European Journal of Operational Research 220 (2012) 251–261

Contents lists available at SciVerse ScienceDirect

European Journal of Operational Research

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

Innovative Applications of O.R.

A Constraint Programming model for fast optimal stowage of container vessel bays

Alberto Delgado [a,*], Rune Møller Jensen [a], Kira Janstrup [b], Trine Høyer Rose [b], Kent Høj Andersen [c]

[a] IT University of Copenhagen, Rued Langgaards Vej 7, 2300 Copenhagen S, Denmark
[b] Copenhagen University, Universitetspark 5, 2100 Copenhagen Ø, Denmark
[c] Århus University, Ny Munkegade 118, 8000 Århus C, Denmark

ARTICLE INFO

Article history:
Received 3 November 2010
Accepted 15 January 2012
Available online 24 January 2012

Keywords:
Container vessel stowage planning
Slot planning
Constraint Programming
Integer Programming

ABSTRACT

Container vessel stowage planning is a hard combinatorial optimization problem with both high economic and environmental impact. We have developed an approach that often is able to generate near-optimal plans for large container vessels within a few minutes. It decomposes the problem into a master planning phase that distributes the containers to bay sections and a slot planning phase that assigns containers of each bay section to slots. In this paper, we focus on the slot planning phase of this approach and present a Constraint Programming and Integer Programming model for stowing a set of containers in a single bay section. This so-called slot planning problem is NP-hard and often involves stowing several hundred containers. Using state-of-the-art constraint solvers and modeling techniques, however, we were able to solve 90% of 236 real instances from our industrial collaborator to optimality within 1 second. Thus, somewhat to our surprise, it is possible to solve most of these problems optimally within the time required for practical application.

1. Introduction

Approximately 90% of all non-bulk cargo is carried in container vessels. An important economical parameter for liner shipping companies is to be able to stow their vessels fast. This not only saves port fees but also decreases the speed at sea which saves bunker and reduces CO_2 emissions. Most stowage plans are produced manually by stowage coordinators using graphical tools, but due to the hardness of the problem and a potential for substantial savings, there recently has been an increasing interest in extending these tools with stowage planning optimization algorithms. These algorithms must also be fast, since stowage coordinators work under time pressure and may have to recompute plans due to loadlist changes or for the sake of evaluating different forecast scenarios. A runtime of more than 10 minutes is impractical according to our industrial collaborator within the liner shipping industry.

We have developed a stowage planning optimization approach that similar to the currently most successful approaches (e.g., [21,14,1]), decomposes the problem hierarchically. We use the 2-phase approach illustrated in Fig. 1. First, the *master planning* phase distributes the containers to load in the port to bay sections of the vessel. The slot planning phase then assigns the containers to load in each bay section to specific slots.

Our master planning approach has been presented in [16]. The focus of this paper is the slot planning phase.[1] A typical large container vessel has about 100 bay sections, which implies that the slot planning phase solves about 100 independent slot planning problems. Thus, given at most 10 minutes to generate complete stowage plans including master planning on hardware that does not support heavy parallelization, we aim at solving each slot planning problem in less than 1 second. This is non-trivial since slot planning is NP-hard and each bay section may hold up to several hundred containers.

Real slot planning problems include a wide variety of vessel structures and containers. To make their study practical, we introduce the Container Stowage Problem for Below Deck Locations (*CSPBDL*), a representative model for stowing containers in bay sections below deck formulated together with our industrial collaborator. Even though this is a simplified representation of the problem, it is to our knowledge the most detailed model published to date.

We then introduce an Integer Programming (IP) and Constraint Programming (CP) model for solving the *CSPBDL* to optimality. The CP model uses state-of-the-art modeling techniques including multiple viewpoints, specific domain pruning rules, and dynamic lower bounds. The IP model is a 0–1 formulation where cuts are introduced to strengthen the LP relaxation.

* Corresponding author. Tel.: +45 72185085; fax: +45 72185001.
E-mail addresses: alde@itu.dk (A. Delgado), rmj@itu.dk (R.M. Jensen), kj@transport.dtu.dk (K. Janstrup), m01thr@math.ku.dk (T.H. Rose), kent@imf.au.dk (K.H. Andersen).

[1] An early version of this work has been presented at the International Conference on Principles and Practice of Constraint Programming in 2009 [9].

0377-2217/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2012.01.028

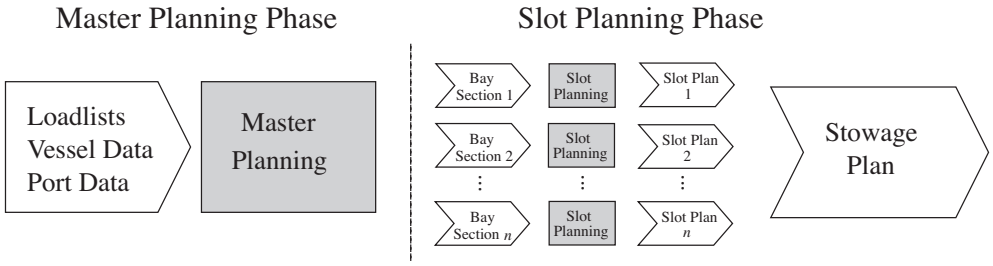

Fig. 1. The master planning and slot planning decomposition of stowage planning.

Despite the size and NP-hardness of slot planning problems, our computational results show that they often can be solved fast in practice. We have derived 236 test instances from real stowage plans from a system deployed by our industrial collaborator [12]. 92% of the instances are solved using a state-of-the-art constraint solver on our CP model within 1 second. Similar but slightly worse results were obtained with the IP model. Thus, somewhat to our surprise, it is possible to define optimal models of slot planning problems that can be solved fast enough to be used in stowage planning optimization tools.

The rest of the paper is organized as follows. Section 2 defines the problem we address in this paper and related work is presented and Section 3. In Section 4, we give a detailed description of our IP model. Section 5 gives a brief introduction to global constraint modeling, and in Section 6 we present our CP model. Computational results are presented in Section 7 and conclusions and directions for future work are discussed in Section 8.

2. Problem statement

A container vessel is a ship that transports box formed containers on a fixed cyclic route. The cargo space of a vessel is divided into *bays*. Each bay is divided into an *on deck* and *below deck* part by a *hatch cover*, which is a flat, leak-proof structure that prevents the vessel from taking in water and allows containers to be stowed on top of it (see Fig. 2). On and below deck parts of a bay are transversally divided into *stacks* that are one container wide, and are composed of two Twenty-foot Equivalent Unit (TEU) stacks and a single Forty-foot Equivalent Unit (FEU) stack. A *location* is a bay section consisting of a set of stacks that are either on or below deck. These stacks are not necessarily adjacent. The left drawing of Fig. 3 shows a typical arrangement of locations in a bay. A stack holds vertically arranged *cells* indexed by *tiers*. Each stack has a weight and height limit that must be satisfied by the containers allocated there. Cells in stacks are divided into two *slots*, a *fore* and an *aft*. The aft slot is situated toward the stern of the vessel, while the fore slot is allocated on the bow side.[2] Some slots have a power plug to provide electricity to containers in case their cargo needs to be refrigerated. Such slots are called *reefer* slots. *Quay cranes* at ports carry out the loading and unloading of containers from the vessel accessing only the top-most containers in stacks.

A *container* is a metal box in which goods can be stored. Each container has a weight, height, length, and port where it has to be unloaded (discharge port), and may need to be provided with electric power (*reefer container*). Containers are 20′, 40′, or 45′ long, and 8′6″ or 9′6″ high (*high-cube containers*). High-cube 20′ foot containers are rare and we assume they do not exist when modeling the slot planning problem. Empty 20′ and 40′ containers weight around two tons while their maximum weight is 24 and 30 tons, respectively. *Pallet wide* containers are slightly wider and can only be placed side-by-side in certain patterns. *IMO containers* carry dangerous goods and must be placed according to a complex set of separation rules. *Out-of-Gauge containers* carry cargo exceeding the inner dimensions of standard containers. The last three types of containers are often placed in special storage areas of the vessel.

Each cell can hold one 40′ or 45′ container or two 20′ containers. The 45′ containers, however, are normally only placed on deck, and some cells may be restricted to either 20′ or 40′ containers. In addition, *odd cells* may exist that only can hold one 20′ container due to the physical layout of the vessel. As an example, the right picture of Fig. 3 shows the slots of a stack below deck with a mixture of different 20′ and 40′ containers loaded. Containers already on board of the vessel when the stowage plan is made are called *loaded containers*. A container in a stack is *overstowing* another container in the same stack if it is stowed above it and discharged at a later port. An overstowing container is expensive, since it must be removed in order to discharge the overstowed container.

