), as well as in other areas such as bin packing problems (Paquay et al., 2018), production–distribution planning (Wei et al., 2017), or production planning and warehouse layout problems (Zhang et al., 2017). One main advantage of these matheuristics is that they are based on the decomposition of mathematical models, so they inherit their main advantages: they are easy for practitioners to implement; they are flexible, allowing constraints to be added or removed to fulfill the requirements of the specific problem being solved; and they are solved by commercial solvers, which have shown a dramatic increase in performance in recent years, often by five or even more orders of magnitude (Bixby, 2002), and are constantly improving. This section presents an Insert-and-Fix matheuristic based on the decomposition of the *BSW* model into simpler models that are easier to solve. It starts from the stowage plan at port $N-1$ (in which we have all the information about what to transport to port N) and goes backwards, including at each iteration a previous port, until the stowage plans for all ports are built. It is combined with constructive heuristics that make it possible to obtain feasible solutions in short running times.

5.1. The Insert-and-Fix algorithm: IF

Insert-and-Fix is an iterative algorithm that solves, at each iteration, an integer linear problem (ILP), which is a subset of the original. In each iteration, the variables that had been declared as binary in the previous iteration are set to the values obtained and new constraints are added together with the binary variables inherent in them. The proposed Insert-and-Fix algorithm, *IF* algorithm, is also combined with a heuristic procedure, StowagePlan(), to provide initial feasible solutions. Fig. 2 shows a schematic diagram of the *IF* algorithm, where π^* represents the best solution found so far and π the solution from which the initial solution provided to the solver is obtained.

In the first iteration, $i = N-1$, the StowagePlan function provides a feasible solution to the entire problem π that initially equals π^*. The ILP solved in this iteration considers only the stowage plans at ports $i-2$, $i-1$, and i, with the aim of minimizing the number of rehandles at $i-1$ and i. The problem therefore consists of: (i) the sets of binary variables t and f to identify the position of 20' and 40' containers and variables e to identify empty cells at ports $i-2$, $i-1$, and i; (ii) variables τ and φ to identify the containers that are unproductively moved at ports $i-1$ and i; and (iii) the Storage and Rehandle constraints from the *BSW* model involving these variables. We give the solver an initial solution taking the corresponding part of π.

In all successive iterations i, such that $i \in \mathcal{O}^r$ and $i \leq N-2$, StowagePlan builds new solutions and, if necessary, updates π and π^*. A new ILP is then solved to obtain the stowage plans at ports $i-2$ and $i-1$, minimizing the number of rehandles at $i-1$ and i. It considers the binary variables t, f, and e at ports $i-2$ and $i-1$, and τ and φ at ports $i-1$ and i, together with the corresponding Storage and Rehandle constraints. Variables t, f, and e at port i are fixed to the values provided by π. Once i equals 2, the algorithm ends with the best solution found π^*.

5.2. Description of the StowagePlan function

The StowagePlan function builds feasible solutions for the complete stowage problem. Algorithm 1 shows its pseudocode. It receives as parameters the iteration i, the solution of the model previously solved, Bv, as well as π^* and π, and returns the updated solutions π^* and π.

When $i = N-1$, the sets Bv, π^* and π are empty. First, the constructive algorithm FinalPort builds the stowage plan of the ship at port $N-1$ (see line 3). Then, taking this stowage plan as its starting point, another constructive algorithm PriorPort generates the stowage plan at port $N-2$. This process is repeated until the stowage plans for all ports are obtained (lines 4 and 5).

$$\text{Min} \sum_{i\in\mathcal{O}^r}\sum_{p\in\mathcal{P}}\Big(\sum_{x\in\mathcal{X}}\tau_i(p,x)+\varphi_i(p)\Big) \tag{34}$$

s.t.

$$f_{i-1j}(p)-f_{ij}(p)\le\varphi_i(p) \qquad \forall i\in\mathcal{O}^r,\ \forall j\in\mathcal{D}^i,\ \forall p\in\mathcal{P} \tag{35}$$

$$t_{i-1j}(p,x)-t_{ij}(p,x)\le\tau_i(p,x) \qquad \forall i\in\mathcal{O}^r,\ \forall j\in\mathcal{D}^i,\ \forall p\in\mathcal{P},\ \forall x\in\mathcal{X} \tag{36}$$

$$t_{i-1i}(p,x)+f_{i-1i}(p)+\tau_i(p,x)+\varphi_i(p)\le t_{i-1i}(p',x)+f_{i-1i}(p')+\tau_i(p',x)+\varphi_i(p') \qquad \forall i\in\mathcal{O}^r,\ \forall p\in\mathcal{P}^-,\ \forall x\in\mathcal{X} \tag{37}$$

$$\varphi_i(p)+\sum_{j\in\mathcal{D}^{i-1}}t_{i-1j}(p,1)+f_{i-1i}(p)\le 1 \qquad \forall i\in\mathcal{O}\setminus\{1\},\ \forall p\in\mathcal{P} \tag{38}$$

$$\sum_{p\in\mathcal{P}}\sum_{x\in\mathcal{X}}t_{i(i+1)}(p,x)=\sum_{k=1}^{i}T^{20}_{k(i+1)} \qquad \forall i\in\mathcal{O} \tag{39}$$

$$\sum_{p\in\mathcal{P}}f_{i(i+1)}(p)=\sum_{k=1}^{i}T^{40}_{k(i+1)} \qquad \forall i\in\mathcal{O} \tag{40}$$

$$\sum_{p\in\mathcal{P}}\sum_{x\in\mathcal{X}}t_{ij}(p,x)=\sum_{p\in\mathcal{P}}\sum_{x\in\mathcal{X}}t_{(i+1)j}(p,x)-T^{20}_{(i+1)j} \qquad \forall i\in\mathcal{O}\setminus\{N-1\},\ \forall j\in\mathcal{D}^{i+1} \tag{41}$$

$$\sum_{p\in\mathcal{P}}f_{ij}(p)=\sum_{p\in\mathcal{P}}f_{(i+1)j}(p)-T^{40}_{(i+1)j} \qquad \forall i\in\mathcal{O}\setminus\{N-1\},\ \forall j\in\mathcal{D}^{i+1} \tag{42}$$

$$\sum_{j\in\mathcal{D}^i}\Big(t_{ij}(p,1)+f_{ij}(p)\Big)=1 \qquad \forall i\in\mathcal{O},\ \forall p\in\mathcal{P} \tag{43}$$

$$\sum_{j\in\mathcal{D}^i}\Big(t_{ij}(p,2)-t_{ij}(p,1)\Big)=0 \qquad \forall i\in\mathcal{O},\ \forall p\in\mathcal{P} \tag{44}$$

$$\sum_{j\in\mathcal{D}^i}\Big(t_{ij}(p',2)-t_{ij}(p,2)\Big)\le 0 \qquad \forall i\in\mathcal{O},\ \forall p\in\mathcal{P}^- \tag{45}$$

Box II.

