Transportation Research Part E 94 (2016) 141–157

Contents lists available at ScienceDirect

Transportation Research Part E

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

A GRASP algorithm for the container stowage slot planning problem

Francisco Parreño [a,*], Dario Pacino [b], Ramon Alvarez-Valdes [c]

[a] University of Castilla-La Mancha, Department of Mathematics, Albacete, Spain
[b] Technical University of Denmark, Department of Management Engineering, Lyngby, Denmark
[c] University of Valencia, Department of Statistics and Operations Research, Burjassot, Valencia, Spain

A R T I C L E I N F O

Article history:
Received 10 December 2015
Received in revised form 20 July 2016
Accepted 23 July 2016
Available online 18 August 2016

Keywords:
Container vessel stowage planning
Slot planning
Integer programming
Heuristic algorithms
GRASP

A B S T R A C T

This work presents a generalization of the Slot Planning Problem which raises when the liner shipping industry needs to plan the placement of containers within a vessel (stowage planning). State-of-the-art stowage planning relies on a heuristic decomposition where containers are first distributed in clusters along the vessel. For each of those clusters a specific position for each container must be found. Compared to previous studies, we have introduced two new features: the explicit handling of rolled out containers and the inclusion of separations rules for dangerous cargo. We present a novel integer programming formulation and a Greedy Randomized Adaptive Search Procedure (GRASP) to solve the problem. The approach is able to find high-quality solution within 1 s. We also provide comparison with the state-of-the-art on an existing and a new set of benchmark instances.

© 2016 Elsevier Ltd. All rights reserved.

1. Introduction

Over the past two decades there has been a continuous increase in demand for cost efficient containerized transportation. In order to meet this demand, shipping companies have deployed larger container vessels, which can nowadays transport up to 19,000 TEUs (Twenty-Foot Container Equivalent Units). These vessels sail from port to port loading and unloading thousands of containers. Minimizing the time a vessel spends in port involves, among other issues, an efficient stowage plan; a plan that describes where each container should be loaded on the vessel.

Container vessels are ships specially designed for the transportation of large amounts of containers with a small crew. Containers are metal boxes designed to withstand significant outer forces. They are particularly robust to high vertical compression, which makes it possible to create high stacks. All containers are fitted with corner castings designed to support the container's weight, and to which security fittings can be attached. ISO standard containers are usually 20′, 40′, or 45′ long. In the US trade it is also possible to find 48′ or 53′ containers (although they are not standard in liner shipping). ISO containers are 8′ wide and 8′6″ high, with the exception of high-cube containers, which are 1 foot taller. High-cube 20′ foot containers are rare and we assume they do not exist when modeling the slot planning problem. Longer containers, such as 45′ containers, are equipped with two extra sets of castings at a 40′ distance. The extra castings allow the longer containers to be stacked on top of 40′ containers. No castings, however, exist at the 20′ position, which means that 20′ containers cannot be stacked on top of longer containers.

Aside from the standard and high-cube containers, there are a number of specialized containers for different kinds of cargo. Fruits and vegetables, for example, must be transported in refrigerated containers called reefers. Liquids can be

* Corresponding author.
E-mail address: francisco.parreno@uclm.es (F. Parreño).

http://dx.doi.org/10.1016/j.tre.2016.07.011
1366-5545/© 2016 Elsevier Ltd. All rights reserved.

transported in tank containers, while break bulk cargo is transported on platforms and/or open-top containers (these, however, constitute only a very small percentage of the cargo). Containers transporting dangerous goods are called IMO containers. Depending on the nature of the cargo, a special IMO code is assigned to the container. Strict separation rules apply to IMO containers (Storck, 2015).

The layout of a container vessel is shown in Fig. 1. The figure shows how containers are arranged in storage areas called bays, along the entire length of the vessel. A bay is composed of a number of cells, each indicating a possible stowage position. Cells usually have a capacity of two TEUs (Twenty-Foot Equivalent Units), meaning that they can either stow two 20′ containers or one 40′ (or 45′) container. Each TEU position within a cell is referred to as a slot. Slots toward the bow of the vessel are called Fore slots and those towards the stern are called Aft slots. Cells are identified by a stack number, indicating its horizontal position within a bay, and a tier number indicating its vertical position. In general there is a distinction between on-deck and below-deck areas of a bay. The below-deck areas are closed by hatch-covers (or hatch-lids), which are tight metallic structures that prevent water from entering. Note that often 45′ containers are only allowed on-deck, and that the below-deck part of a bay is sometimes also called the hold. Fig. 1 also shows how only a subset of the cells have access to electric power (or reefer plugs).

A feasible stowage plan has to satisfy high-level constraints ensuring that the vessel is stable and seaworthy, and low-level constraints concerning the way in which each container is loaded into a position on the vessel. Using this constraint classification, state-of-the-art stowage planning typically follows a 2-phase hierarchical decomposition of the problem (Pacino et al., 2011; Ambrosino et al., 2015; Kang and Kim, 2002; Wilson and Roach, 2000). In the first phase, called Master Planning, containers are distributed to subsections of the vessel called locations. The distribution of containers must satisfy seaworthiness requirements. The draft of the vessel (the immersion depth) must be within limits; e.g. to ensure that the propeller is under water or that the vessel does not run aground when at port. The center of gravity must also be controlled in order for the vessel to be stable and to adjust the trim (the difference between the draft at stern and bow). The uneven cargo distribution and the shape of the vessel's hull result in different opposing forces that stress the structure of the vessel. Example of those are shear and bending moments which must also be within tolerance. One of the main objectives of the Master Planning phase is to minimize hatch-overstowage, meaning the number of containers on deck that need to be removed to get access to containers placed below deck. Since this is not the focus of this study, we refer the reader to Pacino et al. (2011) for a detailed description. The second phase of the decomposition, Slot Planning, refines the container distribution and identifies the exact position of each container in each vessel location. This phase is concerned with low-level constraints governing the physical position of the containers, ensuring that weight and height capacities are satisfied, and that reefers (refrigerated containers) are assigned to positions where a power outlet is available.

Fig. 2 depicts the decomposition approach. Master Planning requires a loadlist and port and vessel data. The loadlist includes the containers to be loaded at the current port and a forecast of those to be loaded at later ports. Vessel and port data include information about the layout of the ship and the depth of the ports of call. As its output is a class-based stowage plan (where only container types are taken into account), it is possible to solve an independent Slot Planning Problem for each location on the vessel. Though this might seem to be a simplification of the problem, the reason for following this path is rooted in the container terminal part of the optimization. Given a class-based stowage plan, terminals can optimize the load sequencing of the containers and thus further reduce the ship's time in port (Monaco et al., 2014).

Since the Master Planning does not take stacking constraints into consideration, there is the possibility that the resulting Slot Planning problems are not feasible. Pacino et al. (2011) handle this issue with a post-optimization procedure whereby containers are removed from the solutions until a feasible slot plan is reached. The authors also show that the number of rolled out containers is so low that more iterations of the decomposition are not needed.

Since solving the stowage planning problem over multiple ports requires the use of forecasts, stowage planners wish to be able to analyze different forecast scenarios. According to Pacino et al. (2011) and Pacino and Jensen (2012) 10 min is the maximum time the industry deems acceptable for stowage planning software to complete computations, which in turn leaves an estimated time of 1 s to solve each Slot Planning Problem.

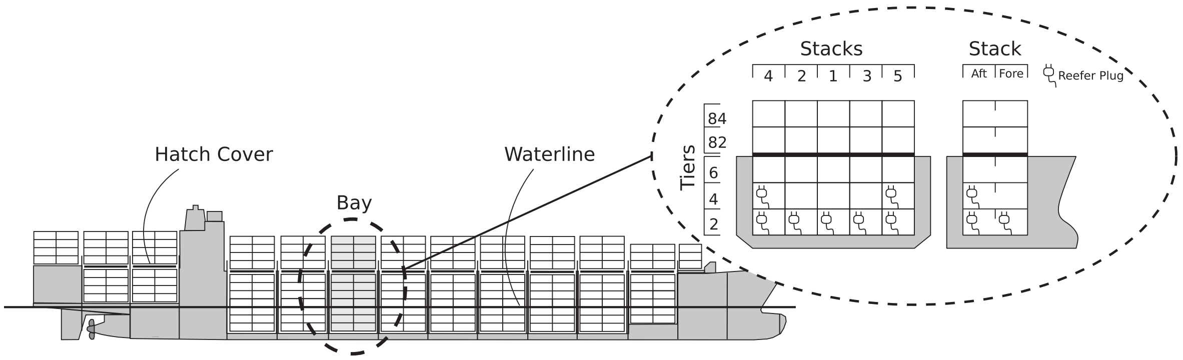

Fig. 1. The layout of a container vessel.

