A Constraint Programming Model for Fast Optimal Stowage of Container Vessel Bays

Alberto Delgado (IT University of Copenhagen, Denmark)
Rune Møller Jensen (IT University of Copenhagen, Denmark)
Kira Janstrup (Department of Transport, Technical University of Denmark)
Trine Høyer Rose (Department of Mathematical Sciences, University of Copenhagen)
Kent Høj Andersen (Department of Mathematical Science, Århus University, Denmark)

IT University Technical Report Series **TR-2010-133**

ISSN 1600–6100 **October 2010**

ISSN 1600–6100

ISBN 9788779492264

Copies may be obtained by contacting:

IT University of Copenhagen
Rued Langgaards Vej 7
DK-2300 Copenhagen S
Denmark

Telephone: **+45 72 18 50 00**
Telefax: **+45 72 18 50 01**
Web `www.itu.dk`

A Constraint Programming Model for Fast Optimal Stowage of Container Vessel Bays

Alberto Delgado (IT University of Copenhagen, Denmark)
Rune Møller Jensen (IT University of Copenhagen, Denmark)
Kira Janstrup (Department of Transport, Technical University of Denmark)
Trine Høyer Rose (Department of Mathematical Sciences, University of Copenhagen)
Kent Høj Andersen (Department of Mathematical Science, Århus University, Denmark)

1 Introduction

Seaborne transportation has become the most important transportation mean for trade cargo. 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 increases the buffer time in schedules which saves bunker due to reduced speeds. For this reason, there has recently been an increasing interest in developing stowage planning optimization algorithms that can provide decision support for human stowage coordinators. These algorithms must also be fast, since stowage coordinators work under time pressure and may have to recompute plans due to loadlist changes. The desire of our industrial collaborator within the liner shipping industry is to use at most 10 minutes of computation time.

A container vessel stowage plan assigns containers to slots on the vessel. It is hard to generate good stowage plans manually since containers cannot be stacked freely due to global constraints like stability, stress forces, and many interfering local rules for arranging containers in stacks.

Early work on stowage planning optimization has mainly focused on "flat" models that introduce a decision variable for each possible slot assignment of the containers (e.g.,Botter and Brinati [1992], Giemesch and Jellinghaus [2003]). None of these models scale beyond small feeder vessels of a few hundred 20-foot equivalent units (TEUs). Approaches with some scalability are heuristic (e.g., Ambrosino et al. [2004], Avriel et al. [1998], Dubrovsky and Penn [2002]), in particular by decomposing the problem hierarchically (e.g.,Ambrosino et al. [2006], Kang and Kim [2002], Wilson and Roach [2000], Gumus et al. [2008], Ambrosino et al. [2009]). These hierarchically decompositions are based on a natural two-level decomposition of the problem that follows the approach used by stowage coordinators. At the first level, containers are assigned to locations (stowage areas in bays) such that the re-handling of containers is minimized, crane utility in ports is maximized, and high-level constraints such as stability and stress requirements of the vessel are satisfied. At the second level, each location is stowed independently by assigning the containers to specific physical positions called slots such that stacking rules and intra location objectives are satisfied. Thus, for the decomposed methods, an important sub-problem is to stow a given set of containers into a location. Since modern vessels typically are divided into more than 100 locations, these sub-problems must be solved within a few seconds in order to solve the overall stowage planning problem in less than 10 minutes, unless heavy parallelization is used.

In this paper, we present the first accurate model of these sub-problems called the *CSPUDL* that we have formulated in collaboration with our industrial partner. We then introduce an Integer Programming (IP) and Constraint Programming (CP) model for solving the *CSPUDL* to optimality. The CP model uses state-of-the-art modelling 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.

It is to our knowledge the first time that modern CP modelling techniques have been applied to stowage planning, and even though we are dealing with an optimization problem, which is typically not a type of problem where CP techniques are applied, it turns out that our instances are solved faster with state-of-the-art CP software applied to our

CP model than with state-of-the-art IP software applied to our IP model. Furthermore, our CP model has the advantage that it is easy for industrial modelers to understand, maintain and extend.

In general, the *CSPUDL* is NP-Complete when stacks are uncapacitated Avriel et al. [2000], but our experimental evaluation of the IP and CP models shows that these sub-problems often are very easy to solve in practice. We have generated 236 test instances by re-stowing containers assigned to locations in real stowage plans used by our industrial collaborator. 92% of the instances could be solved by using state-of-the-art CP software on our CP model within one second.

The rest of the paper is organized as follows: Section 2 provides a definition of the problem that we address in this paper. Section 4 gives a brief introduction to global constraint modelling. In Section 3 we give a detailed description of our IP model, and in Section 5 we present our CP model. The experimental evaluation is presented in Section 6. Related work is presented in Section 7, and finally Section 8 draws conclusions and discusses directions for future work.

2 Container stowage

A container vessel is a ship that transports box formed containers on a fixed cyclic route. The cargo space in a vessel is divided in sub-sections called *bays*, each bay is divided into an *over deck* and *under 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 Figure 1). An under deck stack, as depicted in the left picture of Figure 2, is composed of two Twenty-foot

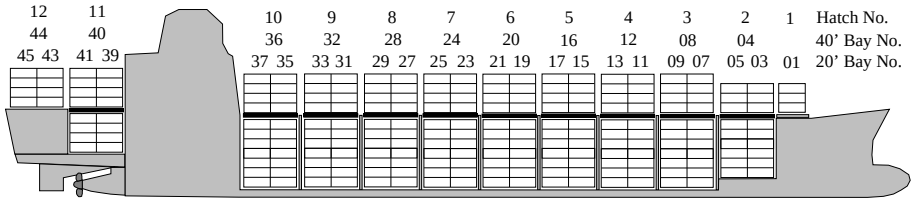

Figure 1: The arrangement of bays in a container vessel. The hatch cover is drawn as a thick line between the over and under deck part of a bay.