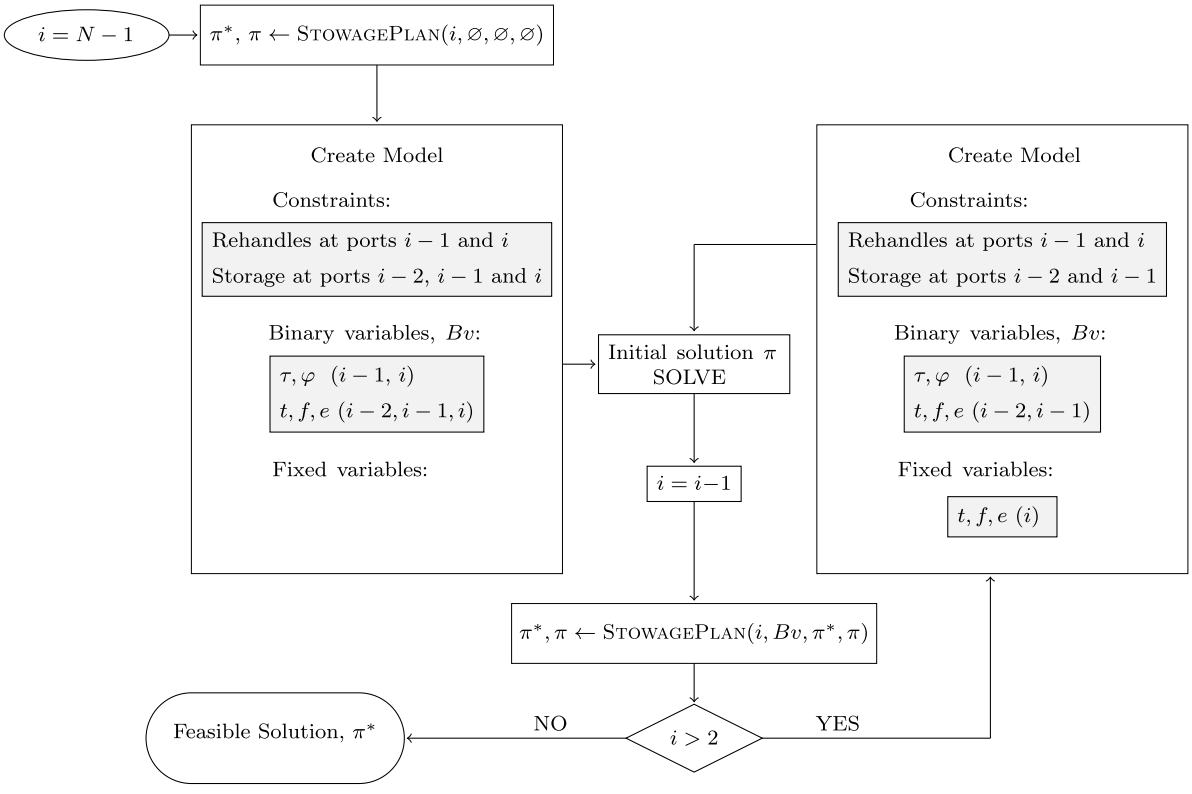

Fig. 2. Schematic figure of the Insert-and-Fix algorithm.

Algorithm 1 A heuristic algorithm to provide initial solutions for the solvers.

```
1: function StowagePlan(i, Bv, π*, π)
2:     if i = N − 1 then
3:         π^{N−1} ← FinalPort(T_{N−1}, F_{N−1})          ▷ Let π^i be the stowage plan of the ship when leaving port i
4:         for k = i to k = 2 do
5:             π^{k−1} ← PriorPort(k, π^k, T, F)
6:         π* ← π
7:     else
8:         π_1 ← π;   π_2 ← π;
9:         π_1^i ← Bv^i;   π_2^i ← Bv^i;                  ▷ Let Bv^i be the stowage plan at port i given by the ILP
10:        π_2^{i−1} ← Bv^{i−1}
11:        for k = i to k = 2 do
12:            π_1^{k−1} ← PriorPort(k, π_1^k, T, F)
13:        for k = i − 1 to k = 2 do
14:            π_2^{k−1} ← PriorPort(k, π_2^k, T, F)
15:        if R(π_1) ≤ R(π_2) then
                                                          ▷ Let R(π) the number of total rehandles produced by solution π
16:            if R(π_1) < R(π*) then π* ← π_1
17:            if R(π_1^{i+1}) < R(π^{i+1})  or  [R(π_1^{i+1}) = R(π^{i+1})  and  R(π_1) < R(π)] then π ← π_1
18:        else
19:            if R(π_2) < R(π*) then π* ← π_2
20:            if R(π_2^{i+1}) < R(π^{i+1})  or  [R(π_2^{i+1}) = R(π^{i+1})  and  R(π_2) < R(π)] then π ← π_2
21:    return π*, π
```