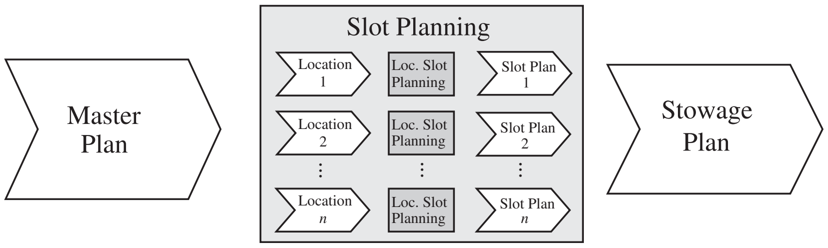

Fig. 2. The master planning and slot planning decomposition of container ship stowage planning.

This paper focuses on the Slot Planning Problem, and presents several contributions:

- It proposes a generalization of the Slot Planning Problem that addresses the feasibility issue in the decomposition proposed in Pacino et al. (2011). We address this issue by incorporating maximum container intake in the objectives of the problem, thus improving the solutions otherwise obtained by greedily removing constraint violating containers.
- The Slot Planning problem is further extended with the inclusion of IMO containers (dangerous goods). A representative set of the rules is modeled as a demonstration of how the complete set could be handled.
- We propose a novel Integer Programming (IP) formulation that not only models the generalized problem, but also has a much tighter bound than the state-of-the-art formulation of Delgado et al. (2012).
- Since even our improved mathematical formulation is not able to solve the problem within the required time limit, we also propose a novel, high-performing, GRASP approach including two constructive procedures, two randomizing strategies, and several improvement moves. The results show that the new algorithm obtains high-quality solutions in very short running times of 1 CPU second per location, outperforming previously published procedures.

The remainder of the paper is structured as follows. A review of relevant literature is presented in Section 2. In Section 3 we define the Slot Planning Problem and present an integer programming formulation. Our proposed solution approach is presented by describing first the construction heuristics in Section 4, followed by the GRASP algorithm, detailed in Section 5. Computational results are presented in Section 6 and conclusions and directions for future work are discussed in Section 7.

2. Literature review

A growing number of studies on stowage planning have been published in the past few years. The contributions can roughly be classified as belonging to multi-phase or single-phase approaches. As we have commented before, multi-phase approaches (Wilson and Roach, 2000; Kang and Kim, 2002; Pacino et al., 2011; Ambrosino et al., 2010, 2015) are currently the best performing solutions, where the stowage planning problem is hierarchically divided into sub-problems, often into master planning and slot planning. The master planning phase focuses on satisfying vessel stability in an aggregated container distribution. In Pacino et al. (2011), for example, master planning decides the number of containers of a specific type to be allocated to a location (a logical subsection of the vessel). This is done taking into account hatch-overstowage and loading crane performance. The distribution of containers is then refined on a per location basis, where stacking constraints are taken into account. Recently a new heuristic for a master planning phase based on a partial variable fixing procedure has been presented in Ambrosino et al. (2015).

Single-phase approaches consider the stowage planning problem as a whole. Since most of these works involve only a very simplified representation of ship stability, they can also be seen as slot planning approaches. Solution methods include mathematical programming (Aslidis, 1984; Botter and Brinati, 1992; Ambrosino et al., 2004; Li et al., 2008), constraint programming (CP) (Ambrosino and Sciomachen, 1998; Delgado et al., 2012), constraint-based local search (Pacino and Jensen, 2012), genetic algorithms (Davidor, 1998; Dubrovsky et al., 2002), dedicated heuristics (Avriel et al., 1998; Ding and Chou, 2015), 3D-packing (Sciomachen and Tanfani, 2003), simulation (Aye et al., 2010), and case-based methods (Nugroho, 2004). Of those that are relevant for the Slot Planning Problem, in Avriel et al. (1998), Ambrosino and Sciomachen (1998), and Dubrovsky et al. (2002) all containers are considered to be of the same size and no distinction is made for reefer and high-cube containers. Approaches that include both 20′ and 40′ containers are Aslidis (1984), Botter and Brinati (1992), Sciomachen and Tanfani (2003), and Ambrosino et al. (2004). Of these, Sciomachen and Tanfani (2003) also include high-cubes, while Ambrosino et al. (2004) include reefers.

To the best of the authors' knowledge, the only two papers that propose a Slot Planning problem comparable to our definition are Delgado et al. (2012) and Pacino and Jensen (2012). In Delgado et al. (2012) a constraint programming model is able to find optimal solutions in very short computational times, and the results are shown to outperform those of a mathematical formulation. The model includes custom built propagators and partial solution evaluators to improve the branch & bound search. Propagators are algorithms repetitively called during the search that help to remove infeasible values from the domain of each variable, while partial solution evaluators are lower bounds calculated on partial variable assignments that can help to cut branches of the search tree. The model, moreover, uses channeling constraints which are binary relations used to merge two different models: in this specific case, one model where the variables decide which container to assign to a slot and another in which variables decide which slot to assign to a container. If a propagator removes infeasible values from the domain of a variable in one model, the channeling constraint removes the corresponding value in the other model.

The approach proposed in Delgado et al. (2012), however, is not robust and fails to find solutions in a number of instances. The application of this model to our generalization of the problem is not trivial. IMO containers could easily be incorporated using an automaton-based propagator; however, modeling the intake maximization would break the channeling constraint, which assumes that all containers must be loaded. This would involve a significant change in the modeling of the problem. The constraint-based local search approach in Pacino and Jensen (2012) combines a constructive algorithm with a local search guided by constraint violation. The procedure starts from a possibly infeasible solution. Each constraint calculates its degree of violation, which the procedure aims to minimize. This is done using a swap operator between containers. Objectives are also modeled as constraints and are then minimized in the same way. Even though the procedure is very fast and often finds near-optimal solutions, it still fails to solve all the tested instances. Moreover, this approach requires the implementation of a delta evaluation and a delta assignment procedure for each of the implemented constraints and search operators. In Pacino et al. (2011) the constraint programming described and the constraint-based local search approach are combined to solve the slot planning problem. For the instances where no solution is found, constraint-violating containers are removed until a feasible configuration is reached. The diving heuristic of the constraint programming approach and the construction heuristic of the constraint-based local search seem to be the strengths of the two solution methods. This motivates our hypothesis that a GRASP-based approach will be able to achieve good results even in a generalized version of the slot planning problem.

3. Problem definition and mathematical formulation

As previously mentioned, Slot Planning assumes that a solution to Master Planning is given. We are thus provided with a set of containers (or container types) to be loaded into a single vessel location. A representative Slot Planning problem was presented in Delgado et al. (2012) and Pacino and Jensen (2012). We extend this definition by including IMO restrictions and maximization of the loaded cargo. Thus, given a set of containers[1] to load into a location, a feasible solution must satisfy the following rules:

(a) Assigned cells must form stacks (containers stand on top of each other in the stacks; they cannot hang in the air).
(b) 20′ containers cannot be stacked on top of 40′ containers.
(c) A 20′ reefer container must be placed in a reefer slot. A 40′ reefer container must be placed in a cell with at least one reefer slot.
(d) The length constraint of a cell must be satisfied (some cells hold only 40′ or only 20′ containers).
(e) The sum of the heights and the sum of the weights of the containers stowed in a stack must be within the stack limits.
(f) All containers already on board must be stowed in their original position and they cannot be moved.
(g) A cell must be either empty or have both slots occupied (no odd slots).
(h) IMO rules must be satisfied.

IMO rules dictate how far apart incompatible cargoes must be stowed (Storck, 2015). Since in Slot Planning we only focus on a location, it is sufficient to concentrate only on rules that enforce stack segregation. Segregation that goes beyond the stack level must be handled at the Master Planning stage. Without loss of generality, we have devised a representative set of rules based on 4 IMO categories. Table 1 shows these rules. The first column represents the IMO category of dangerous goods. For each of those categories, the second column shows the type of cargo that can be stowed in proximity, and the third, cargo that must be kept at least one stack away. Containers with no IMO categorization (or IMO 0) are assumed to be compatible with any other cargo.

According to stowage coordinators, an optimal Slot Plan has to minimize the weighted sum of the following objectives:

(a) Minimize the number of containers left out of the solution. Sometimes is not possible to stow all the containers in the location. A unit cost, α_1, is paid for each container left out of the solution.
(b) Minimize overstows, which are movements of containers that allow the unloading of other container placed below. A unit cost, α_2, is paid for each container overstowing any containers below.
(c) Avoid stacks with containers to be discharged at many different ports. A unit cost, α_3, is paid for each discharge port included in a stack.
(d) Keep stacks empty if possible. A unit cost, α_4, is paid for each stack used.
(e) Avoid loading non-reefer containers into reefer slots. A unit cost, α_5, is paid for each reefer slot occupied by a non-reefer container.