Equivalent Unit (TEU) stacks and a Forty-foot Equivalent Unit (FEU) stack, holding vertically arranged $cells$ indexed by $tiers$. $Quay\,cranes$ at ports carry out the loading and unloading of containers in the vessel, accessing only those containers on top the of each stack at a time.

A *location* is a set of stacks that are either over or under deck. These stacks are not necessarily adjacent, but the stacks are all either over or under deck. The left drawing of Figure 2 shows a typical arrangement of locations in a bay. Each stack has a weight and height limit that must be satisfied by the containers allocated there. Cells in stacks are divided in two *slots*, *fore* and *aft*. The aft slot refers to the position toward the stern on the vessel, while fore slots are allocated on the bow side. 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. Right picture of Figure 2 shows the structure of a stack.

A *container* is a box in which goods are 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*). In an under deck location, containers can be 20 or 40 feet long and 8'6" or 9'6" high. Containers that are 9'6" high are called *high-cube containers*. High-cube containers are 40 feet long. Each cell in a stack can hold one 40-foot container or two 20-foot containers, but there may be some cells in the location that are limited to a single length. Container that are already on board the vessel when the stowage plan is made are called *loaded containers*. A container in a stack is *overstowing* another container in the 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.

As described in the introduction, we investigate the sub-problem of stowing individual locations. Due to the very large number of constraints and objectives involved in stowing containers in over and under deck locations, we focus on under-deck locations and have formulated a representative problem called the *CSPUDL* for stowing containers in

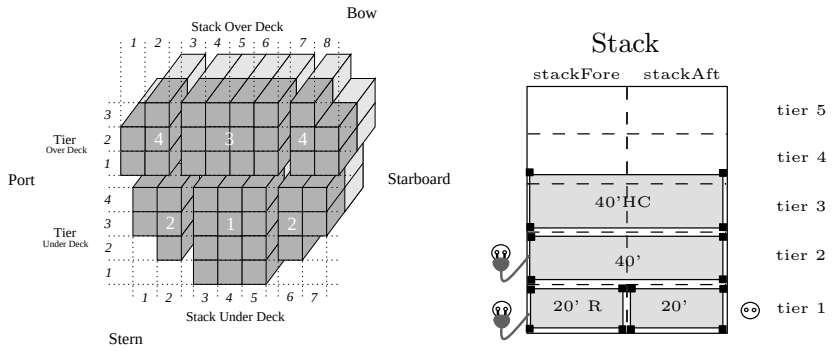

Figure 2: Left: a front view of a vessel bay. There are four locations. Location 1 (3) consists of inner stacks under (over) deck, while location 2 (4) consists of outer stacks in each side. This arrangement makes it simpler to achieve transverse stability when distributing containers to locations. Each stack consists of a set of cells where each cell is divided into a fore (light grey) and aft slot (dark grey). Right: a side view of a partially loaded stack. Each power plug represents a reefer slot. Reefer containers are drawn with electric cords.

under deck locations with our industrial collaborator. Our focus on under deck locations is mainly due to resource limitations of our work. Over deck locations share most constraints and objectives with under deck locations, and we expect similar computational results for these locations. The *CSPUDL* covers all constraint and objective classes of the problem and we expect that it has a high correlation with the complete problem model in terms of solution algorithm performance. Specifically, the CSPUDL includes stacking rules for 20 and 40-foot 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 achieve robustness. The *CSPUDL* excludes break-bulk cargo, out-of-gauge containers, and odd slots (i.e., cells that can only hold a single 20-foot container). In addition, we do not consider IMO and pallet-wide containers since these are often placed in special locations.

The *CSPUDL* is defined as follows. A feasible stowage plan for an under deck location must satisfy the following constraints.

Assigned cells must form stacks (containers stand on top of each other in the stacks. They can not hang in the air).	(1)
20-foot containers can not be stacked on top of 40-foot containers.	(2)
A 20-foot reefer container must be placed in a reefer slot. A 40-foot reefer container must be placed in a cell with at least one reefer slot, either Fore or Aft.	(3)
The length constraint of a cell must be satisfied (some cells only hold 40 or 20-foot containers).	(4)
The sum of the heights and weights of the containers stowed in a stack are within the stack limits.	(5)
All loaded containers must be stowed in their original slots and they can not be swapped to any other slots.	(6)
A cell must be either empty or with both slots occupied.	(7)

Every stowage plan for a location that satisfies these constraints is valid, but since the problem we are solving here is

to find the best stowage plan possible, a set of objectives is defined to evaluate the quality of the solutions.

Minimize overstows. A 100 unit cost is paid for each container overstowing any containers below.	(8)
Avoid stacks where containers have many different discharge ports. A 20 unit cost is paid for each discharge port included in a stack.	(9)
Keep stacks empty if possible. A 10 unit cost is paid for every new stack used.	(10)
Avoid loading non-reefer containers into reefer cells. A 5 unit cost is paid for each non-reefer container stowed in a reefer cell.	(11)

The second, third, and fourth are rules of thumb of the shipping industry with respect to generating stowage 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 industry partner.