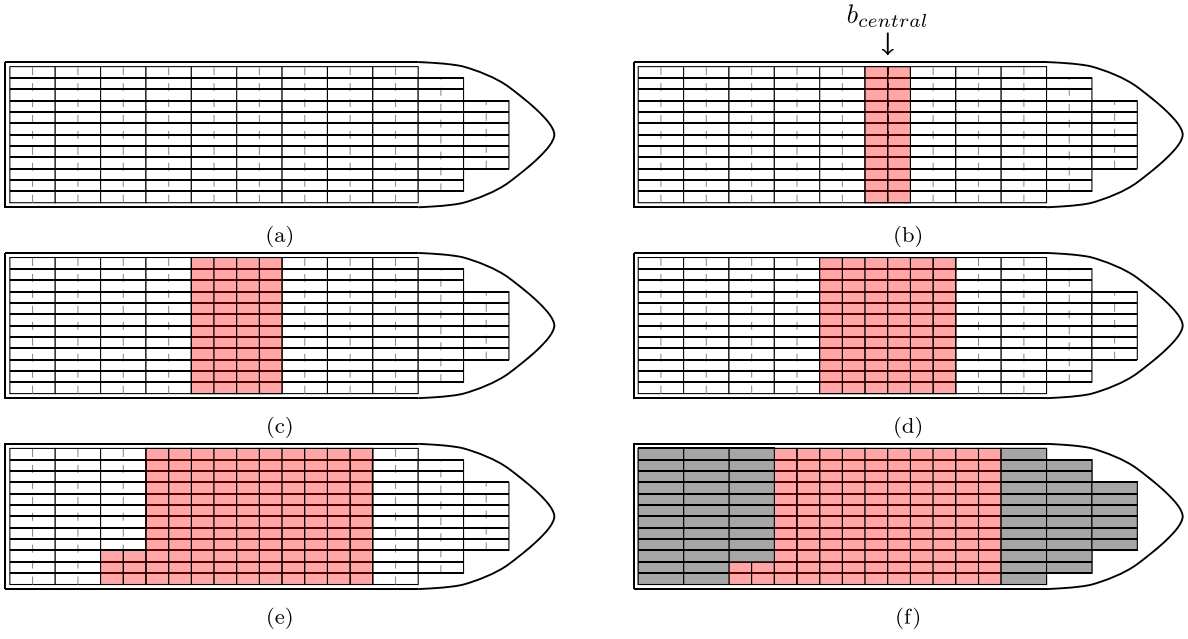

Fig. 3. Example showing the construction phase used in FinalPort() function. The 20' containers are highlighted in red and 40' containers in gray.

For the remaining ports two new solutions π_1 and π_2 are created. Both share with π the stowage plans at ports $N-1,\dots,i+1$. Their stowage plans at port i are given by the solution of the ILP solved in the previous iteration, Bv^i. As regards π_1, the stowage plans at ports $i-1,\dots,1$ are built by using PriorPort. As regards π_2, the stowage plan at port $i-1$ is also taken from the ILP and all other stowage plans at ports $i-2,\dots,1$ are built by PriorPort. If π_1 or π_2 improves on the best solution found in terms of the number of rehandles, π^* is updated in lines 16 and 19. Let π_1 (or π_2) be the solution with the least number of rehandles. If it produces a lower number of rehandles than π at port $i+1$, or produces the same rehandles at port $i+1$ and has a lower overall number, then π is updated to π_1 (or π_2) in line 17 (or 20). The idea behind this is simple. At each iteration we aim to set the best stowage plan at port i, so we keep the one that produces the fewest rehandles at $i+1$, regardless of its total number of rehandles. The objective is to reduce this number step by step, each time setting a port with the least possible number of rehandles.

5.2.1. FinalPort: *Obtaining the stowage plan of the ship when leaving port* $N-1$

The constructive algorithm used to obtain the stowage plan in the last port $N-1$ works in two phases: construction and repair. The

construction phase starts by placing the 20' containers from heavier to lighter. When there are no 20' containers left, it places the 40' containers also from heavier to lighter. It is carried out starting from the bay that occupies the central position in the ship, $b_{\text{central}} = \frac{B+1}{2}$, filling each of its rows one by one. Once all the rows of the central bay have been filled, the construction phase fills the remaining bays from the central bay to the stern and bow of the ship. First bays $b_{\text{central}}-1$ and $b_{\text{central}}+1$, then $b_{\text{central}}-2$ and $b_{\text{central}}+2$, ..., until bays 1 and B are filled up or no containers are left. If the number of bays is even, b_{central} is not integer, so it would start by filling bays $\lceil b_{\text{central}}-1\rceil$ and $\lfloor b_{\text{central}}+1\rfloor$, then $\lceil b_{\text{central}}-2\rceil$ and $\lfloor b_{\text{central}}+2\rfloor$, and so on up to bays 1 and B. Fig. 3 shows an example of how this construction phase works. The ship is initially empty (Fig. 3(a)) and the algorithm starts filling each of the central bay rows (Fig. 3(b)) with 20' containers. Once all the rows of the central bay have been filled, it fills bay $b_{central}-1$ (Fig. 3(c)) and then bay $b_{central}+1$ (Fig. 3(d)). The algorithm continues filling bays $b_{central}-2$, $b_{central}+2$, and $b_{central}-3$ with 20' containers until there are no 20' containers left (Fig. 3(e)). At that stage, it starts placing the 40' containers also from heavier to lighter. If there are free tiers in the row and bay where it has run out of 20' containers, they are filled up with 40' containers (Fig. 3(f)).

The repair phase checks whether the stowage plan meets the stability conditions. First, it checks whether the vertical center of gravity, VCG, is below the centerline of the vessel. When this is not the case, movements are made until this condition is satisfied following the RepairVCG() function described in Algorithm 2. First, the bays are explored looking for empty cells below the centerline and occupied cells above the centerline on lines 2–8. If they are found, containers placed above the centerline are shifted to empty cells below the centerline on lines 9–11. Then, if VCG is not yet within the limits, changes are made between heavier containers above the centerline and lighter containers below the centerline that are in the same bay and of the same size. For each bay, the algorithm builds two lists (lines 13–14), HC to store the information of heavy and medium weight containers placed above the centerline and LC to store the information of light and medium weight containers placed below the centerline. Tiers below the centerline of all rows in each bay are explored (line 16), looking for cells or slots occupied by light or medium weight containers (lines 19–20 and 27–28). Simultaneously, the upper tiers of the rows in the same bay are explored (line 17), looking for cells or slots occupied by heavy or medium weight containers (lines 21–22 and 29–30). As soon as there are 40' containers in both LC and HC, i.e., $LC[0]$ and $HC[0]$ are not empty, their positions are exchanged on line 24. Similarly, when both LC and HC have 20' containers located in slots of same index, $LC[1]$ and $HC[1]$ or $LC[2]$ and $HC[2]$, their positions are exchanged on line 32. Then, the repair phase checks whether the LCG is within the required limits, otherwise a similar procedure is used exchanging heavier and lighter containers in the same tier and of the same size between bays, until the longitudinal stability conditions are satisfied.

5.2.2. PriorPort: *Obtaining the stowage plan at port $k-1$ given the stowage plan at port k*

The PriorPort function provides the stowage plan at port $k-1$. It initializes it to that at port k and then adjusts the plan to suit port $k-1$, removing containers to be loaded at port k and adding those to be unloaded at port k. Let $\mathcal{T}_k$ and $\mathcal{F}_k$ be the sets of 20' and 40' containers loaded at port k, $next_{\mathcal{T}}$ and $next_{\mathcal{F}}$ the sets of 20' and 40' containers to be unloaded at port k ordered by non-increasing weight, and $rem_{\mathcal{T}}$ and $rem_{\mathcal{F}}$ empty sets that will be used to store temporary container movements.