In this paper, we investigate the slot planning problem which is to assign a set of containers to slots in a location that may already hold loaded containers. Due to the large number of constraints and objectives involved in real slot planning, we have developed with our industrial collaborator a representative version of the problem for below deck locations called the *CSPBDL*. On deck locations share most constraints and objectives with below deck locations, thus we expect similar computational results for them. The *CSPBDL* covers all constraint and objective classes of the problem, and we anticipate a high correlation with a complete problem model in terms of solution algorithm performance. Specifically, the *CSPBDL* includes stacking rules for 20′ and 40′ containers, FEU and TEU stack overlapping, reefer containers, loaded containers, and weight and height constraints. The objectives include overstowage and three rules of thumb used by stowage coordinators to ease the stowing of containers in downstream ports. We limit the containers we consider to be 20′ and 40′ long, 8′6″ and 9′6″ high, and reefer and non-reefer. A feasible *CSPBDL* 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 only hold 40′ or 20′ containers).
(e) The sum of the heights and weights of the containers stowed in a stack are within the stack limits.
(f) All loaded containers must be stowed in their original slots and they cannot be swapped to any other slots.
(g) A cell must be either empty or with both slots occupied.

Additionally, an optimal *CSPBDL* minimizes the sum of the following objectives.

[2] The liner shipping industry uses another indexing standard for bays, stacks and tiers than the one presented in this paper which is irrelevant for our purposes.

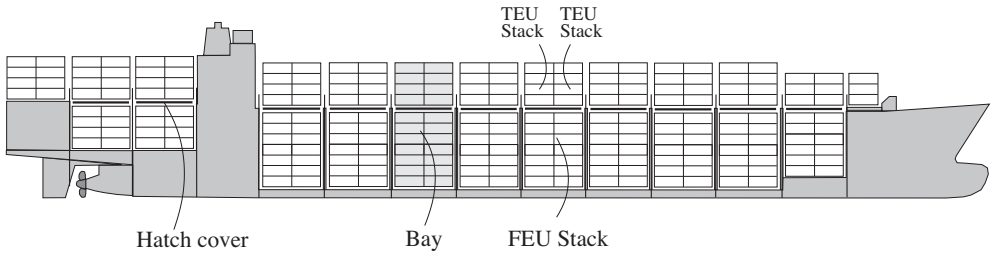

Fig. 2. The arrangement of cargo space in a container vessel.

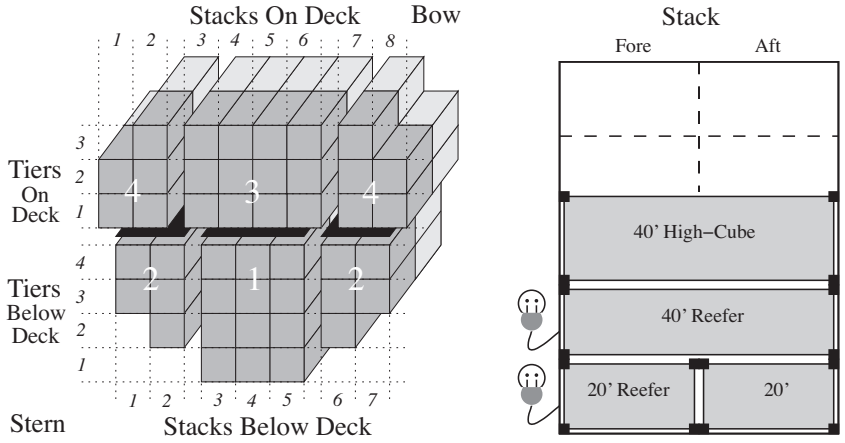

Fig. 3. Left: a front view of a vessel bay. There are four locations. Location 1 and 3 consist of inner stacks below and on deck, respectively, while location 2 and 4 consist of outer stacks in each side. Right: a side view of a partially loaded stack. Each power plug represents a reefer slot. Reefer containers are drawn with electric cords.

(h) Minimize overstows. A 100 unit cost is paid for each container overstowing any containers below.
(i) Avoid stacks where containers have many different discharge ports. A 20 unit cost is paid for each discharge port included in a stack.
(j) Keep stacks empty if possible. A 10 unit cost is paid for each stack used.
(k) Avoid loading non-reefer containers into reefer slots. A 5 unit cost is paid for each non-reefer container stowed in a reefer slot.

The second, third, and fourth objectives are rules of thumb of the shipping industry when generating slot plans for downstream ports in the route of a vessel. Using as few stacks as possible increases the available space in a location and reduces the possibility of overstowage in future ports, so does clustering containers with the same discharge port. Minimizing the reefer objective allows more reefer containers to be loaded in future ports. The cost units reflect the importance of each objective and has been defined by our industrial collaborator.

The CSPBDL is NP-Hard We show that the bin-packing problem can be reduced to the *CSPBDL*. All items in a bin-packing problem are defined as 40′, standard height, non-reefer containers with the same discharge port. Bins are defined as stacks where height limits are set to be sufficiently large to be non-restrictive and with no reefer slots. An optimal solution to an arbitrary bin-packing problem can be found by finding an optimal solution to the *CSPBDL*, showing that the *CSPBDL* is NP-Hard.[3]

[3] It can be shown that overstow minimization also is an NP-hard component of the *CSPBDL*, but the proof requires a version of the problem with uncapacitated stacks [5].

3. Literature review

Slot planning optimization algorithms have either been studied as embedded into single-phase models, or as a part of multi-phase decompositions for generating stowage plans. In the first category, Avriel et al. [6], Dubrovsky and Penn [10], and Ambrosino and Sciomachen [2] propose simple models to stow complete vessels that address similar problems to slot planning optimization. Avriel et al. introduce a 0–1 IP model and a heuristic called the *suspensory heuristic* to stow vessels described as a collection of columns and rows (a rectangular bay). They consider all containers to have the same features and focus on minimizing re-shifting (overstowage) of containers. Dubrovsky and Penn present a genetic algorithm model with the same assumptions as Avriel et al.'s. They claim, however, that their approach is flexible enough to include new constraints. Ambrosino and Sciomachen present a CSP model. Though this model is meant to stow a complete vessel, inter-bay stability constraints can be dropped in order to resemble the slot planning problem. This approach considers 20′ and 40′ containers, but lacks reefer and high-cube containers. Their objective is to minimize overstowage and maximize the number of containers loaded. Aslidis [4], Botter and Brinati [7], Sciomachen and Tanfani [18], and Li et al. [15] present more complex models that also include several of the constraints considered in the master planning problem. Aslidis introduce stacking heuristics for minimizing overstowage, while Botter and Brinati and Li et al. present 0–1 IP models. Botter and Brinati also present two heuristics to stow containers since their IP model is not scalable to real-life instances. Sciomachen and Tanfani introduce a heuristic approach based on the 3D-packing problem. These approaches consider 20′ and 40′ containers, and overstowage minimization. Sciomachen and Tanfani also consider high-cube containers.

In the second category, where slot planning problems are solved in connection with multi-phase approaches for complete vessel stowage planning, Wilson and Roach [21] briefly describe a tabu search algorithm for solving a version of slot planning that must have included reefer slots, length restrictions, minimized overstowage, and avoided discharge port mixing of stacks. They claim that near optimal solutions could be computed fast, but only experimental results for generating a complete stowage plan for a single vessel are described. Kang and Kim [14] describe an enumeration approach for solving a very simple version of slot planning, where only overstow minimization and sorting of 40′ containers after weight are considered. As for Wilson and Roach, no independent experimental evaluation of the algorithm is provided. Ambrosino et al. [3] describe a 0–1 IP model for stowing subsets of vessel bays holding containers with the same discharge port optimally. The model minimizes the time for stowing containers. 20′ and 40′ containers are considered, and containers are sorted according to weight in each stack. In the experimental section, complete stowage plans for a 198 and 2124 TEUs container vessels are generated. The maximum bay size is 20 TEUs for the small vessel and 120 TEUs for the big one. No computational time is provided for solving these sub-problems. In a later work [1], Ambrosino et al. present a constructive heuristic to solve the same sub-problem as the one described in [3], as part of a complete approach to stow vessels. Using this heuristic, they are able to stow a vessel of 5632 TEUs. The heuristic uses 11.8 seconds in average to stow all the bays but the physical layout of the vessel is not described in detail. Zhang et al. [23] and Yoke et al. [22] present multi-phase approaches where the problems solved during the slot planning phase are not independent of each other.

A deployed industrial system introduced by Guilbert and Paquin [12], that provides data to our experiments, solves slot planning problems as linear assignment problems with side constraints. Their model considers all containers and most of the constraints and objectives present in the *CSPBDL*. Overstowage is only considered with respect to loaded containers, and though they minimize mixed stacks with 20′ and 40′ containers, constraint *b* is not present.