3 The IP model

In this section we introduce the binary IP model formulated to solve the *CSPUDL*. Table 1 presents the constant values and sets used in the model. Table 2 presents the variables of the model.

I	Number of containers
J	Number of stacks
D	Number of discharge ports
K_j	Number of cells in stack j
T	Set of 20-foot containers
F	Set of 40-foot containers
r_{jk}	Number of reefer plugs in cell k of stack j
W_j	Weight limit of stack j in kilograms
H_j	Height limit of stack j in meters
s_i	Whether container i is a 20-foot container
l_i	Whether container i is a 40-foot container
h_i	Height in meters of container i
w_i	Weight in kilograms of container i
r_i	Whether container i is reefer
a_{id}	Whether container i is unloaded at port d

Table 1: Constants and sequences in the IP model

$O_{jk} \in \{0,1\}$	Whether container stowed in cell k, stack j overstows container below
$P_{jd} \in \{0,1\}$	Whether there is at least one container in stack j being unloaded at d
$E_j \in \{0,1\}$	Whether stack j is being used
$c_{jki} \in \{0,1\}$	Whether container i is stowed in cell k, stack j
$\delta_{jkd} \in \{0,1\}$	Whether a container below cell k, stack j is unloaded before port d

Table 2: Variables in the IP model

The first three sets of variables, O, P, and E, are used to represent the overstow (8), clustering (9), and freestack (10) objective of the *CSPUDL*. The fourth set of variables, c, represents the stowage plan, and the fifth set is introduced

to model the overstowage objective. The IP model can then be defined as:

$$min\ \ \theta = 100\sum_{j=1}^{J}\sum_{k=1}^{K_j} O_{jk} + 20\sum_{j=1}^{J}\sum_{d=2}^{D} P_{jd} + 10\sum_{j=1}^{J} E_j$$
$$+ 5\sum_{j=1}^{J}\sum_{k=1}^{K_j}((\frac{1}{2}\sum_{i\in T} c_{jki} + \sum_{i\in F} c_{jki})r_{jk} - \sum_{i=1}^{I} r_i c_{jki}) \tag{12}$$

$s.t.$

$$\frac{1}{2}\sum_{i\in T} c_{j(k-1)i} + \sum_{i\in F} c_{j(k-1)i} - \sum_{i\in F} c_{jki} \geq 0 \qquad \forall j\forall k \tag{13}$$
$$\sum_{i\in T} c_{jki} - \sum_{i\in T} c_{j(k-1)i} \leq 0 \qquad \forall j\forall k \tag{14}$$
$$\frac{1}{2}\sum_{i\in T} c_{jki} + \sum_{i\in F} c_{jki} \leq 1 \qquad \forall j\forall k \tag{15}$$
$$\sum_{j=1}^{J}\sum_{k=1}^{K_j} c_{jki} = 1 \qquad \forall i \tag{16}$$
$$\sum_{i'\in T} c_{jki'} \geq 2c_{jki} \qquad \forall j\forall k\forall i \in T \tag{17}$$
$$\sum_{i=1}^{I} c_{jki} r_i \leq r_{jk} \qquad \forall j\forall k \tag{18}$$
$$\sum_{k=1}^{K_j}\sum_{i=1}^{I} c_{jki} w_i \leq W_j \qquad \forall j \tag{19}$$
$$\sum_{k=1}^{K_j}\sum_{i=1}^{I}(\frac{1}{2}(c_{jki} h_i s_i) + c_{jki} h_i l_i) \leq H_j \qquad \forall j \tag{20}$$
$$\sum_{k'=1}^{k-1}\sum_{d'=2}^{d-1}\sum_{i=1}^{I} a_{id'} c_{jk'i} - 2(k-1)\delta_{jkd} \leq 0 \qquad \forall j\forall k\forall d \tag{21}$$
$$a_{id} c_{jki} + \delta_{jkd} - O_{jk} \leq 1 \qquad \forall j\forall k\forall d\forall i \tag{22}$$
$$E_j - c_{jki} \geq 0 \qquad \forall j\forall k\forall i \tag{23}$$
$$P_{jd} - a_{id} c_{jki} \geq 0 \qquad \forall j\forall k\forall i\forall d \tag{24}$$

The objective function (12) is a weighted sum of the four objectives as defined in the *CSPUDL*. 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 containers stowed in slots with reefer plugs, and then subtracting the number of containers that are actually reefers.

Inequality (13) ensures that the cell below a cell stowing a 40-foot container stows either 20 or 40-foot containers, while inequality (14) ensures that a cell below a cell stowing 20-foot containers only stows 20-foot containers, since 20-foot containers can not be stowed on top of 40-foot containers (2). Inequality (15) requires that all cells stow either two 20-foot or one 40-foot container. The fact that a container must be stowed in just one cell is modeled by (16). Inequality (17) forces the number of 20-foot containers in a cell stowing a 20-foot container to be greater or equal to two, since the two sides of a stack must be synchronized (7). The reefer capacity of a cell is constrained by inequality (18), covering the fact that all reefer containers in a cell must be provided with a reefer plug each (3). The weight and height limits of stacks (5) are ensured by (19) and (20), respectively. Inequalities (21) and (22) model the overstowage objective, and inequalities (23) and (24) model the empty stack and clustering objective.