All tiers are traversed from top to bottom. At tier t, all its slots are explored, first the slots of the central bay $b_{\text{central}} = \frac{B+1}{2}$, then those of bay $b_{\text{central}}-1$, those of bay $b_{\text{central}}+1$, and so on up to bay B. If the number of bays is even, it starts with bay $\lceil b_{\text{central}}-1\rceil$, then moves on to $\lfloor b_{\text{central}}+1\rfloor$, and so on up to bay B. Each time a slot or cell occupied by a container in $\mathcal{T}_k$ or in $\mathcal{F}_k$ is found, it is removed from the stowage plan and from the corresponding set and the following steps are carried out:

- Step 1. Go through all the slots above tier t in the same bay and row, removing all existing containers. The 20' containers are added to set $rem_{\mathcal{T}}$ and the 40' containers to set $rem_{\mathcal{F}}$ in non-increasing weight order.
- Step 2. Go through all the tiers below tier t, from tier $t-1$ to 1, in the same bay and row. If a container in the given tier is in $\mathcal{T}_k$ or $\mathcal{F}_k$, it is removed, updating the corresponding set, and the next tier, i.e., the tier below, is explored. Otherwise, the lower tiers are no further explored. Consequently, no containers are left hanging in the stowage plan at any time. If one 20' container is removed from a cell in slot 1 but the container in slot 2 cannot be removed, then the algorithm will only explore the lower slots with index 1.
- Step 3. Fill up the slots that have been left empty in that bay and row, as follows:
  - Step 3.1. If there are 40' containers in the row, but not enough 40' containers in sets $rem_{\mathcal{F}}$ and $next_{\mathcal{F}}$ to fill up the row, all the 40' containers are removed and added to set $rem_{\mathcal{F}}$ in non-increasing weight order. Then go to Step 3.3. If there are 40' containers in the row and there are enough 40' containers in sets $rem_{\mathcal{F}}$ and $next_{\mathcal{F}}$ to fill it, then go to Step 3.4.
  - Step 3.2. If there are cells in the row occupied by just one 20' container, fill in the other slots by assigning first containers from $rem_{\mathcal{T}}$ until it is emptied and then from $next_{\mathcal{T}}$. If it is not possible to fill all these cells, remove as many containers as necessary from the uncompleted cells to fill the maximum number of cells.
  - Step 3.3. Complete as many cells as possible by assigning 20' containers, first from set $rem_{\mathcal{T}}$ and then from $next_{\mathcal{T}}$. First the slots of index 1 are filled and later those of index 2.
  - Step 3.4. Complete the row by assigning 40' containers, first from set $rem_{\mathcal{F}}$ and then from $next_{\mathcal{F}}$.

The procedure ends when all the sets, $\mathcal{T}_k$, $\mathcal{F}_k$, $next_{\mathcal{T}}$, $next_{\mathcal{F}}$, $rem_{\mathcal{T}}$, $rem_{\mathcal{F}}$, are empty. This algorithm also uses the repair phase explained in Section 5.2.1 to ensure that the stowage plan provided meets the stability conditions.

6. Computational experiments

We conducted an extensive computational analysis to test the performance of the integer programming models and the matheuristic proposed in this paper. They were coded in C++ and executed on virtual machines with 4 virtual processors and 16 GBytes of RAM memory each. The virtual machines run Windows 10 Enterprise 64 bits. Virtual machines are 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 GBytes of RAM. All the algorithms and models were run on a single thread using CPLEX 12.10 as a solver.

6.1. Test instances

We use the 405 instances from Roberti and Pacino (2018), in which the ship is full in all ports. These instances are divided into 5 groups: Long, Mixed, Short, Authentic, and Required. The number of ports between the port of load and the port of discharge of containers is on average large in Long instances, small in Short instances, and follows a more random distribution in Mixed and Authentic instances. The Required instances are more challenging because they are generated to force rehandles. Since these instances were generated to test the basic CSPP with all containers of the same size and no stability conditions,

Algorithm 2 A heuristic algorithm to repair solutions not satisfying the VCG limits.

function RepairVCG(π^i)
  for $b \in \mathcal{B}$ **do**
    E^b, $F^b \leftarrow \emptyset$
    for $r \in \mathcal{R}^b$ **do**
      if There are empty cells below the centerline **then**
        Add r to E^b sorted by increasing number of containers placed in the rows.
      else
        Add r to F^b sorted by non increasing weight of containers at the row above the centerline
  for $b \in \mathcal{B}$ **do**
    for $e \in E^b$ **do**
      Fill e up to the centerline with containers of rows in F^b and **if** VCG into the limits **then** END
  for $b \in \mathcal{B}$ **do**
    $HC \leftarrow \emptyset$ ▷ Heavy and medium weight containers placed in cells or slots above the centerline
    $LC \leftarrow \emptyset$ ▷ Light and medium weight containers placed in cells or slots below the centerline
    $L \leftarrow \max\{L^{br} : r \in \mathcal{R}^b\}$
    for $t_{below} = 1$ **to** $t_{below} < centerline$ **do**
      $t_{upper} \leftarrow L - t_{below} + 1$
      for $r \in \mathcal{R}^b$ **do**
        if $weightclass(b, r, t_{below}) = Light$ **or** $weightclass(b, r, t_{below}) = Medium$ **then**
          Add info to $LC[0]$
        if $weightclass(b, r, t_{upper}) = Heavy$ **or** $weightclass(b, r, t_{upper}) = Medium$ **then**
          Add info to $HC[0]$
        if $HC[0] <> \emptyset$ **and** $LC[0] <> \emptyset$ **then**
          Exchange positions from containers in $HC[0]$ and $LC[0]$
          if VCG into the limits **then** END
        for $x \in \mathcal{X}$ **do**
          if $weightclass(b, r, x, t_{below}) = Light$ **or** $weightclass(b, r, x, t_{below}) = Medium$ **then**
            Add info to $LC[x]$
          if $weightclass(b, r, x, t_{upper}) = Heavy$ **or** $weightclass(b, r, x, t_{upper}) = Medium$ **then**
            Add info to $HC[x]$
          if $HC[x] <> \emptyset$ **and** $LC[x] <> \emptyset$ **then**
            Exchange positions from containers in $HC[x]$ and $LC[x]$
            if VCG into the limits **then** END
return π^i

the ship is seen as a single bay with l tiers and r rows. The number of ports varies between $N = 6, 8, 10$, the number of tiers between $L = 6, 8, 10$, and the number of rows between $R = 100, 300, 500$. These sets of instances are used in Section 6.2, in which we compare the *Base* model with other exact methods previously proposed in the literature for the basic CSPP.