[1] Which can be 20′, 40′, high-cube, reefers and IMO.

Table 1
Representative cargo segregation rules.

IMO ID	Free	Stack segregation
1	1, 4	2, 3
2	2	1, 3, 4
3	3, 4	1, 2
4	1, 3, 4	2

The cost structure is designed to reflect the fact that the main objective, by far, is to load all the containers assigned to a location. The secondary objectives allow us to distinguish between solutions that load the same number of containers. Among the secondary objectives, the most important is the minimization of unproductive moves. The weighting of the cost components follows the suggestion by Delgado et al. (2012) (see Section 6.2 for the actual values).

Delgado et al. (2012) developed an integer linear formulation for the slot planning problem. Using it as a basis, we have developed an improved model, reducing the number of constraints and binary variables, maximizing the number of containers loaded instead of imposing the condition of loading all of them, which could be impossible in some cases, and adding constraints to deal with IMO containers. Let us now introduce the mathematical notation:

Sets

I	Index set of containers
T	Index set of 20′ containers
F	Index set of 40′ containers
L	Set of containers already on-board
D	Index set of discharge ports
M	Index set of IMO categories
IMO	Set of category pairs that need segregation
J	Index set of stacks
K_j	Index set of cells in stack $j \in J$

Constants

H_i^c	Height of container $i \in I$
W_i^c	Weight of container $i \in I$
R_i^c	Indicates whether container $i \in I$ is reefer or not
M_i	IMO category of container $i \in I$.
A_{id}	Indicates whether container $i \in I$ is unloaded at port $d \in D$ or not
R_{jk}	Number of reefer plugs in cell $k \in K_j$ of stack j (0, 1, 2)
W_j^s	Weight limit of stack $j \in J$
H_j^s	Height limit of stack $j \in J$
α_f	Weight in the objective function of cost criterion $f \in \{1, \dots, 5\}$

Variables

$$o_{jk} = \begin{cases} 0 & \text{if no containers in stack } j \in J, \text{ cell } k \in K_j, \text{ overstows other container below in the stack} \\ 1 & \text{if one 40}' \text{ or one 20}' \text{ container, occupying cell } k \in K_j \text{ of stack } j \in J, \text{ overstows other container} \\ 2 & \text{if two 20}' \text{ containers, occupying cell } k \in K_j \text{ of stack } j \in J, \text{ overstow other container} \end{cases}$$

$$c_{jki} = \begin{cases} 1 & \text{if container } i \in I \text{ is stowed in cell } k \in K_j \text{ of stack } j \in J \\ 0 & \text{otherwise} \end{cases}$$

$$e_j = \begin{cases} 1 & \text{if stack } j \in J \text{ is used} \\ 0 & \text{otherwise} \end{cases}$$

$$p_{jd} = \begin{cases} 1 & \text{if at least one container in stack } j \in J \text{ is to be unloaded at port } d \in D \\ 0 & \text{otherwise} \end{cases}$$

$$u_{jkd} = \begin{cases} 1 & \text{if a container below cell } k \in K_j \text{ of stack } j \in J \text{ is to be unloaded before port } d \in D \\ 0 & \text{otherwise} \end{cases}$$

$$v_{jm} = \begin{cases} 1 & \text{if at least one container with IMO category } m \in M \text{ is stowed in stack } j \in J \\ 0 & \text{otherwise} \end{cases}$$

Using these variables and parameters, the model is:

$$\text{Min } \alpha_1\left(|I| - \sum_{i\in I}\sum_{j\in J}\sum_{k\in K_j} c_{jki}\right) + \alpha_2\sum_{j\in J}\sum_{k\in K_j} o_{jk} + \alpha_3\sum_{j\in J}\sum_{d\in D} p_{jd} + \alpha_4\sum_{j\in J} e_j + \alpha_5\sum_{j\in J}\sum_{k\in K_j}\left(R_{jk}\sum_{i\in F} c_{jki}(1-R_i^c) + \sum_{i\in T} c_{jki}\left(\frac{1}{2}R_{jk} - R_i^c\right)\right) \tag{1}$$

s.t.

$$\frac{1}{2}\sum_{i\in T} c_{j(k-1)i} + \sum_{i\in F} c_{j(k-1)i} - \sum_{i\in F} c_{jki} \geq 0 \quad \forall j\in J, k\in K_j \setminus \{1\} \tag{2}$$

$$\sum_{i\in T} c_{jki} - \sum_{i\in T} c_{j(k-1)i} \leqslant 0 \quad \forall j\in J,\ k\in K_j \setminus \{1\} \tag{3}$$

$$\frac{1}{2}\sum_{i\in T} c_{jki} + \sum_{i\in F} c_{jki} \leqslant 1 \quad \forall j\in J,\ k\in K_j \tag{4}$$

$$\sum_{j\in J}\sum_{k\in K_j} c_{jki} \leqslant 1 \quad \forall i\in I \tag{5}$$

$$\sum_{i'\in T} c_{jki'} - 2c_{jki} \geqslant 0 \quad \forall j\in J,\ k\in K_j,\ i\in T \tag{6}$$

$$\sum_{i\in I} R_i^c c_{jki} - R_{jk} \leqslant 0 \quad \forall j\in J,\ k\in K_j \tag{7}$$

$$\sum_{k\in K_j}\sum_{i\in I} W_i^c c_{jki} \leqslant W_j^s \quad \forall j\in J \tag{8}$$

$$\sum_{k\in K_j}\left(\frac{1}{2}\sum_{i\in T} H_i^c c_{jki} + \sum_{i\in F} H_i^c c_{jki}\right) \leqslant H_j^s \quad \forall j\in J \tag{9}$$

$$\sum_{d'=1}^{d-1}\left(\sum_{i\in F} A_{id'} c_{jk'i} + \frac{1}{2}\sum_{i\in T} A_{id'} c_{jk'i}\right) \leqslant u_{jkd} \quad \forall j\in J,\ k\in K_j,\ k>1,\ k'\in K,\ k'<k\ d\in D,\ d>1,\ d'\in D,\ d'<d \tag{10}$$

$$\sum_{i\in F} A_{id} c_{jki} + u_{jkd} \leqslant 1 + o_{jk} \quad \forall j\in J,\ k\in K_j,\ d\in D \tag{11}$$

$$\frac{1}{2}\sum_{i\in T} A_{id} c_{jki} + u_{jkd} \leqslant 1 + \frac{1}{2}o_{jk} \quad \forall j\in J,\ k\in K_j,\ d\in D \tag{12}$$

$$|K_j|e_j - \left(\sum_{i\in F} c_{j1i} + \frac{1}{2}\sum_{i\in T} c_{j1i}\right) \geqslant 0 \quad \forall j\in J \tag{13}$$

$$|K_j|p_{jd} - \sum_{k\in K_j}\left(\sum_{i\in F} A_{id} c_{jki} + \frac{1}{2}\sum_{i\in T} A_{id} c_{jki}\right) \geqslant 0 \quad \forall j\in J,\ d\in D \tag{14}$$

$$c_{jki} = 1 \quad \forall (j,k,i)\in L \tag{15}$$

$$|K_j|\, v_{jm} - \sum_{k\in K_j}\left(\sum_{i\in F, M_i=m} c_{jki} + \frac{1}{2}\sum_{i\in T, M_i=m} c_{jki}\right) \geqslant 0 \quad \forall m\in M,\ j\in J \tag{16}$$

$$v_{jm_1} + v_{j+1m_2} \leqslant 1 \quad \forall j \leqslant |J|-1,\ (m_1,m_2)\in IMO \tag{17}$$

$$v_{jm_1} + v_{jm_2} \leqslant 1 \quad \forall j\in J,\ (m_1,m_2)\in IMO \tag{18}$$