3.0.1 Cuts

We focus on removing non-integer solutions allowed by the model by modifying inequality (21) used to define variable δ_{jkd}. The model becomes stronger when this constraint is decomposed such that its semantics applies for each term of the constraint

$$a_{id'}c_{jk'i} \leq \delta_{jkd} \quad \forall j \, \forall k \, \forall d \, \forall i \, \forall k' \, \forall d' \tag{25}$$

where $k' \in \{1, ..., k-1\}$ and $d' \in \{2, ..., d-1\}$. We then introduce cut (26) and (27) that sum over all containers instead of only considering one container at the time, extending the number of left hand side terms from (25) and making the cut stronger. Two constraints are considered since two 20-foot containers can be stowed in a cell. Cut (28) adds terms to the left hand side of (25) by summing over all cells below a cell k instead of over all containers. The cuts are defined by:

$$\frac{1}{2}\sum_{i \in T'} c_{jk'i} \leq \delta_{jkd} \qquad \forall j \forall k \forall d \forall k' \tag{26}$$

$$\sum_{i \in F'} c_{jk'i} \leq \delta_{jkd} \qquad \forall j \forall k \forall d \forall k' \tag{27}$$

$$\sum_{k'=1}^{k-1} a_{id'}c_{jk'i} \leq \delta_{jkd} \qquad \forall j \forall k \forall i \forall d \forall d' \tag{28}$$

where $k' \in \{2, ..., d-1\}$, $d' \in 2, ..., d-1$, and T' and F' are the set of 20 and 40-foot containers with discharge port earlier than d, respectively.

4 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 in 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 extending 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.

In order to find optimal solutions to a CSP, a cost function is defined to evaluate solutions. A *branch and bound* approach is followed, where every time a new solution is found a constraint is posted for the remaining part of the search space such that new solutions always have lower cost values than the previous ones.

Constraints in CP share information through the variables in X. Each constraint has a scope $X' \subset X$, relatively small compare to the size of X, limiting its reasoning 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 y be an integer variable, z a variable with finite domain, and c an array of variables or constants, i.e., $c = [x_1, ..., x_n]$. The *element constraint* Hentenryck and Carrillon [1988] states that z is equal to the y-th variable or constant in c, or $z = x_y$.

$$\begin{aligned} &element(y, z, c) = \\ &\qquad \{(e, f, d_1, ..., d_n) \,|\, e \in D(y), f \in D(z), \forall i . d_i \in D(x_i), f = d_e\} \end{aligned} \tag{29}$$

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton or a regular expression recognizing the strings in the language $L(M)$, and let $X = \{x_1, ..., x_n\}$ be a set of variables with $D(x_i) \subseteq \Sigma$ for $1 \leq i \leq n$. Then the *regular constraint* Pesant [2004] is defined as

$$regular(X, M) = \{(d_1, ..., d_n) \,|\, \forall i.d_i \in D(x_i), d_1...d_n \in L(M)\}. \tag{30}$$

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

$$exactly(N, X, V) = \{(d_1, ..., d_m) | \forall i.d_i \in D(x_i), |\{d_i | d_i = V\}| = N\} \tag{31}$$

Let $X = \{x_1, ..., x_n\}$ and $Y = \{y_1, ..., y_n\}$ be two sets of finite domain variables with domains $D(X) = D(Y) = \{1, ..., n\}$. 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. More specifically

$$\begin{aligned} channeling(X, Y) = \{(dx_1, ..., dx_n, dy_1, ..., dy_n) | \\ \forall i, j \,.\, dx_i \in D(x_i), dy_j \in D(y_j), dx_i = j \Leftrightarrow dy_j = i\}. \end{aligned} \tag{32}$$

A channeling constraint is used to increase the reasoning power of the model by using several isomorphic variable sets (also known as *viewpoints* Smith [2006]). If X is variables representing positions with boxes $\{1, \ldots, n\}$ as domains and Y is variables representing boxes with positions $\{1, \ldots, n\}$ as domains then clearly a channeling constraint will link them consistently together. In particular, notice that the channeling constraint embeds the *alldifferent constraint* that in our example ensures that a position only can hold one box and vice versa.

5 The CP Model

Table 3 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. According to this, the aft and fore slots at the bottom of the left most stack will be assigned to ids 1 and 2, respectively, and the aft and fore slots in the next cell of the same stack to ids 3 and 4. The aft and fore bottom slots of the next stack are then assigned to ids $k + 1$ and $k + 2$, where k is the number of slots in the left most stack. We have $Slots_k = Slots_k^F \cup Slots_k^A$. We have $POD_i < POD_j$ iff the vessel calls the discharge port of container i before the discharge port of container j. The set of *blocked slots* are slots that are unavailable due to the physical structure of the vessel like the two slots in stack 1 and tier 3 over deck in the left drawing of Figure 2. The blocked slots are defined by $Slots^B$.

In our model, the decision variables represent the stowage plan for a set of preselected containers to stow (solution). We consider two possible representations. The first one defines a decision variable for each container in $Cont$ to stow, 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.