The models and algorithms tested in Sections 6.3 and 6.4 require a more complex ship structure. Instead of considering a single bay, we consider $B = 10$ bays with a number of rows varying between $R = 5, 15, 25$. Each position defined by bay-row-tier can contain one 40' or two 20' containers. Therefore the ship capacity ranges from 600 to 5000 TEUs. In addition, the original instances needed to be adapted to consider weight- and size-related constraints. We generated four new sets of instances:

- TR_1: Only considering 40' containers.
- TR_2: Considering 40' containers and different container weights: 1/3 of the containers on board are light, 1/3 medium, and 1/3 heavy.
- TR_3: Dividing the total number of TEUs between 20' and 40' containers. In this case, the loading percentage of 20' and 40' containers varies. In one third of the instances, 25% of the TEUs are occupied by 20' containers and 75% by 40' containers. In another third, 75% are occupied by 20' containers and 25% by 40' containers. The remaining instances have a 50%–50% occupancy rate.
- TR_4: Adding weights to TR_3 with rates of light, medium, and heavy containers being 1/3-1/3-1/3.

The weights of the containers are set according to Table 2. When stability conditions are included, we assume that the LCG should not deviate from the central bay (or the midpoint between the two central bays if the number of bays is even) by more than 5% of the total length of the ship. Therefore, in the instances used we consider G = 20 feet. We take the line that crosses the half height of the rows as the centerline of the ship.

Table 2
Weights of the containers in tonnes according to their size S and weight class $\mathcal{M}$.

	$\mathcal{M}$		
S	Light	Medium	Heavy
20'	7	14	21
40'	10	20	30

6.2. Comparing the base model with existing exact methods for the basic CSPP

We compare the performance of the *Base* model with the mathematical formulations proposed by Avriel et al. (1998) and by Parreño-Torres et al. (2019), and the exact approach by Roberti and Pacino (2018), which combines a column generation-based lower bounding procedure with an MIP model. We refer to them as *APSW*, *PAP*, and *RP*, respectively. An alternative formulation was also proposed by Ding and Chou (2015) but it is less efficient than *APSW* and *PAP* (Parreño-Torres et al., 2019; Roberti and Pacino, 2018), so it is not tested in this comparison. To compare the *Base* model with the existing methods, we have to adapt the definition of the binary variables and assume that each bay-row coordinate holds a 20' container, as the other methods do. We coded both the Avriel et al. (1998) and the Parreño-Torres et al. (2019) models and used the original code of *RP* kindly provided by the authors. All results except *RP* were obtained with CPLEX 12.10. In the case of *RP*

Table 3
Comparison between the *Base* model and the methods proposed by Avriel et al. (1998) [*APSW*], Parreño-Torres et al. (2019) [*PAP*], and Roberti and Pacino (2018) [*RP*], on the instances from Roberti and Pacino (2018).

			APSW		*PAP*		*RP*		*Base*			*APSW*	*PAP*	*RP*	*Base*
	N	#	O	F	O	F	O	F	O	F	#OS	CPU	CPU	CPU	CPU
Short															
	6	27	27	27	27	27	27	27	27	27	27	52	34	1	8
	8	27	27	27	27	27	27	27	27	27	27	245	310	4	62
	10	27	24	27	25	25	27	27	27	27	24	569	874	16	284
		81	78	81	79	79	81	81	81	81	78	278	388	7	111
Mixed															
	6	27	27	27	27	27	27	27	27	27	27	80	50	1	7
	8	27	27	27	27	27	27	27	27	27	27	339	341	5	57
	10	27	24	27	23	23	27	27	27	27	23	670	916	19	276
		81	78	81	77	77	81	81	81	81	77	347	411	8	105
Authentic															
	6	27	27	27	27	27	27	27	27	27	27	74	80	1	14
	8	27	27	27	27	27	27	27	27	27	27	407	771	5	190
	10	27	23	27	16	18	27	27	25	27	16	916	1231	60	402
		81	77	81	70	72	81	81	79	81	70	395	610	16	171
Long															
	6	27	27	27	27	27	27	27	27	27	27	71	55	1	13
	8	27	27	27	27	27	27	27	27	27	27	428	383	8	109
	10	27	22	27	22	22	27	27	27	27	21	743	915	26	324
		81	76	81	76	76	81	81	81	81	75	388	414	11	135
Required															
	6	27	0	27	8	27	27	27	10	27	0	-	-	-	-
	8	27	0	27	3	22	19	27	3	27	0	-	-	-	-
	10	27	1	27	2	11	15	25	1	27	1	162	177	10	110
		81	1	81	13	60	61	79	14	81	1	162	177	10	110
Total/Avg		405	310	405	315	364	385	403	336	405	301	350	451	10	129

Columns "N", "#", "O", and "F" indicate number of ports, instances tested in each group, and optimal and feasible solutions obtained, respectively. Column "#OS" represents the number of instances optimally solved by the 4 methods and the "CPU" columns show the average running time (in seconds) on those instances. The "Total/Avg." row indicates the totals of the columns that contain absolute frequencies and the averages of the columns that contain average values. Rows with similar information are available for each of the five groups of instances. The notation "-" is used when the averages cannot be calculated.

we used CPLEX 12.7.1, as the code provided by the authors included functions that are not available in CPLEX 12.10 and the adaptations were far from straightforward. A maximum running time of one hour per instance was imposed on each method.

Table 3 shows the comparison between the *Base*, *APSW*, and *PAP* models as well as the *RP* method on the 405 instances from Roberti and Pacino (2018) grouped by type of instances (Short, Mixed, Authentic, Long, and Required) and by number of ports N. The *Base* model reaches a solution in all the tested instances and obtains a greater number of optimal solutions than *APSW* and *PAP* (336 versus 310 and 315). It optimally solves all the instances in the Short, Mixed, and Long groups, all but two instances in the Authentic group, but only 14 out of 81 Required instances. A greater number of optimal solutions is obtained by *RP*, reaching 385, even though it does not provide a feasible solution in 2 instances. The last 4 columns in Table 3 show the average running time in the 301 instances optimally solved by the four methods. The *Base* model is 63% faster than the *APSW* model and 71% than the *PAP* model. Nevertheless, *RP* outperforms the *Base* model with an average running time of just 10 s.

In the light of the results, the proposed model improves on the alternative *APSW* and *PAP* models, although the best exact method for the basic CSPP problem is *RP*. However, the *RP* model cannot be easily modified to add new conditions, except if they can be handled in the pricing problem. On the other hand, the here proposed MIP model can be easily extended to handle additional constraints (although this may increase solution times).