The objective function is the sum of the cost criteria defined above, weighted by parameters $\alpha_f, f \in \{1, \ldots, 5\}$. Constraints (2) ensure that there are either two 20′ or one 40′ container below a cell stowing a 40′ container, to avoid 40′ containers hanging in the air. Similarly, (3) force that there must be 20′ containers below a cell stowing 20′ containers. Cell capacity is handled in (4), which requires that a cell stows at most two 20′ containers or one 40′ container. Constraints (5) impose that each container can occupy at most one cell. Moreover, constraints (6) force the number of 20′ containers in a cell to be 0 or 2, since a cell must be empty or with both slots occupied. With constraints (7) we ensure that reefer containers are only stowed in cells with reefer plugs. The weight and height limits of the stacks are enforced by constraints (8) and (9) respectively. Note that as stated in Section 1, all 20′ containers are assumed to have the same height since we do not consider 20′ high-cubes. Constraints (10) ensure that variables u_{jkd} take value 1 if in cell k', below cell k of stack j, there is a container that has to be unloaded at d', a port earlier than d in the vessel route. Variables u_{jkd} are then used to assign the values of the overstowage variables o_{jk} for each cell. Constraints (11) refer to 40′ containers. If there is a 40′ container in cell k of stack j being discharged at port d and $u_{jkd} = 1$, this container overstows another container placed below and thus $o_{jk} = 1$. If any of these two conditions do not hold, there is no overstowage and $o_{jk} = 0$. Constraints (12) do the same for 20′ containers. If cell k of stack j is occupied by two 20′ containers and $u_{jkd} = 1$, the number of these 20′ containers being discharged at port d determines the number of overstows, which can be 0, 1, or 2. The binary variables e_j must be set to 1 if there at least is one container stowed in stack j. This is done in (13) with a $Big - M$ constraint where the value of M corresponds to the cardinality of the set K_j. In a similarly fashion, constraints (14) set the binary variables p_{jd} to 1 if there at least is one container in stack j being unloaded at port d. Constraints (15) ensure that already loaded containers retain their position. The set L of already loaded containers is composed of the triples (i, j, k) indicating that container i must be stowed in stack j and cell k. Constraints (16)–(18) enforce the IMO segregation rules. First, constraints (16) set the binary variable v_{jm} to 1 if there is at least one container in stack j belonging to IMO category m. Then, constraints (17) and (18) use variables v_{jm} to enforce the segregation rules.

4. Constructive algorithm

We are given the list of containers assigned to the location by the solution of the master planning problem as well as the list of containers already stowed in this location with their positions. Following an iterative process, we combine two elements: a list *Cont* of the containers still to be stowed, initially the complete list of containers, and a list S of stacks. Each stack $s \in S$ is then composed of a list C_s of cells in which it is possible to stow a container.

At each step, we first choose a stack from S and consider the first cell available in this stack, that is, the lowest non-occupied cell. Then we choose the container to be stowed from among the containers in *Cont* which could be feasibly stowed in it. The process goes on until all the containers have been stowed or there is no usable space left.

Step 0: Initialization
We have the list *Cont* ordered by port of discharge, length, reefer, and height. With respect to the port, containers appear in decreasing order, so that those going to ports visited later are stowed first. With respect to length, they are in increasing order: we want to stow first 20′ containers and then 40′ containers. Reefer and high cube containers appear before standard containers in the ordered list *Cont* because we want to consider them first when choosing the container to stow. In list S the partially filled stacks appear first and then the empty stacks. Both sublists are ordered separately by their number of available cells in decreasing order. This ordering seeks to minimize the number of stacks being used.
Step 1: Choosing the stack and cell
We choose the first stack in list S and its lowest empty cell. We do not change the stack until it has no more empty cells or no more containers can be stowed in it.
Step 2: Choosing the container
Once a cell has been chosen, we know the characteristics of the cell and we have information about other containers previously stowed in this and other adjacent stacks.
A container can be stowed in the selected position if it satisfies a series of conditions:

- *Type of container*
  Some cells hold only 40′ or only 20′ containers. If the selected cell has one of these restrictions, the container to be stowed must satisfy it.
  If the tier below contains a 40′ container, 20′ containers cannot be stowed in the selected cell.
  If a 20′ container is considered for stowage in one of the slots of the cell, we must check that there is another 20′ container that can occupy the other slot, because both slots of the cell must simultaneously be empty or occupied.
- *Power supply*
  According to the values of R_{jk}, the number of plugs in the cell, we can have the following cases:
  - If $R_{jk} = 0$ (no plugs in the cell), reefer containers cannot be stowed in the cell.
  - If $R_{jk} = 1$ (one plug in one side of the cell), one 40′ reefer container or a 20′ reefer container can be stowed. In the second case, the other 20′ container completing the cell must be non-reefer. If there are no reefer containers to be stowed in the selected cell, non-reefer containers can be stowed with a penalty.

  - If $R_{jk} = 2$ (plugs in both sides of the cell), one reefer 40′ container or a reefer 20′ container can be stowed. In the second case, the other 20′ container completing the cell should also be reefer, although as in the previous case non-reefer containers can also be stowed.
- *Weight*
  The weight of the containers already in the stack plus the weight of the new container(s) must not exceed the total weight limit of the stack.
- *Height*
  The height of the containers already in the stack, plus the height of the new container must not exceed to the total height limit of the stack.
  High-cube containers are subject to an additional test concerning height. We compute the number of standard containers that could be stowed in the stack according to the remaining height. If putting a high-cube container reduces this number, it should not be stowed in this cell, because that would produce an empty space on top of the stack. For instance, if the stack has a remaining height of 27′, three standard containers (8′6″) could be stowed. Putting a high-cube container (9′6″) in this position will reduce the remaining height to 17′6″ and two standard containers could still be stowed in the stack. In this case, the high-cube container can be stowed in the cell. However, if the remaining height were 26′, the high-cube container would reduce the remaining height to 16′6″ and only one more container could be stowed. In this case, the high-cube container would not be stowed in this cell. Nevertheless, this rule only applies if there are other standard containers with the same port of discharge that could occupy the cell. If only high-cube containers for this port are still to be stowed, they can occupy the cell, because the priority assigned to the order of ports overrides a better use of stack height.
- *IMO constraints*
  If there are IMO containers in the stack or in adjacent stacks, the container to be stowed must not be incompatible with them.

When the selected cell is in a tier in the lower half of the stack, the *Cont* list, besides being ordered by port, length, reefer, and height, is also ordered by decreasing weight. Conversely, when the tier is in the upper half of the stack, the order is by increasing weight. We want to stow heavyweight containers in the lower half of each stack, and lightweight containers in the upper half. The idea is to maintain a similar average weight in each stack.

Step 3: Updating the lists

We remove the container (or containers) already stowed from the list and update the values of the total weight and height of the chosen stack. This process continues until there is no container on the list or there is no possible cell in which to stow any remaining container.

Tier-by-tier constructive algorithm

In the previous constructive algorithm, once a stack is open we stow all the containers that fit into the stack before considering a new stack on list *S*. We call this strategy a *sequential* construction. We have also studied an alternative in which instead of choosing a stack and filling it completely, at Step 1 we always choose the lowest empty cell irrespective of the stack to which it belongs. With this strategy the location is filled by tiers, starting from the bottom. We call this strategy a *tier-by-tier* construction. This alternative is not good for minimizing the number of stacks used because it keeps all the available stacks open, but it produces diverse solutions that are useful in the iterative procedure described later.

5. A GRASP algorithm

GRASP, Greedy Randomized Adaptive Search Procedure (Feo and Resende, 1995), is an iterative procedure that combines a constructive phase and an improvement phase. In the constructive phase the solution is built step by step, adding one element to a partial solution. For selecting the element to be added, a greedy strategy is used, but the selection is not deterministic because a random strategy is used to provide diversity. The improvement phase consists of a local search. GRASP has been successfully applied to many combinatorial optimization problems, such as vehicle routing (Prins et al., 2014), territory design (Rios-Mercado and Escalante, 2016), or inventory routing problems (Shao et al., 2015), including container loading problems closely related to the slot planning problem (Moura and Oliveira, 2005; Alonso et al., 2014).

The GRASP metaheuristic algorithm has been chosen because its structure is very appropriate for the characteristics of the problem being solved. As the results from Pacino and Jensen (2012) and Delgado et al. (2012) indicate, proper construction heuristics tend to produce high quality solutions for the Slot Planning problem. The randomization of the construction heuristic within an iterative approach then might be what is needed to escape from the local minima convergence that afflicts the CBLS approach of Pacino and Jensen (2012).

The approach, which is shown in detail in Algorithm 1, follows this general procedure. First a randomize construction algorithm finds a solution to the problem. Successively the solution is improved using a local search procedure, and accepted if it is better than the previous one. These two steps are iterated until a termination criteria is met. Details of each search component are described in the following sections.

5.1. Randomizing the constructive algorithm

We consider the constructive algorithm described in the previous section and randomize Step 2, the order of the list of stacks, and Step 3, the list of containers to be stowed.

In Step 2, we maintained two lists of stacks, first the stacks that are partially filled, and then the empty stacks. Now, instead of having the lists ordered by decreasing number of available cells, we assign to each stack a number taken at random from 0 to its number of available cells, and order each list according to these random numbers. Ties are broken at random.