We include the two sets of decision variables mentioned above as keystones for two different viewpoints in our CP model. To efficiently link the two viewpoints a channeling constraint (32) is posted in the combined model, such that both viewpoints contain the same information all the time. 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. First, since a 40-foot container occupies two 20-foot slots, all 40-foot containers are split into two smaller containers, $Aft40$ and $Fore40$, of the size of a slot each. All 40-foot 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, and $Cont_i^{40A}$ and $Cont_i^{40F}$ represent the 40-foot container i. *Virtual containers*, $Cont^V$, that will be stowed in slots meant to remain empty are also added, together with *blocking containers*, $Cont^B$, that will be allocated in the blocked slots of the location. In the remainder

$Stacks$	Stack index set
$Slots$	Slot index set
$Cont$	Container index set
$Slots^{\{A,F\}}$	Aft (A) and Fore (F) slots
$Slots_k$	Slots of stack k
$Slots_k^{\{A,F\}}$	Aft (A) and Fore(F) slots of stack k
$Slots^{\{R,\neg R\}}$	Reefer (R) and non-reefer ($\neg R$) slots
$Slots^{\neg RC}$	Slots in cells with no reefer plugs
$Slots^{\{B,20,40\}}$	Blocked (B), 20 (20), and 40-foot (40) capacity slots
$Cont^{\{V,B,L\}}$	Virtual (V), blocking (B), and loaded containers (L)
$Cont^{\{20,40\}}$	20 (20) and 40-foot (40) containers
$Cont^{40\{A,F\}}$	$Aft40$ (F), $Fore40$ (A) 40-foot containers
$Cont^{\{20R,40R\}}$	20-foot ($20R$) and 40-foot ($40R$) reefer containers
$Cont^{\neg R}$	Non-reefer 20 and 40-foot containers
$Weight_i$	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
$stack_k^{\{w,h\}}$	h=height, w=weight physical limit of stack k
$Classes$	Possible classifications of stacks according to their features
$class^i$	Set of stacks of class i

Table 3: Index sets and constants of the CP model.

of the paper $Cont$ will refer to this extended set of containers. Finally, $Cont^{\neg R}$ is defined as the set of non-reefer containers that are neither virtual nor blocking containers. Table 4 summarizes the variables in the CP model. In addition to the two sets of decision variables previously defined, extra sets of auxiliary variables are defined in order to facilitate the modeling of the constraints and objectives. The objectives of the CP model are given by:

$$O^v = \sum_{i \in Slots^A} ov(i) \tag{33}$$

$$O^u = \sum_{i \in Stacks} \Big((\sum_{j \in Slots_i} p_j) > 0 \Big) \tag{34}$$

$$O^p = \sum_{i \in Stacks} \Big((\sum_{\rho \in POD} (\sum_{j \in Slots_i} (p_j = \rho))) > 0 \Big) \tag{35}$$

$$O^r = \sum_{i \in Slots^R} (s_i \in Cont^{\neg R}) \tag{36}$$

$$O = 100O^v + 20O^p + 10O^u + 5O^r \tag{37}$$

Objective (33) calculates the total number of overstows. In order to do so, let $ov : Slots^A \rightarrow \{0, 1, 2\}$ denote the number of overstowing containers in a cell represented by its aft slot. We have

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

$C = \langle c_1, \ldots, c_{\lvert Cont \rvert} \rangle$	$c_i \in Slots$ is the slot index of container i
$S = \langle s_1, \ldots, s_{\lvert Slots \rvert} \rangle$	$s_j \in Conts$ is the container index of slot j
$L = \langle l_1, \ldots, l_{\lvert Slots \rvert} \rangle$	$l_j \in Length$ is the length of the container stowed in slot j
$H = \langle h_1, \ldots, h_{\lvert Slots \rvert} \rangle$	$h_j \in Height$ is the height of the container stowed in slot j
$W = \langle w_1, \ldots, w_{\lvert Slots \rvert} \rangle$	$w_j \in Weight$ is the weight of the container stowed in slot j
$P = \langle p_1, \ldots, p_{\lvert Slots \rvert} \rangle$	$p_j \in POD$ is the POD of the container stowed in slot j
$HS = \langle hs_1, \ldots, hs_{\lvert Stacks \rvert} \rangle$	$hs_k \in \{0, ..., stack_k^h\}$ is the current height of stack k
$Lengths_k^{\{A,F\}}$	Aft (A) and Fore (F) variables of L for stack k
$C^V \in C$	Virtual containers
$S_k^E \in S$	Slots with the same features in stack k
$D_k \in \{0, ..., \lvert POD \rvert\}$	Number of different discharge ports in stack k
$O^v \in \{0, ..., \lvert Conts \rvert\}$	Number of overstowing containers
$O^u \in \{1, ..., \lvert Stacks \rvert\}$	Number of used stacks
$O^p \in \{1, ..., \lvert Stacks \rvert * \lvert POD \rvert\}$	Total number of different discharge ports in each stack
$O^r \in \{0, ..., \lvert Slots^R \rvert\}$	Number of non-reefer containers stowed in reefer cells
$O \in \{0, ..., \infty\}$	Solution cost variable

Table 4: Variables of the CP model.