4. The IP model

In this section, we introduce a binary IP model formulated to solve the *CSPBDL*. Table 1 presents the constant values, sets, and variables used in the model.

The first three sets of variables from Table 1, *o*, *p*, and *e*, are used for computing the cost of overstow (*h*), clustering (*i*), and using stacks (*j*) according to the *CSPBDL*. The variables in the fourth set, *c*, are the decision variables of the problem, and represent the stowage plan. The fifth set represents indicator variables introduced to model the overstowage objective. The IP model is defined as:

$$\min \quad 100\sum_{i\in I} o_i + 20\sum_{j\in J}\sum_{d\in D-\{1\}} p_{jd} + 10\sum_{j\in J} e_j + 5\sum_{j\in J} \times \sum_{k\in K_j}\left(R_{jk}\sum_{i\in F} c_{jki}(1-R_i^c) + \sum_{i\in T} c_{jki}\left(\frac{1}{2}R_{jk} - R_i^c\right)\right) \tag{1}$$

s.t.

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

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

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

Table 1
Constants, sets, and variables in the IP model.

Constants and sets	
I	Containers index set
J	Stacks index set
D	Discharge ports index set
K_j	Cells in stack j index set
T	20′ containers index set
F	40′ containers index set
R_{jk}	Number of reefer plugs in cell k of stack j
W_j^s	Weight limit of stack j in kilograms
H_j^s	Height limit of stack j in meters
H_i^c	Height in meters of container i
W_i^c	Weight in kilograms of container i
R_i^c	Indicates whether container i is reefer
A_{id}	Indicates whether container i is unloaded at port d. It is 1 if i is unloaded in port d, 0 otherwise
L	Loaded containers index set
M	Tuple set of loaded containers and corresponding cells indices: $\{(j,ki)\|j\in J, k\in K_j, i\in L\}$
Variables	
$o_i\in\{0,1\}$	Container i overstowing
$p_{jd}\in\{0,1\}$	At least one container in stack j being unloaded at d
$e_j\in\{0,1\}$	Stack j being used
$c_{jki}\in\{0,1\}$	Container i being stowed in cell k, stack j
$\delta_{jkd}\in\{0,1\}$	Container below cell k, stack j being unloaded before port d

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

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

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

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

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

$$\sum_{k'=1}^{k-1}\sum_{d'=2}^{d-1}\sum_{i\in I} A_{id'} c_{jk'i} - 2(k-1)\delta_{jkd} \leqslant 0 \quad \forall j\in J, k\in K_j, d\in D \tag{10}$$

$$A_{id}c_{jki} + \delta_{jkd} - o_i \leqslant 1 \quad \forall j\in J, k\in K_j, \quad d\in D, i\in I \tag{11}$$

$$e_j - c_{jki} \geqslant 0 \quad \forall j\in J, k\in K_j, i\in I \tag{12}$$

$$p_{jd} - A_{id}c_{jki} \geqslant 0 \quad \forall j\in J, k\in K_j, \quad d\in D, i\in I \tag{13}$$

$$c_{jki} = 1 \quad \forall (j,k,i)\in M \tag{14}$$

The objective function (1) is a weighted sum of the four objectives as defined in the *CSPBDL*. The first three objectives are calculated straightforward since there are specific variables in the model that account for them. The fourth objective is calculated by determining the number of non-reefer containers stowed in slots with reefer plugs. 40′ and 20′ containers are considered independently.

Inequality (2) ensures that there is either two 20′ or one 40′ container below a cell stowing a 40′ container, while inequality (3) constraints the containers below a cell stowing 20′ containers to be 20′ long (*b*). Inequality (4) requires that all cells stow at most either two 20′ or one 40′ container. Containers are forced to be stowed in exactly one cell by (5). Inequality (6) forces the number of 20′ containers in a cell to be 0 or 2, since the two sides of a stack must be synchronized (*g*). The reefer capacity of a cell is constrained by inequality (7), covering the fact that all reefer containers in a cell must be provided with a reefer plug each (*c*). The weight and height limits of stacks (*e*) are enforced by (8) and (9), respectively. Inequality (10) ensures that the variables δ_{jkd} are

assigned the correct value according to their semantics. These variables are then used in inequality (11) to assign the overstowage variables o_i for each container. Inequality (12) sets the variable related to the empty stack objective (j) for each stack, and inequality (13) does the same for the variables related to the clustering objective (i) for each stack at each discharge port. Loaded containers are assigned to their corresponding cell by equality (14).

4.1. Cuts

We add cuts that focus on removing solutions with non-integer values assigned to variables δ_{jkd} by the Linear Programming (LP) relaxation. First we decompose inequality (10) into several inequalities, one for each variable $c_{jk'i}$, that combined together are semantically equivalent to (10):

$$A_{id'}c_{jk'i} \leqslant \delta_{jkd} \quad \forall j \in J, k \in K_j, d \in D, i \in I, k' \in K', d' \in D' \tag{15}$$

where $K' = \{k | k \in \{1,\ldots,k-1\}\}$ and $D' = \{d | d \in \{2,\ldots,d-1\}\}$. We then increase the size of the left hand side term of (15) by considering at once all containers unloaded earlier than port d. Two inequalities are introduced to the model (16), (17), since 20′ and 40′ containers need to be treated differently. Additionally, cut (18) adds terms to the left hand side of (15) by considering all cells below cell k. The cuts are defined by:

$$\frac{1}{2}\sum_{i \in T'_d} c_{jk'i} \leqslant \delta_{jkd} \quad \forall j \in J, k \in K_j, d \in D, k' \in K' \tag{16}$$

$$\sum_{i \in F'_d} c_{jk'i} \leqslant \delta_{jkd} \quad \forall j \in J, k \in K_j, d \in D, k' \in K' \tag{17}$$

$$\sum_{k'=1}^{k-1} A_{id'}c_{jk'i} \leqslant \delta_{jkd} \quad \forall j \in J, k \in K_j, i \in I, d \in D, d \in D' \tag{18}$$

where $K' = \{k | k \in \{2,\ldots,k-1\}\}$, $D' = \{d | d \in \{2,\ldots,d-1\}\}$ and T'_d and F'_d are the set of 20′ and 40′ containers with discharge port earlier than d, respectively.

5. Global constraint modeling

A Constraint Satisfaction Problem (CSP) is a triple (X,D,C) where X is a set of variables, D is a mapping of variables to finite sets of integer values, with $D(x)$ representing the domain of $x \in X$ and $D(X) = \Pi_{x \in X} D(x)$ being the Cartesian product of domains, and C is a set of constraints. Each $c \in C$ is defined over a sequence $X' \subseteq X$ as a subset of allowed combinations of $D(X')$. A solution to a CSP is a complete assignment that maps every variable to a value from its domain that satisfies all constraints in C.

Constraint Programming (CP) is a relatively new technique that combines local consistency algorithms with search. The process of removing inconsistent values from the domain of the variables is called *propagation*. A depth-first backtracking search explores the search space of the problem incrementally. It extends a *partial solution* by selecting unassigned variables from X and assigning them to values from their domains. This selection process is called *branching*, and a strategy to select variables and values following a specific criteria is called a *branching strategy*. Propagation is executed every time a new branching is generated. If the domain of each variable has been reduced to a single value, the CP solver has found a solution to the CSP. For a partial solution, we refer to the minimum and maximum value of the domain of variable x as $\underline{x}$ and $\overline{x}$, respectively. A cost function is defined for a CSP in order to evaluate the quality of its solutions and *branch and bound* is used to find optimal solutions.

Constraints in CP share information through the variables in X. Each constraint has a scope $X' \subseteq X$, that often is relatively small compared to the size of X, limiting its propagation power. *Global constraints* have been introduced to overcome this. A global constraint groups together a set of small constraints capturing tractable structures for global propagation. Below is a brief description of the global constraints used in our CP model.