In Step 3, we considered two ways of ordering the containers depending on the tier of the cell. Now, instead of considering the complete ordered list of containers, we take a random sample. The size of this sample is determined by a parameter δ that indicates the proportion of the containers to be sampled, so each container has a probability δ of being part of the sample. We go through the ordered sample and the first container that satisfies all the conditions is stowed in that cell. This technique was proposed by Resende and Werneck (2004), who called it *sample plus construction*. The diversity of the process is managed by the value of the parameter δ. If $\delta = 1$, all the containers are selected for the sample and the first one satisfying all the conditions is always chosen, making the selection deterministic. On the contrary, when δ is closer to 0, the number of selected candidates is smaller and the best containers have a lower probability of being selected. This means that the cell may be occupied by containers that are not so suitable according to their port, length or reefer property. In that sense, this procedure introduces more diversity into the set of solutions produced in the iterative procedure. With this method it is, however, possible that none of the selected containers satisfies all the conditions for occupying the cell. In that case, we go through the complete ordered list of containers and the first container that satisfy the conditions is stowed in that position.

In order to introduce even more diversity into the solutions, we use two more randomizing procedures. On the one hand, since we have developed two constructive methods, sequential and tier-by-tier, we randomly choose one of them at each iteration to build the solution. On the other hand, instead of always ordering the containers by decreasing weight in the lower half of the stack and increasing weight in the upper half, at each iteration we randomly choose between increasing and decreasing order for the weights.

Due to randomization we can have stacks with some overstowage, which can easily be removed by reordering the containers in the stack. Then, once a solution is built by the randomized algorithm we try to reduce this overstowage by reordering the containers stowed in each stack.

5.2. Determining the parameter δ

A preliminary computational experiment showed that no value of δ always produced the best results. Therefore we decided to use a Reactive GRASP strategy, proposed by Prais and Ribeiro (2000), in which at each iteration δ is taken at random from a set of discrete values $\{0.1, \ldots, 0.9\}$. Initially, all these values have the same probability of being chosen. In the iterative process we keep the value of the solutions obtained with each value of δ. After a certain number of iterations the probabilities are modified. Those corresponding to values of δ that have produced high-quality solutions are increased and, conversely, the probabilities of the values producing low-quality solutions are decreased. The procedure is similar to that of Delorme et al. (2004).

As we have two different methods (sequential and tier-by-tier) for building the solution, it is possible that a given value of δ is good for the sequential but not for the tier-by-tier procedure or vice versa, so we keep two lists of values for δ, one for the sequential case and another for the tier-by-tier.

5.3. Improvement phase

In this section we try to improve the solutions obtained by the randomized constructive algorithm by using local search. We have defined eight improvement moves, based on two basic strategies. The first strategy consists of removing some of the containers in the solution and then filling the location again using a deterministic procedure. Six different improvement moves have been devised from this strategy (also called ruin-and-recreate), which are described in Section 5.3.1. The second strategy, from which the two remaining moves are created, consists of exchanging containers between stacks and is described in Section 5.3.2. The eight improvement moves are applied to each solution that passes the filter described in Section 5.3.3. If some of these moves produce improvements, we take the one with the best improvement as the new solution.

5.3.1. Removing and filling moves

The first set of three improvement moves is based on the observation that wrong decisions might be taken by the randomized constructive process, resulting in bad solutions. Let γ be a parameter determining how much of the cur-

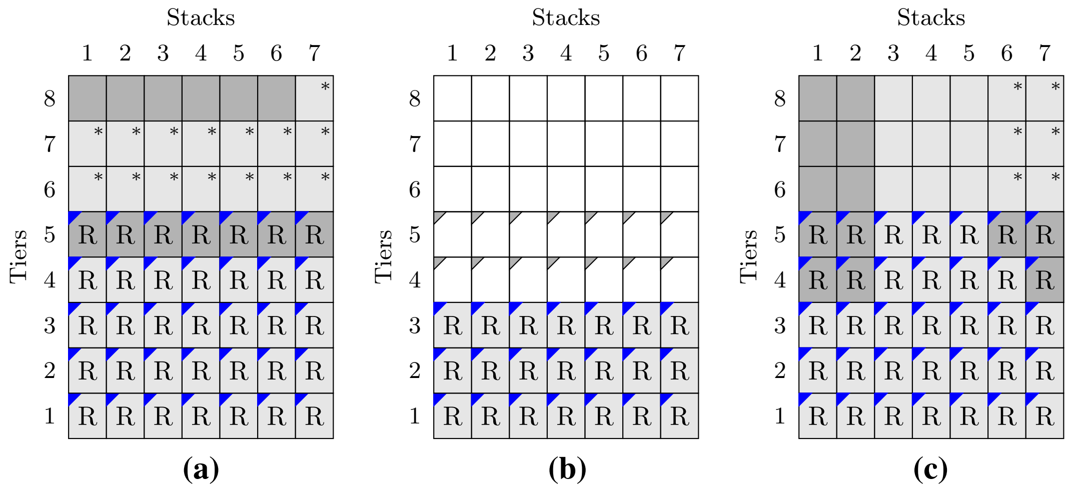

Fig. 3. Improving solutions. Removing last containers stowed.

rent solution should be kept. Since we want to relax the late decisions, we start removing containers starting from the last inserted. We do so until only $\gamma\%$ of the original solution is left. The containers are then reinserted using the sequential deterministic procedure. Values of γ are chosen at random in the interval $(35, 95)$. The values of the interval are set so as to remove a significant part of the solution without starting the constructive process from scratch. They are similar to those used in previous studies on related problems (Alonso et al., 2014; Alvarez-Valdes et al., 2013).

We use the sequential constructive algorithm described in Section 4 to complete the partially destroyed solution, but in Step 2 we consider three different ways of ordering the container list. The first way is to order it by decreasing weight. Another alternative is ordering it by increasing weight and the third alternative consists of using decreasing weight but choosing the stack with most available cells instead of the stack with the fewest available cells. Then, three improvement moves, leading to three different solutions, are defined.

In Fig. 3 we can see an example of this improvement move. The small triangle in the top-left corner indicates that the slot has a power supply and a letter *R* indicates that it is a reefer container. The colors indicate the port of destination, dark gray meaning port 1 and light gray port 2. Overstowing containers are highlighed with a $*$. In the initial solution at the left-hand side (Fig. 3(a)), obtained by the tier-by-tier constructive algorithm, the number of overstowing containers is 15. We remove a percentage of the last containers stowed (Fig. 3(b)) and we stow them again using the constructive deterministic algorithm (Fig. 3(c)). The number of overstowing containers is reduced to 6, so the solution has been improved.

The second set of three moves is based on the removal of containers within a set of stacks. We randomly select a set of stacks in the solution and remove all the containers stowed in them. The solution is then completed again using the deterministic algorithm with the three different container orderings described before. We can see an example in Fig. 4. In this case stacks 1, 2, and 7 are removed. In the new solution there are no overstowing containers and in each of the new stacks 1, 2, and 7, all the containers have the same port of discharge.

5.3.2. Exchanging containers

In this move we try to improve the solution by exchanging containers between stacks in the current solution. The procedure is intended to allow more containers to be stowed, so it is only applied to solutions in which it is not possible to stow all the containers assigned to the location.

We try first to improve the distribution of weights among the stacks. We want to exchange containers between different stacks in such a way that a heavy stack increases its weight, without exceeding its limit, and a light stack is made lighter in order to be able to accommodate containers that have been left out due to the weight constraints. A similar movement is applied to the distribution of heights.

Fig. 5 shows an example of this move. The numbers below each stack correspond to the weight limit and the total weight of the containers in the stack. The numbers in the containers represent their weights. We exchange two containers in stack 3, which is the lightest stack, one of them with a container in stack 1 and the other with a container in stack 2, in order to

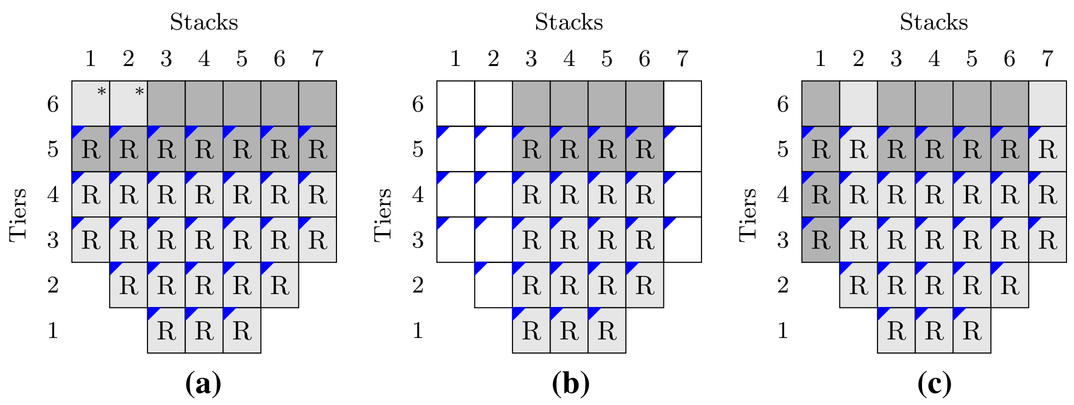

Fig. 4. Improving solutions. Removing stacks.