where $be : Slots \rightarrow 2^{Slots}$ associates a slot with the set of slots in the same stack that are below it, $minP : 2^{Slots} \rightarrow POD$ is the earliest discharge port among the containers allocated in a set of slots, and $\oplus$ denotes the exclusive or Boolean operator. The empty stack objective (10), is represented by (34). The smallest discharge port index, 0, is assigned to virtual and blocking 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 (35) calculates the number of different discharge ports of containers stowed in each stack and objective (36) counts the number of non-reefer containers stowed in reefer slots. Objective (37) defines the cost function of the $CSPUDL$. The branch and bound algorithm applied to solve this problem constrains the cost variable O of the next solution to be lower than the cost of the solution with lowest cost found so far. The constraints of the CP model are given by:

$$channeling(C, S) \tag{38}$$

$$c_j = c_k + 1, \quad \forall i \in \{1, ..., |Cont^{40F}|\},\ j = Cont_i^{40F},\ k = Cont_i^{40A} \tag{39}$$

$$element(s_i, t_i, Length), \forall i \in Slots \tag{40}$$

$$element(s_i, h_i, Height), \forall i \in Slots \tag{41}$$

$$element(s_i, w_i, Weight), \forall i \in Slots \tag{42}$$

$$element(s_i, p_i, POD), \forall i \in Slots \tag{43}$$

$$s_i \in Cont^B, \forall i \in Slots^B \tag{44}$$

$$s_{pos(j)} = j, \forall j \in Cont^L \tag{45}$$

$$regular(Length_i^{\pi}, R), \pi \in \{A, F\}, \forall i \in Stacks \tag{46}$$

$$s_i \notin Cont^{20R},\ \ \forall i \in Slots^{\neg R} \tag{47}$$

$$s_i \notin Cont^{40R},\ \ \forall i \in Slots^{\neg RC} \tag{48}$$

$$s_i \in Cont^{20},\ \ \forall i \in Slots^{20} \tag{49}$$

$$s_i \in Cont^{40},\ \ \forall i \in Slots^{40} \tag{50}$$

$$\sum_{j \in Slots_i^{\pi}} h_j \leq hs_i,\ \ \pi \in \{A, F\}, \forall i \in Stacks \tag{51}$$

$$\sum_{j \in Slots_i} w_j \leq stack_i^w,\ \ \forall i \in Stacks \tag{52}$$

Constraint (38) connects the two viewpoints such that both sets of variables C and S have the same level of information all the time. Constraint (39) guarantees that each $Aft40$ and $Fore40$ container representing the same 40-foot container are stowed in the same cell. Element constraints (29) are posted to bind all auxiliary variables introduced to the model to a viewpoint. Constraints (40) , (41), (42), and (43) bind each slot variable to the auxiliary variables representing the length, height, weight and discharge port of the container stowed in such slot, i. e., in the case of constraint (40), the element constraints represent $Length_{s_i} = l_i, \forall i \in Slots$. *Blocked* slots are restricted to stow just *Blocked* containers by constraint (44), and *Loaded* containers are stowed in their pre-defined slots by constraint (45), where $pos : Cont^L \rightarrow Slots$ is a function that associates loaded containers with the slots they occupy. The valid patterns that containers stowed in stacks must follow according to their length are defined by (1) and (2). After assigning a length of -1 and 0 to blocked and virtual containers, respectively, we define a regular expression $R = -1^*20^*40^*0^*$ that recognizes all the valid patterns length wise according to these two constraints. Constraint (46) introduces a regular constraint (30) for each aft and fore stack in order to restrict their stacking patterns to follow those defined by R. Constraints (47) and (48) model the reefer constraint (3). Constraint (47) removes 20-foot reefer containers from the domain of non-reefer slots, whilst for the 40-foot reefer containers, constraint (48) removes 40-foot reefer containers from cells where neither aft nor fore slots are reefer slots. Constraints (49) and (50) restrict the domains of slots that just have 20 or 40-foot container capacity to be within the set of 20 and 40-foot containers, respectively. The height limit of each stack in the location is constrained by (51). All containers stowed in each side of a stack must be less or equal to the variable representing the height limit of the stack[1]. Constraint (52) restricts the weight of all containers stowed in a stack to be within the limits.

5.1 Symmetry-breaking and implied constraints

We introduce a set of constraints to the CP model that aim at reducing the size of the search space of the problem. These constraints do not represent any new features of the *CSPUDL*, but instead they improve the reasoning power of

[1]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 5.3.

the solver by exploiting the different combinatorial structures of the model and making explicit restrictions introduced to the problem also in our model.

$$exactly(\mathcal{V}, 0, |Cont^V| + |Cont^B|), \ \forall \mathcal{V} \in \{P, W, H\} \tag{53}$$

$$exactly(P, p, Cont^{P=p}), \ \forall p \in POD \tag{54}$$

$$exactly(W, w, Cont^{W=w}), \ \forall w \in Weights \tag{55}$$

$$exactly(H, H^\alpha, Cont^\alpha), \ \forall \alpha \in \{N, HC\} \tag{56}$$

$$h_j = h_k, \ \forall i \in Stacks, \ \forall_{j,k}.\ j \in Slots_i^A \wedge k \in Slots_i^F \wedge same(j,k) \tag{57}$$

$$sort(C^V) \tag{58}$$

$$s_i \notin Cont^{40A}, \ \forall i \in Slot^F \tag{59}$$

$$s_i \notin Cont^{40F}, \ \forall i \in Slot^A \tag{60}$$

$$s_j \leq s_k, \ \forall i \in Stacks \ , \forall_{j,k}.\ j \in Slots_i^A \wedge k \in Slots_i^F \wedge equal(j,k) \tag{61}$$

$$sort(S_i^E), \ \forall i \in Stacks \tag{62}$$

$$lex(class^i), \ \forall i \in Classes \tag{63}$$

Constraints (53), (54), (55), and (56) 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 set of variables due to a lack of global perspective by the element constraints. To improve this, we first assign to the weight, height, and discharge port of virtual and blocking containers the value zero. Since these containers are not suppose to affect total height, weight, or overstowage of each stack. Then, constraint (53) limits the number of variables set to zero from P, W, and H to be the exact sum of blocked and virtual containers. Additionally, constraints (54), (55), and (56) 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. Since a cell must be either empty or with its two slots occupied (7), and there are no 20-foot high-cube containers according to the definition of our problem, constraint (57) is posted to limit the height of two slots in the same cell to have the same height. A function $same : Slots \times Slots \rightarrow \{true, false\}$ that associates pairs of slots to the Boolean value true when they belong to the same cell is defined.