6.3. Performance of the proposed IP models for the basic CSPP and its generalizations

We aim to evaluate how the simplified problem, which only considers 40' containers with no stability constraints, becomes more and more complex as we add different container sizes as well as container weights and stability constraints. Table 4 shows the results obtained by the integer programming models, $Base$, BW, BS, and BSW, on the instance sets TR_1, TR_2, TR_3, and TR_4, considering a time limit of one hour per instance.

With respect to the simplified problem, the *Base* model finds a feasible solution in every single instance tested. It optimally solves all the instances in the Short, Mixed, and Authentic groups with zero rehandles and an average running time lower than 75 s. All but one of the Long instances were also optimally solved. However, only 26 out of the 81 instances in the Required group were optimally solved, with an average of 1.4 rehandles. The high number of rehandles indicated in column "R" explains the poor quality of the feasible solutions when optimality is not ensured. If we study the problem considering container weight and stability constraints, the *BW* model obtains solution in 301 of the 405 instances, solving 238 of them to optimality. It can be seen that as the number of ports increases, the difficulty of the problem also increases. The model that considers different container sizes and no stability constraints, *BS*, performs worse than the *BW* model because of the stacking constraints. It reaches optimal and feasible solutions in 194 and 283 instances out of 405 respectively. The problem considering both different container sizes and stability constraints is much more difficult. The *BSW* model only solves 74 instances and just 2 instances of 10 ports and 5 of 8 ports. From Table 4 it can be concluded that the inclusion of container sizes has a stronger effect than the container

Table 4
Performance of the *Base*, *BW*, *BS*, and *BSW* models on Short, Mixed, Authentic, Long, and Required instances, grouped by number of ports N.

			Base				*BW*				*BS*				*BSW*			
	N	#	O	F	R	CPU	O	F	R	CPU	O	F	R	CPU	O	F	R	CPU
Short																		
	6	27	27	27	0.0	3	27	27	0.0	83	27	27	0.0	321	16	16	0.0	871
	8	27	27	27	0.0	20	22	22	0.0	816	18	20	1.0	1339	4	4	0.0	1396
	10	27	27	27	0.0	107	9	9	0.0	623	6	11	22.0	2548	0	1	6.0	3600
		81	81	81	0.0	44	58	58	0.0	445	51	58	4.5	1094	20	21	0.3	1101
Mixed																		
	6	27	27	27	0.0	3	27	27	0.0	81	27	27	0.0	187	15	15	0.0	1042
	8	27	27	27	0.0	18	25	25	0.0	731	19	22	85.2	1424	1	1	0.0	2339
	10	27	27	27	0.0	123	15	16	0.2	1444	5	12	484.8	2732	0	0	–	–
		81	81	81	0.0	48	67	68	0.0	641	51	61	126.1	1134	16	16	0.0	1123
Authentic																		
	6	27	27	27	0.0	6	27	27	0.0	178	27	27	0.0	421	11	12	0.7	1639
	8	27	27	27	0.0	53	19	20	0.1	1175	12	17	269.2	2647	0	0	–	–
	10	27	27	27	0.0	337	2	9	10.9	3325	0	7	1737.1	3600	0	0	–	–
		81	81	81	0.0	132	48	56	1.8	1040	39	51	328.2	1599	11	12	0.7	1639
Long																		
	6	27	27	27	0.0	5	27	27	0.0	134	26	27	0.0	463	13	13	0.0	1105
	8	27	27	27	0.0	43	24	25	0.2	1056	18	22	1.4	2053	0	0	–	–
	10	27	26	27	0.0	362	10	13	0.8	1726	3	11	429.6	2875	1	1	0.0	2161
		81	80	81	0.0	137	61	65	0.2	807	47	60	79.3	1489	14	14	0.0	1181
Required																		
	6	27	17	27	2.1	1429	3	27	232.9	3225	4	27	198.3	3114	1	11	3.3	3397
	8	27	3	27	4.4	3273	1	18	4.6	3412	2	17	792.9	3251	0	0	–	–
	10	27	6	27	285.2	2950	0	9	16.8	3600	0	9	1904.7	3600	0	0	–	–
		81	26	81	97.2	2551	4	54	120.8	3350	6	53	678.8	3240	1	11	3.3	3397
Total/Avg		405	349	405	19.5	582	238	301	22.1	1199	194	283	231.2	1679	62	74	0.7	1549

Columns "N", "#", "O", and "F" indicate number of ports, instances tested in each group, and optimal and feasible solutions obtained, respectively. The "R" and "CPU" columns represent the average number of rehandles and the average running time (in seconds) on instances in which a feasible solution is obtained. The "Total/Avg." row indicates the totals of the columns that contain absolute frequencies and the averages of the columns that contain average values. Rows with similar information are available for each of the five groups of instances. The notation "-" is used when the averages cannot be calculated.

Table 5
Performance of the *IF* algorithm on the instances of set TR_4 solved and not solved by *BSW* model, grouped by number of ports N.

				BSW		*IF*			*IF*	
	N	#	F	R	CPU	R	CPU	NSF	R	CPU
Short	6	27	16	0.0	871	0.7	360	11	0.0	827
	8	27	4	0.0	1396	1.5	368	23	22.2	2131
	10	27	1	6.0	3600	1.0	408	26	91.2	2992
		81	21	0.3	1101	0.9	364	60	48.1	2265
Mixed	6	27	15	0.0	1042	0.0	39	12	0.0	163
	8	27	1	0.0	2339	0.0	36	26	12.3	2038
	10	27	0	–	–	–	–	27	40.7	2303
		81	16	0.0	1123	0.0	39	65	21.8	1802
Authentic	6	27	12	0.7	1639	2.8	1390	15	2.0	1287
	8	27	0	–	–	–	–	27	19.2	2665
	10	27	0	–	–	–	–	27	55.8	3384
		81	12	0.7	1639	2.8	1390	69	29.8	2647
Long	6	27	13	0.0	1105	3.5	1086	14	6.1	1142
	8	27	0	–	–	–	–	27	38.0	2854
	10	27	1	0.0	2161	2.0	1218	26	114.8	3220
		81	14	0.0	1181	3.4	1096	67	61.1	2638
Required	6	27	11	3.3	3397	34.4	3236	16	68.9	3600
	8	27	0	–	–	–	–	27	135.0	3511
	10	27	0	–	–	–	–	27	256.9	3563
		81	11	3.3	3397	34.4	3236	70	166.9	3551
Total/Avg		405	74	0.7	1549	6.5	1025	331	66.9	2601

Columns "N" and "#" indicate the number of ports and number of instances tested in each group. The "F" and "NSF" columns represent the instances solved and not solved by *BSW* model, and columns "R" and "CPU" represent the average number of rehandles and the average running time (in seconds). The "Total/Avg." row indicates the totals of the columns that contain absolute frequencies and the averages of the columns that contain average values. Rows with similar information are available for each of the five groups of instances.

weights and stability constraints, and that the combination of both characteristics makes the resulting model very difficult to solve.