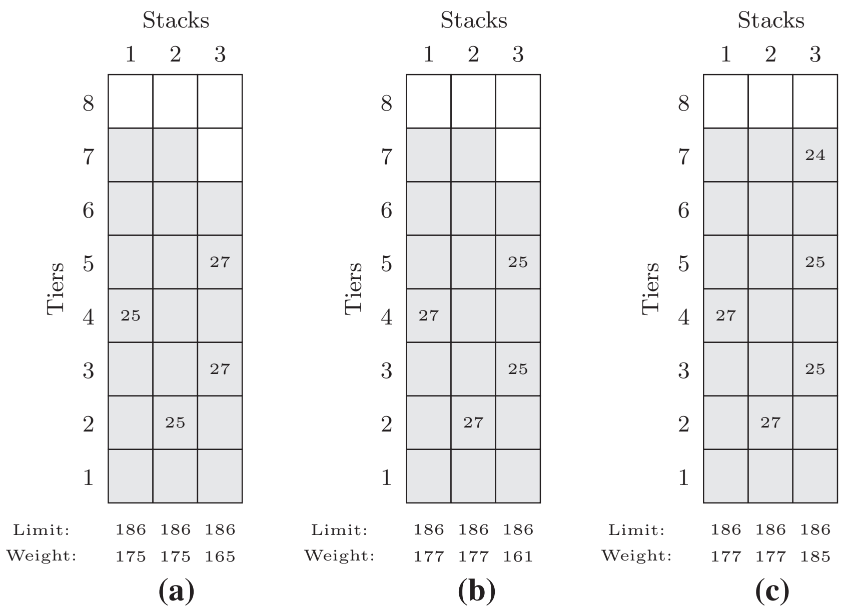

Fig. 5. Exchanging containers between stacks.

decrease its total weight, without exceeding the limits of the other stacks. Having reduced the load in stack 3 (Fig. 5(b)), it is possible to add one more container (Fig. 5(c)).

5.3.3. Statistical filter

We use a filter to speed up the iterations of GRASP, so that local search is not applied to all the solutions obtained by the randomized construction phase, but only to some promising unvisited solutions. This strategy is called GRASP filtering (Feo et al., 1994).

The filter tries to identify unpromising solutions, that is, solutions for which there is a high probability that they will not improve on the best known solution after the improvement phase. In order to do so, we use the information collected in the iterative process. Initially, the first 500 iterations are made, always going to the improvement phase and collecting at each iteration the values of sol_{ini}, the solution obtained by the randomized constructive phase, and sol_{ls}, the solution after the improvement phase. We consider the distribution of sol_{ini}/sol_{ls} and calculate its median. Then, in the subsequent iterations of GRASP, a solution obtained in the first phase with value sol_{ini} only goes to the improvement phase if $sol_{ini} \leqslant median \cdot sol_{best}$, where sol_{best} is the value of the best known solution.

The complete flow of the GRASP procedure appears in Algorithm 1.

Algorithm 1. Reactive GRASP

```
Require:
    $D = \{1, 2, \ldots, 9\}$, set of possible values of $\delta$
    $O_{best} = \infty;\ O_{worst} = 0$
    $Sum_{D_i^*} = 0$, sum of the solution values obtained using value $D_i$
    $P(D_i = D_i^*) = P_{D_i^*} = \frac{1}{|D|},\ \forall D_i^* \in D$, probability of using value $D_i$
    $numIter = 0$
    $M = \infty$, quality threshold
    $SO_{ini}$, set of first 500 solution values of randomized constructive phase
    $SO_{ls}$, set of first 500 solution values of improvement phase
Ensure:
    $S : Solution,\ O_s : Solution\ objectivefunction$
1:  while Stop Criterion not Satisfied do
2:      Choose $D_i^*$, with probability $P_{D_i^*}$
3:      $n_{D_i^*} \leftarrow n_{D_i^*} + 1$
4:      $numIter \leftarrow numIter + 1$
5:      $\{S, O_s\} \leftarrow ConstructiveRandomized(D_i)$
6:      if $O_s < M * O_{best}$ or $numIter < 500$ then
7:          repeat
8:              $\{S, O_s\} \leftarrow Improvement(S)$
9:          until S not Improved
10:         if $numIter = 500$ then
11:             $M \leftarrow ComputeMedian(SO_{ini}, SO_{ls})$
12:         end if
13:     end if
14:     if $O_s < O_{best}$ then
15:         $O_{best} \leftarrow O_s$
16:     end if
17:     if $O_s > O_{worst}$ then
18:         $O_{worst} \leftarrow O_s$
19:     end if
20:     $Sum_{D_i^*} \leftarrow Sum_{D_i^*} + O_s$
21:     if $mod(numIter, 500) = 0$ then
22:         $eval_{D_i^*} = \left(\frac{mean_{D_i^*} - O_{worst}}{O_{best} - O_{worst}}\right)^2,\ \forall D_i^* \in D$
23:         $P(D_i^*) = \frac{eval_{D_i^*}}{\sum_{D_{i'} \in D}(eval_{D_i'})},\ \forall D_i^* \in D$
24:     end if
25: end while
```

6. Computational experiments

The algorithms were coded in C++ and run on a Intel Core Duo 2.93 GHz with 4 GB of RAM. The IP models were implemented in CPLEX 12.5 and run with a time limit of 120 CPU seconds and using 4 threads. For the GRASP algorithms the stopping criteria are a time limit of 1 CPU second, or a maximum of 20,000 iterations, or 4000 iterations without improving the best known solution, whichever occurs first.

6.1. Test instances

We used two sets of instances to evaluate our approach. The first set, Set I, was the one used by Delgado et al. (2012), composed of 236 slot planning instances derived from complete stowage plans provided by a shipping company. Each instance was generated by restowing a random location in one of the stowage plans. Since the plans were applied in real

Table 2
Characteristics of the instances of sets I and II.

Set	Location	#	PODs			#Containers loaded			#Containers to load		
			Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
I		236	1	1.2	4	0	4.2	117	1	41.7	136
II	1	85	5	15.3	26	6	21.8	40	6	15.5	30
II	2	85	6	17.1	27	12	39.1	70	11	26.2	52
II	3	85	6	17.9	29	17	57.8	102	16	37.8	74
II	4	85	6	18.4	29	24	76.3	134	20	48.7	96
II	5	85	6	18.6	29	34	107.0	186	28	66.9	133
	Overall	661	1	11.7	29	0	40.4	186	1	40.0	136

life, we can assume that the containers were assigned to locations according to the preferences of stowage coordinators. In this first set, only in one instance was it not possible to stow all the containers in the location.

In order to extend the computational study, we created a new set, Set II, composed of 425 instances. Some of these instances had IMO containers and in general they were more difficult than those in Set I, because in some of them it was not possible to stow all the containers assigned to each location. There were 4 groups in total: Weight, HC, Capacity and IMO. Each group was aimed at a specific problem property, e.g. in the weight group all instances were forced to have a total weight of containers equal to a specified percentage of the total weight capacity.

- *Weight* instances (75), in which the weight was the most important factor. The total weight of the containers was set to 50%, 75% and 100% of the total weight allowed in the location.
- *HC* instances (125), in which the High-Cube containers determined the problem. The percentage of High-Cube containers was set to 0%, 10%, 20%, 40%, 50%.
- *Capacity* instances (75), in which the most important factor was the number of containers being stowed. The percentage of containers with respect to the capacity of the location was set to 50%, 75%, 100%.
- *IMO*instances (150), in which there were IMO containers, with their special loading constraints. We randomly selected 10 from each of the Weight, HC, and Capacity groups. The percentage of IMO containers was set to 15%, 30%, 45%, 60% and 75%.

We generated 5 instances for each location and designed 5 locations of different sizes, with location 1 being the smallest and location 5 the largest. For all instances, except for the *Capacity* group, the number of containers was set at random between 70% and 100% of the location capacity.

In Table 2 we can see the characteristics of all the instances. The first row shows the averages for the 236 instances of Set I, while the instances of Set II are divided, according to the location. The table shows that instances in Set II have in general more ports of discharge, containers already loaded, and containers to load. Instances of Set II also include IMO containers, besides the usual mix of 40′, 20′, and reefer containers.