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. First, since all virtual containers have the same features, it is not relevant where each container is stowed. Constraint (58) posts a sorting constraint over the virtual containers, forcing the slots where these containers will be stowed to follow a non-decreasing order, removing symmetrical solutions generated by swapping them. Second, splitting up all 40-foot containers into two smaller containers $Aft40$ and $Fore40$ also generates symmetrical solutions that are broken by constraint (59) and (60). Third, constraint (61) limits the possibility of swapping containers between two slots of a cell that have the same features. A function $equal : Slots \times Slots \rightarrow \{true, false\}$ similar to function $same$ defined above associates pairs of slots to the Boolean value true when they 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 (62) sorts in a non-decreasing order the indices of the containers stowed in slots with the same feature for each stack of the location. It is necessary, however, to assign indices to containers in a fixed way in order to avoid conflicts between this constraint and the one that constrains stacking patterns (46). 20-foot containers will be assigned a lower index than 40-foot containers, and virtual containers will have the highest index possible. Finally, the possible symmetries between stacks with identical characteristics are considered. Stacks are classified according to their features: slot capacity, reefer capacity, height and weight limit. Constraint (63) avoids symmetrical solutions generated by the containers stowed in similar stacks being swapped with each other by requiring a lexicographical ordering on the container indices of the containers stowed in these stacks. This constraint works as follows: let $Stack_1$=$\{s_1, s_2, s_3\}$ and $Stack_2$ = $\{s_4, s_5, s_6\}$ be two stacks with the same features. A complete assignment A_1 =

$\{s_1 = 1, s_2 = 2, s_3 = 3, s_4 = 4, s_5 = 5, s_6 = 6\}$ is symmetrical to $A_2 = \{s_1 = 4, s_2 = 5, s_3 = 6, s_4 = 1, s_5 = 2, s_6 = 3\}$, since A_2 is generated by swapping containers from slots at the same tier level in $Stack_1$ and $Stack_2$. The lexical order constraint between stacks $Stack_1$ and $Stack_2$ will rule out assignment A_2, since $(4, 5, 6) \not<^d (1, 2, 3)$.

5.2 Branching strategies

Our branching 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. To do so, 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. In the case of the first sub-branching, since three of the four objectives of the $CSPUDL$ rely on the discharge port of the containers stowed in the slots of the location, we start by branching over the set of discharge port variables P and prefer to assign slots with a container that favors the clustering and overstowage objectives among the first free slots bottom-up of all stacks. First we determine the discharge ports of the containers already stowed in each stack. Then, when possible, we select a slot with a container in its domain that has a discharge port that has been previously used in the same stack as the slot, in order to avoid increasing the pure stack and overstowage objectives. If it is not possible to find such slot, a slot with a container in its domain having a discharge port less or equal to the one stowed in the slot right beneath is selected, reducing the probability of overstowage. The slots from stacks already used are considered first to reduce the used stack objective. When it is necessary to select a slot from an empty stack, the farthest discharge port possible for the slot is selected. The stacks are considered in a non-increasing order according to their available slots. 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, stowing a container with height h into the first slot bottom-up in the stack. For W we follow the same approach as with H, but 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 after the discharge port, height, and weight of the containers to be stowed in each slot have been decided. Slots from stacks are selected bottom-up, choosing the container with the smallest index from the domain of the variables. The domain size of variables in P are considerably smaller than that of 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 better objective value. Therefore, a large part of the search process concentrates on a much smaller sub-problem. It only branches over the remaining variables when a solution with a better objective value is likely to be found. Additionally, this decomposition of the branching allows us to introduce a new symmetry-breaking constraint. After the first branching has finished discharge ports have been assigned to all slots in the location. It is possible then to generate symmetrical solutions by swapping containers with the same discharge port within the same stack stowed in slots with the same features. A sorting constraint over each sub-set of slots within a stack with the same features and discharge port is posted to break this symmetry.

5.3 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 rule.

Overstowage. To calculate a lower bound on the overstowage of a partial solution ρ, we define a new function $min\overline{P}(X) = min_{i \in X}(\overline{p}_i | p_i \in P)$ that considers the upper bound $\overline{p}_i$ of each variable $p_i \in P$. We then define a lower bound $ov_\rho(i)$ of $ov(i)$ for any completion of ρ as

$$ov_\rho(i) = \begin{cases} 2 & \text{if } s_i \in Cont^{20} \wedge \underline{p}_i > min\overline{P}(be(i)) \wedge \underline{p}_{i+1} > min\overline{P}(be(i)) \\ 1 & \text{if } s_i \in Cont^{40} \wedge \underline{p}_i > min\overline{P}(be(i)) \vee \\ & \quad s_i \in Cont^{20} \wedge (\underline{p}_i > min\overline{P}(be(i)) \oplus \underline{p}_{i+1} > min\overline{P}(be(i))) \\ 0 & \text{otherwise} \end{cases}$$