Let x be an integer variable, y a variable with finite domain, and $C = \{c_1,\ldots,c_n\}$ a set of constants. The *element constraint* [13] states that y is equal to the xth constant in C.

$$element(x,y,C) = \{(e,f) | e \in D(x), f \in D(y), f = c_e\}.$$

Let M be a deterministic finite automaton or a regular expression recognizing the strings in the language $L(M) \subseteq \Sigma^*$, and let $X = \{x_1,\ldots,x_n\}$ be a set of variables with $D(x_i) \subseteq \Sigma$ for $1 \leqslant i \leqslant n$. Then the *regular constraint* [17] is defined as

$$regular(X,M) = \{(d_1,\ldots,d_n) |\ \forall i.d_i \in D(x_i), d_1 \cdots d_n \in L(M)\}.$$

Let n and v be two integer values, and $X = \{x_1,\ldots,x_m\}$ a set of finite domain variables. The *exactly constraint* ensures that exactly n variables in X are assigned to value v.

$$exactly(n,X,v) = \{(d_1,\ldots,d_m) | \forall i.d_i \in D(x_i), |\{d_i | d_i = v\}| = n\}$$

Let $X = \{x_1,\ldots,x_n\}$ and $Y = \{y_1,\ldots,y_n\}$ be two sets of finite domain variables with domains $D(X) = D(Y) = \{1,\ldots,n\}$. The *channeling constraint* states that a value j assigned to a variable $x_i \in X$ represents the index of the variable $y_j \in Y$ that has been assigned value i from its domain. Formally

$$channeling(X,Y) = \{(e_1,\ldots,e_n,f_1,\ldots,f_n) |$$
$$\forall i,\ j.e_i \in D(x_i),\ f_j \in D(y_j),\ e_i = j \Longleftrightarrow f_j = i\}.$$

A channeling constraint is used to increase the propagation power of the model by connecting several isomorphic variable sets (also known as *viewpoints* [20]). If X is a set of variables representing positions with boxes $\{1,\ldots,n\}$ as domains and Y is a set of variables representing boxes with positions $\{1,\ldots,n\}$ as domains, then clearly a channeling constraint will link them consistently together. In particular, the channeling constraint embeds the *alldifferent constraint* that in our example ensures that a position only can hold one box and vice versa.

6. The CP model

Table 2 presents the index sets and constants of our CP model. All index sets are integer subsets. The stack in the left most part of the location has the lowest index in *Stacks*. Indices in *Slots* are assigned to physical slots as follows. For each cell, the aft and fore slots have consecutive indices. The slot indices in each stack are ordered bottom-up and the slot indices between stacks are ordered from left to right in the location. We have $Slots_k = Slots^F_k \cup Slots^A_k$, and $POD_i < POD_j$ iff the vessel calls the discharge port of container i before the discharge port of container j.

In our model, the decision variables represent the stowage plan for a set of containers to be stowed. We use two isomorphic representations. The first one defines a decision variable for each container in *Cont* to be stowed, and as domain of the variables the slots in *Slots*. The second one defines a decision variable for each slot in *Slots*, and as domain of the variables the set of containers in *Cont* to be stowed.

The two sets of decision variables mentioned above define two different viewpoints in our CP model. These two viewpoints are linked with a channeling constraint. The current formulation of the problem, however, does not allow a straightforward use of this constraint since in most of the cases the number of slots is larger than the number of containers, which breaks an important precondition of the channeling constraint. To tackle this issue, we modify the original definition of the problem by extending the number of containers with artificial containers to match the number of slots.

Table 2
Index sets and constants of the CP model.

$Stacks$	Stack index set
$Slots$	Slot index set
$Cont$	Container index set
$Slots^{\{A,F\}}$	Aft and Fore slots index set
$Slots_k$	Slots of stack k index set
$Slots_k^{\{A,F\}}$	Aft and Fore slots of stack k index set
$Slots^{\{R,\neg R\}}$	Reefer (R) and non-reefer ($\neg R$) slots index set
$Slots^{\neg RC}$	Slots in cells with no reefer plugs index set
$Slots^{\{20,40\}}$	20′ and 40′ capacity slots index set
$Cont^{\{V,L\}}$	Virtual (V) and loaded (L) containers index set
$Cont^{\{20,40\}}$	20′ and 40′ containers index set
$Cont^{40\{A,F\}}$	*Aft40*, *Fore40* 40′ containers index set
$Cont^{\{20R,40R\}}$	20′ and 40′ reefer containers index set
$Cont^{\neg R}$	Non-reefer 20′ and 40′ containers index set
$Weight_i$	Weight of container i
POD_i	Discharge port of container i
$Length_i$	Length of container i
$Height_i$	Height of container i
$Cont^{P=p}$	Number of containers with discharge port p
$Cont^{\{W=w,H=h\}}$	Number of containers with weight w and height h
$Cont^{\{NC,HC\}}$	Number of normal (NC) and high-cube (HC) containers
$stack_k^{\{w,h\}}$	Weight and height limit of stack k
$Classes$	Set of stack classes
$class^i$	Set of stacks of class i

Table 3
Variables of the CP model.

$C = \{c_1,\ldots,c_{\lvert Cont\rvert}\}$	$c_i \in Slots$, slot index of container i
$S = \{s_1,\ldots,s_{\lvert Slots\rvert}\}$	$s_j \in Cont$, container index of slot j
$L = \{l_1,\ldots,l_{\lvert Slots\rvert}\}$	$l_j \in Length$, length of container stowed in slot j
$H = \{h_1,\ldots,h_{\lvert Slots\rvert}\}$	$h_j \in Height$, height of container stowed in slot j
$W = \{w_1,\ldots,w_{\lvert Slots\rvert}\}$	$w_j \in Weight$, weight of container stowed in slot j
$P = \{p_1,\ldots,p_{\lvert Slots\rvert}\}$	$p_j \in POD$, POD of container stowed in slot j
$HS = \{hs_1,\ldots,hs_{\lvert Stacks\rvert}\}$	$hs_k \in \{0,\ldots,stack_k^h\}$, current height of stack k
$o^v \in \{0,\ldots,\lvert Cont\rvert\}$	Number of overstowing containers
$o^u \in \{1,\ldots,\lvert Stacks\rvert\}$	Number of used stacks
$o^p \in \{1,\ldots,\lvert Stacks\rvert\lvert POD\rvert\}$	Number of different discharge ports in each stack
$o^r \in \{0,\ldots,\lvert Slots^R\rvert\}$	Number of non-reefers stowed in reefer cells
$o \in \mathbb{N}$	Solution cost variable
$C^V \subset C$	Virtual containers
$S_i^E \subset S$	Slots with the same features in stack i

First, since a 40′ container occupies two slots, all 40′ containers are split in two parts, *Aft40* and *Fore40*, with the size of a single slot. All 40′ containers from $Cont$ and $Cont^{40}$ are replaced by *Aft40* and *Fore40* containers. We define $Cont^{40A}$ and $Cont^{40F}$ to be the indices of *Aft40* and *Fore40* containers, respectively. $Cont^{40A}$ and $Cont^{40F}$ have the same cardinality. *Virtual containers*, $Cont^V$, that will be stowed in slots meant to remain empty are also added. In the remainder of the paper, $Cont$ will refer to this extended set of containers. Finally, $Cont^{\neg R}$ is the set of non-reefer containers excluding virtual containers.

Table 3 summarizes the variables in the CP model. In addition to the two sets of decision variables, extra sets of auxiliary variables are defined to facilitate the modeling of the constraints and objectives. The objectives of the CP model are given by:

$$o^v = \sum_{i \in Slots^A} ov(i) \tag{19}$$

$$o^u = \sum_{k \in Stacks} \left(\sum_{j \in Slots_k} p_j > 0 \right) \tag{20}$$

$$o^p = \sum_{k \in Stacks} \left(\sum_{\rho \in POD} \sum_{j \in Slots_k} (p_j = \rho) \right) > 0 \tag{21}$$

$$o^r = \sum_{i \in Slots^R} (s_i \in Cont^{\neg R}) \tag{22}$$

$$o = 100o^v + 20o^p + 10o^u + 5o^r \tag{23}$$

Objective (19) calculates the total number of overstows. $ov(i)$ is the number of overstowing containers in a cell represented by its aft slot i. We have

$$ov(i) = \begin{cases} 2 & \text{if } s_i \in Cont^{20} \wedge p_i > minP(be(i)) \wedge p_{i+1} > minP(be(i)) \\ 1 & \text{if } (s_i \in Cont^{40} \wedge p_i > minP(be(i))) \vee \\ & (s_i \in Cont^{20} \wedge (p_i > minP(be(i)) \oplus p_{i+1} > minP(be(i)))) \\ 0 & \text{otherwise} \end{cases}$$

where $be(i)$ is the set of slots below slot i in the same stack, $minP(I)$ is the earliest discharge port among the containers assigned to a set of slots I, and $\oplus$ denotes the *exclusive-or* boolean operator. The empty stack objective (j), is represented by (20). The smallest discharge port index, 0, is assigned to virtual containers. Thus, when a stack i is empty, the sum of the values assigned to the subset of P variables in i is 0, otherwise the stack is being used. Objective (21) calculates the number of different discharge ports of containers stowed in each stack, and objective (22) counts the number of non-reefer containers stowed in reefer slots. Objective (23) defines the cost function of the *CSPBDL*. The branch and bound algorithm applied to solve this problem constrains the cost variable o of the next solution to be lower than that of the solution with lowest cost found so far. The constraints of the CP model are given by:

$$channeling(C,S) \tag{24}$$

$$c_{fore(i)} = c_i + 1, \quad \forall i \in \{1,\ldots,\lvert Cont^{40A}\rvert\} \tag{25}$$

$$element(s_i, l_i, Length) \quad \forall i \in Slots \tag{26}$$

$$element(s_i, h_i, Height) \quad \forall i \in Slots \tag{27}$$

$$element(s_i, w_i, Weight) \quad \forall i \in Slots \tag{28}$$

$$element(s_i, p_i, POD) \quad \forall i \in Slots \tag{29}$$

$$s_{pos(j)} = j \quad \forall j \in Cont^L \tag{30}$$

$$regular(Length_i^\pi, R) \quad \forall \pi \in \{A,F\}, i \in Stacks \tag{31}$$

$$s_i \notin Cont^{20R} \quad \forall i \in Slots^{\neg R} \tag{32}$$

$$s_i \notin Cont^{40R} \quad \forall i \in Slots^{\neg RC} \tag{33}$$

$$s_i \in Cont^{20} \quad \forall i \in Slots^{20} \tag{34}$$

$$s_i \in Cont^{40} \quad \forall i \in Slots^{40} \tag{35}$$

$$\sum_{j \in Slots_i^\pi} h_j \leqslant hs_i \quad \forall \pi \in \{A,F\}, i \in Stacks \tag{36}$$

$$\sum_{j \in Slots_i} w_j \leqslant stack_i^w \quad \forall i \in Stacks \tag{37}$$

Constraint (24) connects the two viewpoints such that both sets of variables C and S always have the same level of information. $fore(i)$ is the *Fore40* container bound to *Aft40* container i. Constraint (25) guarantees that the *Aft40* and *Fore40* part of a 40′ container are stowed in the same cell. Element constraints are used to bind all auxiliary variables introduced in the model to a viewpoint. Constraints (26)–(29) bind each slot variable to the auxiliary variables representing the length, height, weight and discharge port of the container stowed in the slot. *Loaded* containers are stowed in their pre-defined slots by constraint (30), where $pos(j)$ is the slot occupied by loaded container j. The valid patterns that containers stowed in stacks must follow according to their length are defined by (a) and (b). After assigning a length of 0 to virtual containers, we define a regular expression $R = 20^*40^*0^*$ that recognizes all the valid patterns according to these two constraints. Constraint (31) introduces a regular constraint for each aft and fore stack in order to restrict their stacking patterns to follow those defined by R. Constraints (32) and (33) model the reefer constraint (c). Constraints (34) and (35) restrict the domains of slots that just have 20′ or

40′ container capacity to be the set of 20′ and 40′ containers, respectively. The height limit of each stack in the location is constrained by (36). All containers stowed in each side of a stack must be less or equal to the variable representing the height limit of the stack.[4] Constraint (37) restricts the weight of all containers stowed in a stack to be within the limits.

6.1. Symmetry-breaking and implied constraints

We introduce a set of constraints to the CP model that aim at reducing the search space size and increase propagation. These constraints are implied by existing constraints and break symmetries either already present in the problem or introduced by our model representation.

$$exactly(\mathcal{V}, 0, |Cont^{V}|) \quad \forall \mathcal{V} \in \{P, W, H\} \tag{38}$$

$$exactly(P, p, Cont^{P=p}) \quad \forall p \in POD \tag{39}$$

$$exactly(W, w, Cont^{W=w}) \quad \forall w \in Weights \tag{40}$$

$$exactly(H, H^{\alpha}, Cont^{\alpha}) \quad \forall \alpha \in \{N, HC\} \tag{41}$$

$$h_j = h_k \quad \forall i \in Stacks, \; j,k \in \left\{(j,k) | j \in Slots_i^A, k \in Slots_i^F.eqCell(j,k)\right\} \tag{42}$$

$$sort(C^{V}) \tag{43}$$

$$s_i \notin Cont^{40A} \quad \forall i \in Slot^F \tag{44}$$

$$s_i \notin Cont^{40F} \quad \forall i \in Slot^A \tag{45}$$

$$s_j \leqslant s_k \quad \forall i \in Stacks, \; j,k \in \left\{(j,k) | j \in Slots_i^A, k \in Slots_i^F.eqType(j,k)\right\} \tag{46}$$

$$sort\left(S_i^E\right) \quad \forall i \in Stacks \tag{47}$$

$$lex(class^{i}) \quad \forall i \in Classes \tag{48}$$

Constraints (38)–(41) are implied constraints meant to improve the propagation power of the solver with respect to the auxiliary variables *P*, *W*, and *H*. Each individual auxiliary variable z_i is linked to a slot variable s_i with an element constraint. This ensures correctness but leads to weak propagation between the two sets of variables due to a lack of global perspective by the element constraints. To improve this, we first assign the value zero to the weight, height, and discharge port of virtual containers (these containers are not supposed to affect total height, weight, or overstowage of each stack). Then, constraint (38) limits the number of variables set to zero from *P*, *W*, and *H* to be the exact number of virtual containers. Additionally, constraints (39)–(41) restrict the number of variables from *P*, *W*, and *H* assigned to each possible discharge port, weight or height to match the total number of containers with such feature, respectively. Constraint (42) restricts the height of the containers stowed in the aft and fore slots of a cell to be equal. This is possible since both slots in a cell must be either empty or occupied at the same time (*g*), and there are no 20′ high-cube containers available to stow. The function *eqCell*(*j*,*k*) indicates that two slots *j* and *k* belong to the same cell.

The weight of the containers make each of them almost unique, limiting the possibility of applying symmetry breaking constraints. It is possible, however, to break some of the symmetries introduced into the problem by our model representation. First, since all virtual containers have the same features, it is not relevant where each container is stowed. Constraint (43) posts a sorting constraint over the virtual containers, forcing the slots where these containers will be stowed to follow a non-decreasing order. Second, splitting 40′ containers into *Aft40* and *Fore40* parts also generates symmetrical solutions that are broken by constraint (44) and (45). Third, constraint (46) limits the possibility of swapping containers between two slots of a cell that have the same features. The function *eqType*(*j*,*k*) indicates that two slots *j* and *k* belong to the same cell and have the same features, i.e., same reefer plug and length restrictions. Fourth, when all containers have the same discharge port, symmetrical solutions are generated by swapping containers stowed in slots with the same features within the same stack. Constraint (47) sorts in a non-decreasing order the indices of the containers stowed in slots with the same features of each stack. Indices are assigned to containers such that conflicts between constraint (47) and valid stacking patterns (31) are avoided. 20′ containers are assigned a lower index than 40′ containers, and virtual containers have the highest index possible. Finally, symmetries between stacks with identical characteristics are considered. Stacks are classified according to their features: slot capacity, reefer capacity, height and weight limit. Constraint (48) removes symmetrical solutions generated by the containers stowed in similar stacks being swapped with each other, by requiring a lexicographical ordering on the indices of the containers stowed in these stacks.

6.2. Branching strategies

Our branching strategy takes advantage of the structure of the model and uses the sets of different auxiliary variables in order to find high-quality solutions early in the search. We decompose the branching process into four sub-branchings: the first one focuses on finding high-quality solutions, the second and third on feasibility of two problematic constraints, and the fourth finds a valid assignment for the decision variables *S*. A detailed description of our branching strategy can be found in [8]. In the case of the first sub-branching, since three of the four objectives of the *CSPBDL* rely on the discharge port of the containers, we start by branching over the set of discharge port variables *P*. Variables bound to slots with containers that favor the clustering and overstowage objectives among the first free slots bottom-up of all stacks are preferred. After assigning all variables in *P*, we branch over the height and weight variables, *H* and *W*. We start by branching over *H* following a best-fit decreasing approach for selecting a stack, then we assign the smallest height possible among that of the containers to be stowed into the first free slot bottom-up in the stack. We use a similar approach for *W*, where the best fit is considered to be the stack with the greatest amount of free weight. Finally, we branch over *S* in order to generate a concrete stowage plan. The domain size of variables in *P* are considerably smaller than any of the viewpoints, making the process of finding valid assignments for *P* easier. Once a valid stowage plan is found, most of the time the search algorithm backtracks directly to the *P* variables in order to find solutions with a lower cost. Therefore, a large part of the search process concentrates on a much smaller sub-problem. Branching is performed over the remaining variables only when a solution with a lower cost is likely to be found.