6.2. Results of the integer linear model

The parameters in the objective function, $\alpha_f,\ f = 1, \ldots, 5$, have been set as follows:

$$\alpha_1 = 1000 * (|I| + |L| + 1), \quad \alpha_2 = 100, \quad \alpha_3 = 20, \quad \alpha_4 = 10, \quad \alpha_5 = 5.$$

These values, taken from Delgado et al. (2012), represent the preferences of the stowage coordinators consulted in their study.

Table 3 shows the results obtained for the two sets of instances when using the model described in Section 3, and compares them with those obtained by using the formulation suggested by Delgado et al. (2012), where the new constraints and

Table 3
Results for the integer formulations.

	#	Our integer formulation				IP (Delgado et al., 2012)				#Opt both
		UB	LB	Time	#Opt	UB	LB	Time	#Opt	
Set I	236	41	8	13.5	214	323	8	19.8	203	214
Weight	75	3862	331	81.5	28	20,286	277	84.4	26	28
HC	125	13,607	418	78.3	51	33,417	153	82.7	48	53
Capacity	75	2721	831	67.8	33	16,717	733	76.7	29	34
IMO	150	9228	975	82.3	44	19,616	925	83.2	46	46
Overall	661	5429	435	55.3	370	15,085	356	59.9	352	375

Table 4
Comparing strategies for the GRASP algorithm.

Type	#	Constructive		Randomized		Reactive		Local search		Filtering	
		#Out	#Opt.	#Out	#Opt.	#Out	#Opt.	#Out	#Opt.	#Out	#Opt.
Set I	236	75	160	17	190	16	190	1	209	2	209
Weight	75	145	1	67	8	67	7	41	15	43	15
HC	125	233	0	71	14	62	14	40	17	38	18
Capacity	75	140	0	86	10	87	11	68	17	66	17
IMO	150	933	1	416	29	396	30	297	33	296	32
Overall	661	1526	162	657	251	628	252	447	291	445	291
Time			<0.001		0.144		0.156		0.854		0.821
Cost			2868		1331		1266		916		902

Table 5
Comparing GRASP with IP on instances in which they load different amounts of containers.

	#	IP	GRASP	IP#out	GRASP#out
Set I	236	0	1	5	2
Weight	75	0	12	175	43
HC	125	3	28	969	38
Capacity	75	3	5	124	66
IMO	150	19	51	1031	296
Overall	661	25	97	2304	445

objective components have been added. The table shows the average values of the lower (LB) and upper bounds (UB), in thousands, for each group of instances, the average running times (Time) and the number of optimal solutions (#Opt). In the last column we can see the number of optimal solutions using the best lower bound and upper bound.

The results in Table 3 show that the new model finds significantly higher lower bounds and significantly lower upper bounds than the formulation by Delgado et al. (2012) for each group of instances. However, the large average gaps between lower and upper bound indicate that for some instances the solutions obtained can be very far from the optimal solutions, especially when the instance size increases.

6.3. Comparing different strategies for the GRASP algorithm

Table 4 compares different strategies for the GRASP algorithm:

- The *Constructive* deterministic algorithm.
- The *Randomized* constructive algorithm (at each iteration, δ is taken at random in $(0.5, 0.95)$).
- The *Reactive* GRASP algorithm (the value of the parameter δ is adjusted reactively as the iterative process proceeds).
- The Reactive GRASP algorithm with *Local Search*.
- The Reactive GRASP algorithm using the filter to select the solutions to be improved (*Filtering*).

For each method the table shows the sum of the number of containers rolled out (*#Out*) and the number of optimal solutions obtained (*#Opt*).[2] The last three rows show the sum of these values for all the problems, the average time, and the average cost per instance, in thousands.

We can see that the Reactive GRASP strategy slightly improves the results obtained when randomly selecting the value of δ, which in turn significantly improves the results of the deterministic constructive algorithm. Much better results are obtained when the solutions produced by the randomized constructive algorithm go to a Local Search procedure, although with a substantial increase in the running times. Finally, using the filter slightly reduces the computing times of the Local Search and obtains very similar results. Therefore our GRASP algorithm includes reactive adjustment of the parameter δ and local search with filtering.

6.4. Comparing GRASP with Integer Linear Programming models

In this section we compare the results of the GRASP algorithm with the best values obtained using the two integer formulations (denoted as IP), in order to assess the quality of the GRASP solutions. This comparison is made using two tables. In Table 5 we can see a comparison in terms of the number of containers loaded, columns three and four show the number of

[2] Calculated by comparing against the best lower bound found by both integer formulations.

Table 6
Comparing GRASP with IP on instances in which they load the same number of containers.

Type	#	Better	Equal	Worse
Set I	235	2	226	7
Weight	63	8	18	37
HC	94	16	19	59
Capacity	67	7	20	40
IMO	80	12	32	36
Overall	539	45	315	179

Table 7
Comparing with the integer formulation with instances grouped by size.

Containers	#	IP		GRASP		GRASP vs IP	
		#Out	#Opt	#Out	#Opt	#Worse	#Better
1–18	131	39	130	49	113	17	0
19–27	134	71	108	78	63	70	0
28–38	125	50	64	34	50	52	19
39–59	141	388	43	129	38	45	56
60–136	132	1756	30	155	27	21	67
	661	2304	375	445	291	203	142

instances in which one algorithm loads more containers, while columns five and six show the total number of containers rolled out by each algorithm. It can be observed that of the 661 test instances, there are only 122 in which GRASP and IP do not load the same number of containers. In most of these 122 instances GRASP obtains better solutions. The table shows how the integer formulations struggle to load containers, especially in the HC and IMO sets. GRASP, on the other hand, is able to load a considerably larger number of containers.

Table 6 contains the remaining 539 instances in which both algorithms load the same number of containers and compares the values of the solutions obtained by each algorithm. It can be observed that in the majority of cases, 315, the values coincide, while in 179 GRASP obtains worse values, with an average deviation of 3%, and in 45 GRASP gets better values, with an average improvement of 11%. These results show that GRASP can effectively be used to find high quality solutions for the slot planning problem.

Table 7 shows the results when the instances are grouped by size. The first column indicates the size of the problem measured as the number of containers to be loaded, the second column the number of instances in each group, the third and fourth columns the results achieved by Integer Formulations and columns five and six the results of the GRASP algorithm, in terms of the number of optimal solutions (#Opt) and the number of containers rolled out (#Out). The last two columns show the number of cases in which GRASP obtains worse and better solutions than IP. We can see that for small problems the results of integer formulations are better than GRASP, but as the instance size increases, the results of GRASP improve on the results of the integer formulations. The number of containers left out in the solutions of the integer formulations is very high when the size of the problem increases.

Table 8
Comparing with the constraint programming algorithm by Delgado et al. (2012) and the CBLS heuristic algorithm by Pacino and Jensen (2012).

Class	#	GRASP (1 s)			CBLS			CP (1 s)			CP (10 s)		
		%Sol	%Opt	Time	%Sol	%Opt	Time	%Sol	%Opt	Time	%Sol	%Opt	Time
1	13	100	100	6.3	100	59	0.1	100	100	0.1	100	100	0.1
2	22	100	100	15.4	100	77	3.6	91	91	3.6	91	91	21.6
3	13	100	100	7.8	100	92	0.5	100	100	0.5	100	100	0.5
4	78	100	99	37.3	100	92	6.0	96	95	6.0	99	99	19.7
5	36	100	94	22.3	97	58	7.1	92	89	7.1	92	92	39
6	15	100	100	6.0	93	80	1.2	100	93	1.2	100	100	5.4
7	14	100	93	9.0	93	79	2.3	64	57	6.8	64	64	53.5
8	14	100	100	6.1	93	43	1.5	93	93	1.5	93	93	10.5
9	17	94	82	12.3	94	47	5.2	76	76	5.2	88	88	36.5
10	8	100	100	4.8	100	88	0.7	100	100	0.7	100	100	0.7
11	6	100	83	3.2	50	17	1.3	83	83	1.3	83	83	10.3
Total	236	99.6	96.6	130.5	96.2	72.5	29.5	92.0	90.0	34.0	94.0	94.0	198.0

6.5. Comparing with other algorithms

Here we compare our GRASP approach to two state-of-the-art algorithms for the Slot Planning problem: the Constraint-Based Local Search (CBLS) by Pacino and Jensen (2012) and the Constraint Programming (CP) model of Delgado et al. (2012). Both approaches assumes that all containers must be loaded and do not support IMO segregation, thus comparison is only performed for Set I.

According to the literature, the CBLS and CP algorithms have been run on a Linux machine with two Quad Core Opteron processors at 1.7 GHz with 8 GB of memory. Even though this is an older machine, considering that the CBLS is run in parallel, we believe the comparison to be fair. Moreover, for this comparison, we consider any solutions from our GRASP approach that rolls out containers to be infeasible.