6.4. Performance of the proposed matheuristic algorithm: IF algorithm

We evaluate the matheuristic *IF* algorithm on the TR_4 instances; the results are shown in Table 5. We differentiate between the instances in which the model obtains a solution and those in which no solution is obtained. Out of the 74 instances solved by the *BSW* model, the *IF* algorithm solves the 16 in the Mixed group with an average of zero rehandles, the 21 in the Short group with an average of 0.9 rehandles, and the 12 and 14 instances in the Authentic and the Long group with an average of 2.8 and 3.4 rehandles. These values are slightly larger than those obtained by the *BSW* model. However, the average number of rehandles increases considerably in the Required group with 34.4 rehandles. Out of the 331 instances not solved by the *BSW* model, the *IF* algorithm reaches a solution in each of them with an overall average number of rehandles of 66.9.

Table 6 shows the results obtained by the *IF* algorithm considering as a time limit one, two, and four hours (*IF* 1h, *IF* 2h, and *IF* 4h). We assign a maximum time at each iteration that is equal to the overall time remaining divided by the number of iterations still to be performed. The table also shows the solutions obtained by the fully constructive procedure *Heur* and by the *BSW* model with the initial solutions provided by *Heur* and a time limit of one, two, and four hours per instance. We refer to the model with an initial solution provided as *BSW**. The *Heur* procedure consists of using the StowagePlan function with inputs $i = N - 1$ and empty sets. The *Heur* procedure obtains a solution in all instances, with an average of 128.8 rehandles. Launching the *BSW* model with initial solutions, we see that no satisfactory results are obtained, since in most cases CPLEX does not manage to improve on the initial solution, due to the high number of variables and restrictions involved in the model. It only improves the average number of rehandles by 6.4% in an average running time of 3203 s. If instead with a time limit of one hour, the model is run with two and four hours, the average number of rehandles is cut by 2.7% and 5.4%. The *IF* algorithm obtains an average of 55.8 rehandles in an average

Table 6
Performance of the *IF* algorithm with time limits of one, two, and three hours on the Authentic, Long, Mix, Required, and Short instances, grouped by number of ports *N*.

			Heur	*BSW**			*IF* 1h		*IF* 2h		*IF* 4h	
	N	#	R	R 1h	R 2h	R 4h	R	CPU	R	CPU	R	CPU
Short												
	6	27	35.4	14.6	11.4	10.1	0.4	550.4	0	660	0.4	930
	8	27	110.8	99.0	95.0	79.9	19.1	1869.5	7	3070	5.3	5621
	10	27	235.7	235.7	227.3	212.3	87.9	2896.0	59	5225	23.7	8341
		81	127.3	116.4	111.3	100.8	35.8	1771.9	22	2985	9.8	4964
Mixed												
	6	27	14.5	3.7	3.7	3.7	0.0	94	0.0	92	0.0	93
	8	27	93.5	90.2	85.7	83.8	11.9	1964	11.6	3502	10.9	6597
	10	27	207.6	207.6	207.6	207.6	40.7	2303	30.1	3943	26.8	7521
		81	106.2	101.6	100.0	99.4	17.5	1454	13.9	2512	12.5	4737
Authentic												
	6	27	22.7	16.4	14.5	7.0	2.4	1333	2.0	2473	1.8	4311
	8	27	52.9	52.9	51.5	48.4	19.2	2665	17.1	4750	12.3	8806
	10	27	114.7	114.7	114.7	114.7	55.8	3384	40.0	6625	25.1	11837
		81	63.4	61.3	60.2	56.7	25.8	2460	19.7	4616	13.0	8318
Long												
	6	27	34.4	15.5	12.6	12.0	4.9	1115	4.4	1976	4.4	3709
	8	27	129.5	125.6	121.9	121.9	38.0	2854	34.1	5336	34.7	10546
	10	27	197.0	197.0	195.9	195.8	110.6	3146	97.4	6123	88.3	12168
		81	120.3	112.7	110.1	109.9	51.1	2371	45.3	4478	42.4	8808
Required												
	6	27	113.0	64.9	54.6	49.7	54.9	3452	51.0	6896	49.8	13608
	8	27	211.7	211.7	205.0	205.0	135.0	3511	126.3	6891	122.0	13513
	10	27	356.0	356.0	356.0	356.0	256.9	3563	247.4	7069	222.0	14082
		81	226.9	210.9	205.2	203.6	148.9	3508	141.6	6952	131.3	13734
Total/Avg		405	128.8	120.6	117.4	114.1	55.8	2313	48.5	4309	41.8	8112

Column "N" and "#" indicate number of ports and number of instances tested in each group, and columns "R" and "CPU" represent the average number of rehandles and the average running time (in seconds). The "Total/Avg." row indicates the totals of the columns that contain absolute frequencies and the averages of the columns that contain average values. Rows with similar information are available for each of the five groups of instances.

running time of 2313 s by considering a time limit of one hour. It cuts the number of rehandles by 13% and 25% with time limits of two and four hours, respectively. Focusing on the number of ports, the number of rehandles is reduced by 9.7% with time limits of 1 h to 4 h in the instances with 6 ports. It is reduced by 17% with time limits of 1 to 4 h in the instances with 8 ports and by 30% in the instances with 10 ports. The results reveal a significant increase in the difficulty of the problem as the number of ports increases, dramatically increasing the average number of rehandles. The *IF* algorithm with a time limit of 4 h solves the instances with 6 ports in the Short, Mixed, Authentic, and Long groups with an average of 1.6 rehandles, the instances with 8 ports with an average of 15.7 rehandles, and those of 10 ports with an average of 40.9 rehandles. The average number of rehandles for the Required group is 131.3: an average of 49.8 rehandles for the 6-port instances, 122.0 rehandles for the 8-port instances, and 222.0 for the 10-port instances.