6.3. Lower bounds

Five domain pruning rules are defined over partial solutions. Each rule solves a relaxed version of a sub-problem related to an objective or a constraint, generating lower bounds for their corresponding objectives and pruning values from the domain of the variables in the scope of the constraint. The lower bounds are described in more detail in [8].

Overstowage. To calculate a lower bound on the overstowage of a partial solution ρ, we define a new function $min\overline{P}(I) = min_{i \in I}(\bar{p}_i | p_i \in P)$ that selects the minimum upper bound $\bar{p}_i$ among the variables in *P*. Let $ov_{\rho}(i)$ be a function identical to $ov(i)$ where

[4] The *HS* variables are not necessary to define the height constraint but play an important role in the height constraint lower bound introduced in Section 6.3.

$\overline{minP}$ substitutes $minP$. It is easy to show that $ov_\rho(i)$ is a lower bound of $ov(i)$ for any completion of ρ [8]. The pruning effect of the lower bound is achieved by adding the constraint $o^v \geqslant \sum_{i \in Slots^A} ov_\rho(i)$. An additional pruning rule can be applied when the domain of o^v has been reduced to a single value that is equal to the lower bound. In this situation, we enforce that all containers below non-overstowing containers are discharged at a later port:

$$|D(o^v)| = 1 \wedge \sum_{i \in Slots^A} ov_\rho(i) = o^v \rightarrow$$
$$\forall i \in \{k \in Slots^A | ov_\rho(k) = 0\},\ j \in be(i).p_i \leqslant p_j.$$

Empty stack. In the remainder, we refer to container i as *unstowed* if the domain of c_i has more than one element. For the empty stack lower bound, a relaxation of the stowage problem is solved. The height capacity of the stacks is the only constraint considered and the containers to stow $Cont^N_\rho = \{i \in Cont | i \notin Cont^V, |D(c_i)| > 1\}$ are accounted as normal height containers. We first consider the *used stacks* of $\rho, Stacks^U_\rho = \{i \in Stacks | \exists j \in Slots_i, |D(s_j)| = 1, s_j \notin Cont^V\}$, where there are containers already stowed. The lower bound procedure stows as many containers as possible from $Cont^N_p$ in $Stacks^U_\rho$ such that the height capacity constraint is fulfilled. Once the used stacks are completely filled up, the empty stacks are sorted in decreasing order by height capacity and filled up with the remaining containers of $Cont^N_\rho$. The number of used stacks L^u_ρ is the sum of used stacks $\left|Stacks^U_\rho\right|$ plus the empty stacks necessary to stow all remaining containers. L^u_ρ is a lower bound of the number of used stacks of any completion of ρ since the approach to solve the relaxed problem uses a minimum number of stacks. The pruning effect is achieved by adding $o^u \geqslant L^u_\rho$.

Pure stack. As with the used stack lower bound, a relaxed assignment problem is solved considering just the height capacity constraint and all containers not yet stowed as normal height containers. First, we introduce an alternative definition of the pure stack objective. Let $Q_i = |\{k \in Stacks | \exists j \in Slots_k.p_j = i\}|$ be the number of stacks where at least one container with discharge port i is stowed. We can express the pure stack objective as $o^p = \sum_{i \in POD} Q_i$. For this definition of the pure stack objective, we introduce a lower bound for a partial solution ρ. Let $Cont^{N,P=i}_\rho$ be the set of unstowed containers in ρ with discharge port $i, Stacks^{P=i}_\rho$ be the set of stacks stowing at least one container with discharge port i, and $Stacks^{\neg P=i}_\rho = Stacks \setminus Stacks^{P=i}_\rho$ be the set of stacks where no container with discharge port i is allocated. Our goal is to generate a lower bound $L^p_\rho(i)$ independently for each Q_i, based on the approach followed to generate lower bounds for the used stacks objective. The pruning effect is achieved by adding $o^p \geqslant \sum_{i \in POD} L^p_\rho(i)$.

Reefer. A lower bound L^r_ρ for the reefer objective of a partial solution ρ can be deduced from a counting argument. Let $S^{\neg R}_\rho = |\{i \in Slots^R | s_i \notin Cont^R, |D(s_i)| = 1\}|$ denote the number of reefer slots stowing a non-reefer container in ρ. Clearly, $o^r \geqslant S^{\neg R}_\rho$ for any completion of ρ. We tighten the lower bound of the reefer objective by considering the unstowed reefer containers and the reefer slots with more than one container in their domain that will not stow a virtual container. Let C^R_ρ denote the number of unstowed reefer containers in ρ. Further, let S^{UR}_ρ be the reefer slots where no virtual container will be stowed. If $S^{UR}_\rho > C^R_\rho$ then at least $S^{UR}_\rho - C^R_\rho$ extra reefer slots will stow non-reefer containers. Thus, we can tighten L^r_ρ as follows:

$$L^r_\rho = \begin{cases} S^{UR}_\rho - C^R_\rho + S^{\neg R}_\rho : & \text{if } S^{UR}_\rho > C^R_\rho \\ S^{\neg R}_\rho : & \text{otherwise.} \end{cases}$$

The pruning effect is achieved as usual by adding $o^r \geqslant L^r_\rho$.

Height. The domains of auxiliary variables from sequences H and HS are tightened, and some conditions necessary for a partial solution to be viable are checked by solving three relaxed problems. First, the number of normal and high-cube containers that can possibly be stowed in the remaining free space of each stack is calculated. A stack j of some partial solution ρ has free height $h_\rho(j) = \overline{hs_j} - h^s_j$, where h^s_j denote the height of the stowed containers in stack j. Let $M^N_\rho(j)$ and $M^{HC}_\rho(j)$ denote the maximum number of normal and high-cube containers that can be placed in stack j, respectively. We then have

$$M^N_\rho(j) = \lfloor h_\rho(j)/h(N) \rfloor,$$
$$M^{HC}_\rho(j) = \lfloor h_\rho(j)/h(HC) \rfloor,$$

where $h(N)$ and $h(HC)$ denote the height of normal and high-cube containers. Let C^N_ρ and C^{HC}_ρ denote the number of unassigned normal and high-cube containers of ρ, respectively. Then, all possible stowage plans generated from partial solution ρ must satisfy

$$\sum_{j \in Stacks} M^N_\rho(j) \geqslant C^N_\rho \wedge \sum_{j \in Stacks} M^{HC}_\rho(j) \geqslant C^{HC}_\rho.$$

Second, since containers cannot hang in the air, they must be stowed consecutively, bottom-up in all stacks. Therefore, when the sum of the height of containers stowed below tier n equals to $\overline{hs_j}$, slots above tier n will not stow real containers. We stow virtual containers in slots of stack j that are above its height upper bound $\overline{hs_j}$. In the cases where the height of the container to be stowed is not known yet, it is assumed that the container will have normal height, since this generates an upper bound in the number of slots used in stack j. Additionally, the virtual containers are removed from slots that are below $\underline{hs_j}$, since these slots must stow real containers. Finally, we update $\underline{hs_j}$ by applying the bin packing propagation rule suggested in [19]

$$\underline{hs_j} \geqslant \sum_{i \in Cont} Height_i - \sum_{i \in Stacks \setminus \{j\}} \overline{hs_i},\ \forall j \in Stacks.$$

7. Experiments

The CP and IP models have been implemented in Gecode 3.3 [11] and CPLEX 12.2, respectively. All the experiments were run on a Linux machine with two Quad Core Opteron processors at 1.7 GHz and 8 GB of memory. Two hundred and thirty-six slot planning instances have been derived from complete stowage plans provided by our industrial collaborator. Each instance is made by restowing a random location in one of the stowage plans.[5] Since the plans have been applied in real life, we can assume that the containers have been assigned to locations according to the preferences of stowage coordinators.

In order to characterize hard instances, we have partitioned them into groups with different features. Table 4 presents the features of each group of instances, showing the group id, the number of instances in the group, the minimum, maximum, and average capacity of locations (Cap.) and number of containers to stow (Cont.) in TEUs, the features of the containers present group (40′, 20′, reefer, and high-cube), and the number of instances with 1, 2 or 3 different discharge ports in the group.

Notice that the instances have a low number of PODs. This is to expect from high quality stowage plans that avoid POD mixing. This does not imply that the instances are easy since instances with a single POD still embeds the bin-packing problem.

Fig. 4 shows the space utilization as a function of location size. The space utilization is the number of TEUs to stow divided by the total TEU capacity of the location. One hundred and seventy-two of

[5] This allows us to consider more aspects of the problem than the slot planning algorithms developed for [16]. We assign concrete containers rather than container types and include high-cube containers and real weight rather than weight classes.

Table 4
Grouping of instances.