Table 8 presents the aggregated results of the three approaches on each of the instance groups of Set I. The fist column indicates the group ID and the second the number of instances it is composed of. Successively, for each solution method, column %Sol indicates the percentage of problems solved, column %Opt presents the percentage of problems solved optimally, and column Time shows the total running time needed to solve all the instances in the group. Note that two results are presented for the CP approach, one with a time limit of 1 s and one with a time limit of 10 s.

The first thing to notice is that the CBLS approach is generally faster than the GRASP algorithm. The quality of the solutions found by the GRASP approach are, however, much better. This indicates that the CBLS converges too fast toward some local minima from which it is not able to escape.

The CP model performs also extremely well in terms of execution time but lacks in robustness. It finds both fewer feasible and optimal solutions than the GRASP approach. When CP is given more time, the results improve but are still not competitive with GRASP. The runtime performance of the CP model, is highly dependant on the lower bound of the objective. The lower bound, in the same way as the entire CP model, assumes that all containers must be loaded and thus cannot be reapplied to the more general case where containers can be rolled out.

Two things are important to notice. First, the GRASP approach is able to find solutions for all the test instances. Only one instance in class 9 was not solved. However, it turns out that the instance is infeasible if all containers must be loaded. Second, all instances can be solved within the 1 s time limit imposed by the industry. These results show that in the special case where all containers must be loaded, the GRASP approach improves the state-of-the-art while retaining a comparable runtime performance.

7. Conclusion

We have studied the slot planning problem and introduced two features that have not been considered in previous studies. On the one hand, we explicitly include the minimization of rolled out containers and do not assume that all the containers fit into the location, because this is not always the case in practice. On the other hand, we take into account the separation rules that arise when handling containers with dangerous goods.

We have developed a new integer formulation. Although it obtains better results than other existing integer models, the computational results show that it is not suitable for solving real instances in the very short computing times required by the industry. Therefore we have developed a GRASP algorithm, combining constructive procedures, randomization strategies, and improvement moves. We have performed a computational study, using test instances available in the literature and a new set of instances that we have generated with different characteristics. The GRASP algorithm can be used as a black box, without further parameter tuning, and the results show that it performs well on all types of instances, improving on previously reported results. The handling of unloaded containers and the inclusion of segregation rules for dangerous cargo contributes to the further applicability of stowage planning optimization in the shipping industry.

In future research we plan to solve a more general slot planning problem in which the solution of the master planning phase is not given as a set of specific containers assigned to a location, but as a number of containers of each port, length, and weight class assigned to each bay. That involves an assignment phase of containers to bays, besides the loading problem for each bay. The problem is more complex, but also more realistic, and could be addressed using a metaheuristic scheme similar to that used in this study.

Acknowledgements

This study has been partially supported by the Spanish Ministry of Science and Technology, DPI2011-24977 and DPI2014-53665-P, cofinanced by FEDER funds, by Project PPII-2014-011-A, Consejeria de Educacion y Ciencia, Junta de Comunidades de Castilla-La Mancha, and by Generalitat Valenciana, PROMETEO/2013/049.

References

Alonso, M., Alvarez-Valdes, R., Parreño, F., Tamarit, J., 2014. A reactive GRASP algorithm for the container loading problem with load-bearing constraints. Euro. J. Indust. Eng. 8, 669–694.

Alvarez-Valdes, R., Parreño, F., Tamarit, J., 2013. A grasp/path Relinking algorithm for two- and three-dimensional multiple bin-size bin packing problems. Comp. Operat. Res. 40, 3081–3090.

Ambrosino, D., Anghinolfi, D., Paolucci, M., Sciomachen, A., 2010. An experimental comparison of different heuristics for the master bay plan problem. In: Proceedings of the 9th International Symposium on Experimental Algorithms, pp. 314–325.
Ambrosino, D., Paolucci, M., Sciomachen, A., 2015. Experimental evaluation of mixed integer programming models for the multi-port master bay plan problem. Flex. Serv. Manuf. J. 27 (2–3), 263–284.
Ambrosino, D., Sciomachen, A., 1998. A constraint satisfaction approach for master bay plans. Water Stud. Ser., 175–184
Ambrosino, D., Sciomachen, A., Tanfani, E., 2004. Stowing a containership: the Master Bay Plan Problem. Transport. Res. Part A: Policy Pract. 38 (2), 81–99.
Aslidis, AH., 1984. Optimal Container Loading. Massachusetts Institute of Technology.
Avriel, M., Penn, M., Shpirer, N., Witteboon, S., 1998. Stowage planning for container ships to reduce the number of shifts. Ann. Oper. Res. 76 (0), 55–71.
Aye, W.C., Low, M.Y.H., Ying, H.S., Jing, H.W., Min, Z., 2010. Visualization and simulation tool for automated stowage plan generation system. Proceedings of the International Multiconference of Engineers and Computer Scientists 2010 (IMECS 2010), Hong Kong, vol. 2, pp. 1013–1019.
Botter, R.C., Brinati, M.A., 1992. Stowage container planning: a model for getting an optimal solution. In: Proceedings of the Seventh International Conference on Computer Applications in the Automation of Shipyard Operation and Ship Design, pp. 217–229.
Davidor, Y. 1998. Method for Determining a Stowage Plan. US Patent 5809489, September 15.
Delgado, A., Jensen, R.M., Janstrup, K., Rose, T.H., Andersen, K.H., 2012. A constraint programming model for fast optimal stowage of container vessel bays. Euro. J. Operat. Res. 220 (1), 251–261.
Delorme, X., Gandibleux, X., Rodriguez, J., 2004. Grasp for set packing problems. Euro. J. Operat. Res. 153 (3), 564–580.
Ding, D., Chou, M.C., 2015. Stowage planning for container ships: a heuristic algorithm to reduce the number of shifts. Euro. J. Operat. 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. Heurist. 8 (6), 585–599.
Feo, T.A., Resende, M.G., 1995. Greedy randomized adaptive search procedures. J. Global Optim. 6 (2), 109–133.
Feo, T.A., Resende, M.G.C., Smith, S.H., 1994. A greedy randomized adaptive search procedure for maximum independent set. Operat. Res. 42 (5), 860–878.
Kang, J.-G., Kim, Y.-D., 2002. Stowage planning in maritime container transportation. J. Operat. Res. Soc. 53 (4), 415–426.
Li, F., Tian, C., Cao, R., Ding, W., 2008. An integer linear programming for container stowage problem. In: Bubak, M., van Albada, G., Dongarra, J., Sloot, P. (Eds.), Computational Science ICCS 2008, vol. 5101, pp. 853–862.
Monaco, M.F., Sammarra, M., Sorrentino, G., 2014. The terminal-oriented ship stowage planning problem. Euro. J. Operat. Res. 239 (1), 256–265.
Moura, A., Oliveira, J., 2005. A GRASP approach to the container-loading problem. IEEE Intell. Syst. 20 (4), 50–57.
Nugroho, S., 2004. Case-based stowage planning for container ships. In: The International Logistics Congress.
Pacino, D., Delgado, A., Jensen, R.M., Bebbington, T., 2011. Fast generation of near-optimal plans for eco-efficient stowage of large container vessels. Computational Logistics, vol. 6971. Springer, pp. 286–301.
Pacino, D., Jensen, R.M., 2012. Constraint-based local search for container stowage slot planning. In: Proceedings of the International Multiconference of Engineers and Computer Scientists, pp. 1467–1472.
Prais, M., Ribeiro, C.C., 2000. Reactive grasp: an application to a matrix decomposition problem in TDMA traffic assignment. INFORMS J. Comput. 12 (3), 164–176.
Prins, C., Lacomme, P., Prodhon, C., 2014. Order-first split-second methods for vehicle routing problems: a review. Transport. Res. Part C 40, 179–200.
Resende, M.G., Werneck, R.F., 2004. A hybrid heuristic for the p-median problem. J. Heurist. 10 (1), 59–88.
Rios-Mercado, R., Escalante, H., 2016. Grasp with path relinking for commercial districting. Exp. Syst. Appl. 44 (1), 102–113.
Sciomachen, A., Tanfani, E., 2003. The master bay plan problem: a solution method based on its connection to the three-dimensional bin packing problem. IMA J. Manage. Math. 14 (3), 251–269.
Shao, Y., Furman, K.C., Goel, V., Hoda, S., 2015. A hybrid heuristic strategy for liquified natural gas inventory routing. Transport. Res. Part C 53, 151–171.
Storck, 2015. Storck Guide Stowage & Segregation to IMDG Code (amdt. 37-14). Storck Verlag Hamburg.
Wilson, I., Roach, P., 2000. Container stowage planning: a methodology for generating computerised solutions. J. Operat. Res. Soc. 51 (11), 1248–1255.