7. Conclusions and future research

We have studied a simplified container stowage planning problem with different container sizes, weights and stability constraints which, to the best of our knowledge, has never been solved without decomposition before. We provided integer programming formulations for this general problem and for three special cases: (i) identical container sizes and different container weights, (ii) identical container weights and different container sizes, and (iii) identical container weights and sizes. Through the computational study conducted, it can be seen that the model proposed for case (iii), that is for the basic container stowage planning problem, outperforms the state-of-the-art models. Furthermore, we observe that the difficulty of solving the problems increases considerably as containers of different sizes and stability constraints are included. In fact, when solved via a commercial solver, the integer programming model only obtained 2 feasible solutions in the 81 instances of 10 ports and 5 in the 81 instances of 8 ports within the time limit of one hour. Although the formulation did not perform well as a whole, failing to provide a solution in 82% of the instances, exploiting its decomposable structure led to the *IF* algorithm, which solves each of the instances tested. *IF* is an Insert-and-Fix matheuristic combined with constructive algorithms providing the model solved at each iteration with an initial solution. We observed that the average number of rehandles increases sharply with the number of ports. The *IF* algorithm solves all the instances of 6 and 8 ports in the Short, Mixed, Authentic, and Long groups with an average of 8.7 rehandles; however, the average of rehandles is 40.9 for the 10-port instances in these groups.

Future research directions will focus on including other characteristics of the real problems as well as on developing new modeling techniques and other metaheuristic approaches that will not rely on the performance of integer programming solvers.

CRediT authorship contribution statement

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

Acknowledgments

We would like to thank the anonymous referees for their comments which have helped to significantly improve this paper. This work has been partially supported by the Spanish Ministry of Science, Innovation, and Universities, FPU Grant A-2015-12849 and under the

project “OPTEP-Port Terminal Operations Optimization” (No. RTI2018-094940-B-I00) financed with FEDER, Spain funds. The second author acknowledges the partial support by Data-driven logistics, Spain (FWO-S007318N) and Internal Funds KU Leuven, Spain.

References

Ambrosino, D., Paolucci, M., Sciomachen, A., 2015a. Experimental evaluation of mixed integer programming models for the multi-port master bay plan problem. Flexible Serv. Manuf. J. 27 (2–3), 263–284.

Ambrosino, D., Paolucci, M., Sciomachen, A., 2015b. A MIP heuristic for multi port stowage planning. Transp. Res. Proc. 10, 725–734.

Ambrosino, D., Paolucci, M., Sciomachen, A., 2017. Computational evaluation of a MIP model for multi-port stowage planning problems. Soft Comput. 21 (7), 1753–1763.

de Araujo, S., Arenales, M., Clark, A., 2008. Lot sizing and furnace scheduling in small foundries. Comput. Oper. Res. 35, 916–932.

Avriel, M., Penn, M., 1993. Exact and approximate solutions of the container ship stowage problem. Comput. Ind. Eng. 25 (1–4), 271–274.

Avriel, M., Penn, M., Shpirer, N., 2000. Container ship stowage problem: complexity and connection to the coloring of circle graphs. Discrete Appl. Math. 103 (1–3), 271–279.

Avriel, M., Penn, M., Shpirer, N., Witteboon, S., 1998. Stowage planning for container ships to reduce the number of shifts. Ann. Oper. Res. 76, 55–71.

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

Botter, R., Brinati, M., 1992. Stowage container planning: A model for getting an optimal solution. In: Vieria, C., Martins, P., C., K. (Eds.), Proc. IFIP TC5/WG5.6 Seventh International Conference on Computer Applications in the Automation of Shipyard Operations and Ship Design, VII. North Holland, pp. 217–229.

Ding, D., Chou, M.C., 2015. Stowage planning for container ships: A heuristic algorithm to reduce the number of shifts. European J. Oper. Res. 246 (1), 242–249.

Dubrovsky, O., Levitin, G., Penn, M., 2002. A genetic algorithm with a compact solution encoding for the container ship stowage problem. J. Heuristics 8 (6), 585–599.

Jensen, R., Pacino, D., Ajspur, M., Vesterdal, C., 2018. Container Vessel Stowage Planning. Weilbach.

Kang, J.-G., Kim, Y.-D., 2002. Stowage planning in maritime container transportation. J. Oper. Res. Soc. 53 (4), 415–426.

Kessel, O., 1977. Planning the Loading and Unloading Procedures for Containerized Cargoships: An OR Case-Study. Technical Report, IMSOR, Technical University of Denmark.

Larsen, R., Pacino, D., 2020. A heuristic and a benchmark for the stowage planning problem. Marit. Econ. Logist. 23 (1), 94–122.

Lee, Y., Lee, K., 2020. Lot-sizing and scheduling in flat-panel display manufacturing process. Omega 93, 102036.

Li, J., Zhang, Y., Ma, J., Ji, S., 2018. Multi-port stowage planning for inland container liner shipping considering weight uncertainties. IEEE Access 6, 66468–66480.

Pacino, D., Delgado, A., Jensen, R.M., Bebbington, T., 2011. Fast generation of near-optimal plans for eco-efficient stowage of large container vessels. In: International Conference on Computational Logistics. Springer, pp. 286–301.

Paquay, C., Limbourg, S., Schyns, M., Oliveira, J., 2018. MIP-based constructive heuristics for the three-dimensional Bin Packing Problem with transportation constraints. Int. J. Prod. Res. 56, 1581–1592.

Parreño-Torres, C., Alvarez-Valdes, R., Parreño, F., 2019. Solution strategies for a multiport container ship stowage problem. Math. Probl. Eng. Art. ID 9029267.

Roberti, R., Pacino, D., 2018. A decomposition method for finding optimal container stowage plans. Transp. Sci. 52 (6), 1444–1462.

Toso, E., Morabito, R., Clark, A., 2009. Lot sizing and sequencing optimisation at an animal-feed plant. Comput. Ind. Eng. 57, 813–821.

UNCTAD, 2019. Review of maritime transport 2019. In: United Nations Conference on Trade and Development. URL https://unctad.org/en/PublicationsLibrary/rmt2019_en.pdf.

Wei, W., Guimaraes, L., Amorim, P., Almada-Lobo, B., 2017. Tactical production and distribution planning with dependency issues on the production process. Omega 67, 99–114.

Wilson, I.D., Roach, P.A., 1999. Principles of combinatorial optimization applied to container-ship stowage planning. J. Heuristics 5 (4), 403–418.

Wolsey, L., 1998. Integer Programming. Wiley.

Zhang, E., Mei, Q., Liu, M., Zheng, F., 2018. Stowage planning in multiple ports with shifting fee minimization. Sci. Program. Art. ID 3450726.

Zhang, G., Nishi, T., Turner, S., Oga, K., Li, X., 2017. An integrated strategy for a production planning and warehouse layout problem: Modeling and solution approaches. Omega 68, 85–94.