Grp.	#Inst.	Cap. (TEUs)			Cont. (TEUs)			40′	20′	R	HC	#POD		
		Min	Max	Avg	Min	Max	Avg					1	2	⩾3
1	13	16	116	63	8	116	54	*				13		
2	22	8	168	68	8	136	52		*			22		
3	13	30	124	74	8	124	68	*	*			13		
4	78	6	208	79	2	202	63	*			*	78		
5	36	38	176	97	8	170	81	*	*		*	36		
6	15	42	172	73	16	74	46	*		*		15		
7	14	72	204	147	24	202	117	*	*	*	*	14		
8	14	40	148	96	40	136	87	*		*	*		14	
9	17	44	220	124	36	200	111	*	*	*	*		15	2
10	8	72	176	122	10	156	93	*		*	*		6	2
11	6	48	176	101	28	148	84	*	*	*	*		3	3

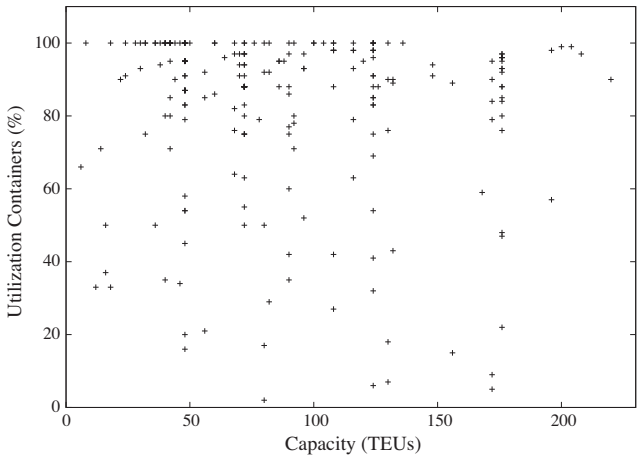

Fig. 4. Utilization as a function of location capacity.

the 236 instances have a space utilization above 80% which is to expect from real stowage plans.

7.1. Impact of CP enhancements

We first analyze the impact of the different enhancements of the CP model introduced in Section 6. We define four CP models. The *basic* model includes only the core constraints and objectives of the *CSPBDL* (19)–(37). A simple branching strategy is used in this model, where the stacks are filled up bottom-up from left to right and the container with the smallest index in the domain of the slot variable to be branched on is stowed in the slot. The *improved* model includes the symmetry-breaking and implied constraints from Section 6.1. Its branching strategy is similar to the *basic* model, however, the containers are assigned indices based on their features to avoid conflicts with some of the new constraints introduced. Finally, the *branching* and *advanced* models include the tailor-made branching strategy introduced in Section 6.2 and the lower bounds introduced in Section 6.3, respectively.

Recall that we aim at solving *CSPBDL* instances within 1 second. The solver can return an optimal solution before that, but after one second it must return its current solution. The results are summarized in Table 5. Each row corresponds to one of the instance groups introduced in Table 4. The first column provides the id of the group, the second one shows the percentage of instances of the group solved by all CP models, and the subsequent columns the percentage of instances solved and proven optimal by each model independently. The last row presents the results for the complete set of instances.

The total number of instances solved and proven optimal increases for each extension of the basic model. A more careful inspection of the table shows that this does not apply to all groups individually, but overall the impact of the model improvements are quite similar for each group. The total time needed for processing all instances is also reduced considerably for each model, from 186.3 seconds for the basic model, to 126.9 seconds for the improved model, 56.8 seconds for the branching model, and 34 seconds for the advanced model. Instances not solved were accounted by 1 second in the total time.

To compare the runtimes of individual instances, we analyze the subset of 156 instances (66.1%) solved by all four CP models (first column of Table 5). The left graph of Fig. 5, shows the runtime of the models for each instance, where label BA represents the basic model, IM the improved one, and BR and AD the branching and

Table 5
Percentage of instances solved and proven to optimality by the CP models.

Grp.	All (%)	Basic		Improved		Branching		Advanced	
		Sol (%)	Opt (%)	Sol (%)	Opt (%)	Sol (%)	Opt (%)	Sol (%)	Opt (%)
1	100	100	69	100	85	100	100	100	100
2	82	86	54	82	50	91	90	91	91
3	92	92	61	92	69	100	100	100	100
4	74	82	11	83	61	96	90	96	95
5	36	53	11	69	44	89	75	92	89
6	80	80	7	93	40	100	93	100	93
7	36	71	0	64	14	64	29	64	57
8	50	79	14	79	29	93	93	93	93
9	47	76	18	65	12	76	47	76	76
10	75	62	12	87	37	100	62	100	100
11	67	67	17	83	33	83	67	83	83
Total	66	77	21	80	48	89	80	92	90

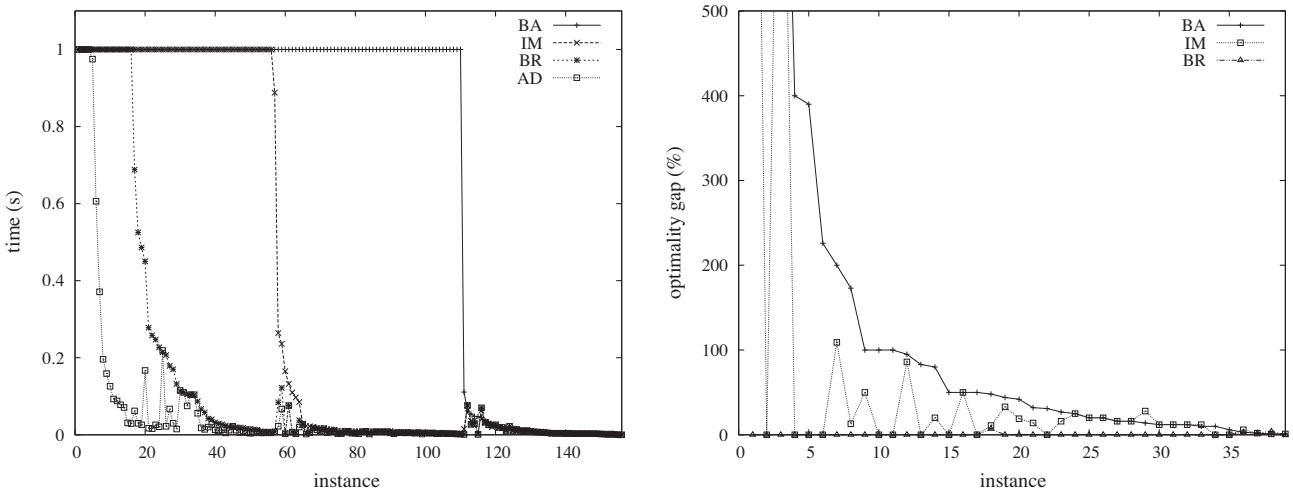

Fig. 5. Runtime (left) and optimality dominance (right) of the four CP models.

Table 6
Results for the CP and IP models with 1 second (upper table) and 10 seconds (lower table) limit. Dominant results are highlighted in bold.

	Sol (%)	Opt (%)	Time (s)	Sol (%)	Opt (%)	Time (s)
Grp.	CP 1 second			IP 1 second		
1	100	100	0.1	100	100	1.7
2	**91**	**91**	3.6	59	59	11.6
3	**100**	**100**	0.5	54	54	8.9
4	**96**	**95**	6.0	87	79	27.7
5	**92**	**89**	7.1	33	28	31.3
6	100	93	1.2	100	93	4.0
7	**64**	**57**	6.8	29	21	11.2
8	**93**	**93**	1.5	43	36	10.6
9	**76**	**76**	5.2	24	24	14.8
10	**100**	**100**	0.7	62	62	4.4
11	**83**	**83**	1.3	50	50	3.5
Total	**92**	**90**	34	64	59	129.8
Grp.	CP 10 seconds			IP 10 seconds		
1	100	100	0.1	100	100	1.8
2	91	91	21.6	**95**	91	50.4
3	**100**	**100**	0.5	92	85	35.3
4	**99**	**99**	19.7	96	94	87.0
5	**92**	**92**	39.0	72	56	192.0
6	100	**100**	5.4	100	93	13.0
7	64	**64**	53.5	64	29	102.8
8	**93**	**93**	10.5	79	64	74.1
9	**88**	**88**	36.5	53	41	112.3
10	**100**	**100**	0.7	88	62	31.5
11	**83**	**83**	10.3	67	50	30.5
Total	**94**	**94**	198	86	76	730.9

advance models, respectively. We have sorted the instances such that the expected runtime dominance between the models is clearly observable.

The right graph shows the optimality gap of 39 out of the 156 instances that at least one model solved suboptimally, the labels follow the same conventions as in the left graph. Again, we have sorted the instances to highlight a quite robust optimality dominance between the models.

Wrt. the hardness of different instance groups, the results in Table 5 indicate that instances with many features (group 8–11) are slightly harder than instances with few features. In particular, instances only stowing 40′ containers (group 1 and all even groups except 2) are significantly easier than instances stowing both 20′ and 40′ containers. Interestingly, there is no significant positive correlation between space utilization and runtime. This is surprising as one would expect the hardness of slot planning problems to increase with space utilization.

7.2. Comparing the performance of the IP and CP models

In this section, we compare the performance of the IP and CP model using a 1 second and 10 seconds runtime limit. Table 6 summarizes the percentage of instances solved, proven to optimality, and runtime of the models over the instance groups. The runtime of each instance group is calculated by summing up the time taken for solving all the instances in the group. Unsolved instances were accounted by the time limit of the experiment. The upper table shows the results of the experiments with a 1 second time limit. The CP model is clearly dominant in all groups in both, number of instances solved and instances solved optimally. In total, the CP model solves 28% more instances than the IP model in a quarter of the time. When the time limit is extended to 10 seconds (lower table in Table 6), the IP model increases the number of instances solved by a 22% and the ones solved optimally by a 17%. The runtime, however, increases by a factor of five. The instances solved by the CP model increases by a modest 1.7%, and the ones solved optimally by 3.8%, while the runtime increases a factor of six. The total time of the CP model, however, is still around a quarter of the IP model.

The hardness of instances for the IP model is similar to that of the CP model: instances with many features are slightly harder than instances with few, and instances stowing 40′ containers only are easier than instances stowing both 20′ and 40′ containers.

As in the CP experiment, we now focus on the instances solved by both models. Table 7 summarizes the results of the experiments with one and ten seconds time limit, showing the percentage of instances solved by both models, only the IP model, only the CP model, and none of them. For the 1 second time limit experiment, a total of 148 instances (63%) are solved by both models. All solutions produced by the CP model are optimal. The IP model produced 2 suboptimal solutions with optimality gap of 67% and 214%. For the 10 seconds time limit experiment, the number of instances solved by both models increased by 19% (45 instances). The number of suboptimal solutions also increased for the IP model (from 2 to 8), with optimality gap ranging from 3.1% to 317%. The suboptimal instances are spread over 6 different groups.

Table 7
Results for the instances solved by any of the two models with one and 10 seconds time limit.

Grp.	1 second				10 seconds			
	Both (%)	IP (%)	CP (%)	None (%)	Both (%)	IP (%)	CP (%)	None (%)
1	100	0	0	0	100	0	0	0
2	55	5	35	5	82	14	4	0
3	54	0	46	0	92	0	8	0
4	86	1	10	3	95	1	4	0
5	33	0	58	9	67	6	24	3
6	100	0	0	0	100	0	0	0
7	29	0	36	35	50	14	22	14
8	43	0	50	7	71	8	21	0
9	24	0	53	24	53	0	35	12
10	62	0	38	0	88	0	12	0
11	50	0	33	17	67	0	17	16
Tot	63	1	29	7	82	4	11	3

8. Conclusion

In this paper, we have introduced an accurate definition called *CSPBDL* of stowing a set of containers in a bay section. The *CSPBDL* is NP-hard and is an important sub-problem of the successful multi-phase approaches to stowage planning optimization. We have developed two CP and IP models to solve the *CSPBDL* optimally. Our computationally results show that both models can be solved fast and that it is possible to improve the performance of the CP model such that it can produce optimal solutions for 90% of 236 industrial instances in less than 1 second, which is well within the time requirements for practical stowage support tools. Future research includes improving the performance and stability of our solvers (e.g., diving heuristics and other techniques may be used to improve the IP model) and extending the *CSPBDL* to include on deck locations and special containers such as out-of-gauge, pallet-wide, and containers with dangerous goods.

Acknowledgements

We would like to thank the anonymous reviewers and the following academic and industrial collaborators: Christian Schulte, Mikael Lagerkvist, Thomas Stidsen, Jon Freeman, Robert John Milton, Nicolas Guilbert, Kim Hansen, and Thomas Bebbington. This research was partly funded by The Council for Strategic Research, within the programme "Green IT".

References

[1] D. Ambrosino, D. Anghinolfi, M. Paolucci, A. Sciomachen, An experimental comparison of different heuristics for the master bay plan problem, in: Proceedings of the 9th International Symposium on Experimental Algorithms, 2010, pp. 314–325.
[2] D. Ambrosino, A. Sciomachen, A constraint satisfaction approach for master bay plans, Maritime Engineering and Ports 36 (1998) 175–184.
[3] D. Ambrosino, A. Sciomachen, D. Anghinolfi, M. Paolucci, A new three-step heuristic for the master bay plan problem, Maritime Economics and Logistics 11 (1) (2009) 98–120.
[4] A.H. Aslidis, Optimal container loading. Master's thesis, Massachusetts Institute of Technology, 1984.
[5] M. Avriel, M. Penn, N. Shpirer, Container ship stowage problem: complexity and connection to the coloring of circle graphs, Discrete Applied Mathematics 103 (2000) 271–279.
[6] M. Avriel, M. Penn, N. Shpirer, S. Witteboon, Stowage planning for container ships to reduce the number of shifts, Annals of Operations Research 76 (1998) 55–71.
[7] R. Botter, M.A. Brinati, Stowage container planning: a model for getting an optimal solution, Proceedings of the IFIP TC5/WG5.6 Seventh International Conference on Computer Applications in the Automation of Shipyard Operation and Ship Design, vol. VII, North-Holland Publishing Co., Amsterdam, The Netherlands, 1992, pp. 217–229.
[8] A. Delgado, R.M. Jensen, K. Janstrup, T.H. Rose, K.H. Andersen, A constraint programming model for fast optimal stowage of container vessel bays. Tech. Rep. TR-2010-133, IT University of Copenhagen, 2010.
[9] A. Delgado, R.M. Jensen, C. Schulte, Generating optimal stowage plans for container vessel bays, in: Proceedings of the 15th International Conference on Principles and Practise of Constraint Programming (CP-09), Springer, 2009, pp. 6–20.
[10] O. Dubrovsky, G.L.M. Penn, A genetic algorithm with a compact solution encoding for the container ship stowage problem, Journal of Heuristics 8 (2002) 585–599.
[11] Gecode Team, Gecode: Generic constraint development environment, 2006. <http://www.gecode.org>.
[12] N. Guilbert, B. Paquin, Container vessel stowage planning, Patent Publication US2010/0145501, 2010.
[13] P.V. Hentenryck, J.P. Carrillon, Generality vs. specificity: an experience with AI and OR techniques, in: Proceedings of the National Conference on Artificial Inteligence (AAAI), ACM press, 1988, pp. 660–664.
[14] J. Kang, Y. Kim, Stowage planning in maritime container transportation, Journal of the Operations Research Society 53 (4) (2002) 415–426.
[15] F. Li, C. Tian, R. Cao, W. Ding, An integer programming for container stowage problem, in: Proceedings of the International Conference on Computational Science, Part I, LNCS, vol. 5101, Springer, 2008, pp. 853–862.
[16] D. Pacino, A. Delgado, R.M. Jensen, T. Bebbington, Fast generation of near-optimal plans for eco-efficient stowage of large container vessels, in: Proceedings of the Second International Conference on Computational Logistics (ICCL'11), Springer, 2011, pp. 286–301.
[17] G. Pesant, A regular language membership constraint for finite sequences of variables, in: Proceeding of Principles and Practice of Constraint Programming, Lecture Notes in Computer Science, vol. 3258, Springer, 2004, pp. 482–495.
[18] A. Sciomachen, A. Tanfani, The master bay plan problem: a solution method based on its connection to the three-dimensional bin packing problem, IMA Journal of Management Mathematics 14 (2003) 251–269.
[19] P. Shaw, A constraint for bin packing, in: Proceeding of Principles and Practice of Constraint Programming, Lecture Notes in Computer Science, vol. 3258, Springer, 2004, pp. 648–662.
[20] B. Smith, Modelling, in: F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Elsevier, 2006. Chapter 11.
[21] I.D. Wilson, P. Roach, Container stowage planning: a methodology for generating computerised solutions, Journal of the Operational Research Society 51 (11) (2000) 248–255.
[22] M. Yoke, H. Low, X. Xiao, F. Liu, S.Y. Huang, W.J. Hsu, Z. Li, An automated stowage planning system for large containerships, in: Proceedings of the 4th Virtual International Conference on Intelligent Production Machines and Systems, 2009.
[23] W.-Y. Zhang, Y. Lin, Z.-S. Ji, Model and algorithm for container ship stowage planning based on bin-packing problem, Journal of Marine Science and Application 4 (3) (2005).