Proposition 5.1. *$ov(i) \geq ov_\rho(i)$ for any completion of ρ.*

Proof. Assume by contradiction $ov_\rho(i) = 2$ for some cell i, but there exists a completion of ρ where $ov(i) = 1$. Since $ov_\rho(i) = 2$ we have that $\underline{p}_i > min\overline{P}(be(i)) \wedge \underline{p}_{i+1} > min\overline{P}(be(i))$ but this implies that $p_i > minP(be(i)) \wedge p_{i+1} > minP(be(i))$ for any completion of ρ which means that $ov(i) = 2$ for any completion of ρ which is impossible. The remaining cases can be shown in a similar fashion. □

The pruning effect of the lower bound is achieved by adding the constraint $O^v \geq \sum_{i \in Slots^A} ov_\rho(i)$. 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 can enforce that containers below a cell that has been identified by the lower bound to be non-overstowing actually are so

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

This rule plays an important role in situations where the value of O^v has been determine by some constraint other than the overstowage constraint, e.g., a linear constraint that calculates the total objective value of a stowage plan.

Empty stack. In the remainder, we call a container i *unstowed* when the domain of c_i has not been reduced to a single slot yet $|D(c_i)| > 1$. For the empty stack lower bound, a relaxation of the stowage problem is solved, where the height capacity of the stacks is the only constraint considered and all unstowed and non-virtual containers, $Cont_\rho^N = \{i | i \in Cont, |D(c_i)| > 1, i \notin Cont^V\}$, are accounted as normal height containers. We first consider the used stacks of ρ where there are containers already stowed, defined as $Stacks_\rho^U = \{i \in Stacks \,|\, \exists j \in Stacks_i, |D(s_j)| = 1, s_j \notin Cont^B \cup Cont^V\}$. The lower bound procedure stows as many containers as possible from $Cont_p^N$ in $Stacks_\rho^U$, such that the height capacity constraint is fulfilled. Once the used stacks are completely filled up, the empty stacks $Stacks_\rho^E = Stacks \setminus Stacks_\rho^U$ are sorted in decreasing order by height capacity and filled up following this order with the remaining containers of $Cont_\rho^N$. The number of used stacks L_ρ^u is then the sum of used stacks $|Stacks_\rho^U|$ plus the empty stacks necessary to stow all remaining containers in $Cont_p^N$. L_ρ^u is a lower bound of the number of used stacks of any completion of ρ since the approach to solve the relaxed problem clearly is optimal. Since the height of all containers in $Cont_p^N$ have been reduced to the normal height, the order of stowing is irrelevant and leads to the largest number of additional containers stowed in used stacks in ρ. Since the empty stacks are filled in order of largest capacity first and the order of allocation again is irrelevant, the fewest possible number of empty stacks are being used. The pruning effect of the lower bound is achieved by adding the constraint $O^u \geq L_\rho^u$.

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 $Q_i = |\{s \in Stacks \,|\, \exists j \in Stacks_s \,.\, p_j = i\}|$ 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 $Cont_\rho^{N,P=i}$ be the set of unstowed containers in ρ with discharge port i, $Stacks_\rho^{P=i}$ be the set of stacks stowing at least one container with discharge port i, and $Stacks_\rho^{\neg P=i} = Stacks \setminus Stacks_\rho^{P=i}$ be the set of stacks where no container with discharge port i is allocated. Our goal is to generate a lower bound, $L_\rho^p(i)$ independently for each Q_i, based on the approach followed to generate lower bounds for the used stacks objective. To compute $L_\rho^p(i)$ for some $i \in POD$ and partial solution ρ, we solve a relaxed allocation problem where only containers in $Cont_\rho^{N,P=i}$ are stowed. When a container from $Cont_\rho^{N,P=i}$ is stowed in a stack $Stacks^{P=i}$ no penalty is paid. On the contrary, if such container is stowed in a stack $Stacks^{\neg P=i}$, a penalty must be paid. This situation resembles the one from the used stacks objective. We have a set of stacks where containers can be allocated without paying any penalty and a set of stacks where the containers must pay a penalty for being allocated. Our lower bound aims at using as few stacks as possible from the set of stacks $Stacks^{\neg P=i}$ where a penalty must be paid.

Thus, by treating $Stacks_\rho^{P=i}$ as $Stacks_\rho^U$, $Stacks_\rho^{\neg P=i}$ as $Stacks_\rho^V$, and $Cont_\rho^{N,P=i}$ as $Cont_\rho^N$ and following the allocation approach for the used stacks objective, we get an optimal solution to the relaxed problem such that $Q_i \geq L_\rho^p(i)$. The pruning effect of the lower bound is achieved by adding the constraint $O^p \geq \sum_{i \in POD} L_\rho^p(i)$.

Reefer A lower bound L_ρ^r for the reefer objective of a partial solution ρ can be deduced from a counting argument. Let $S_\rho^{\neg R} = |\{i \in Slots^R \,|\, D(s_i) = 1, s_i \notin Cont^R\}|$ denote the number of reefer slots stowing a non-reefer container

in ρ. Clearly, $O^r \geq S_\rho^{\neg R}$ 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 $C_\rho^R = |\{i \in (Cont^{40R} \cap Cont^{40A}) \cup Cont^{20R} \mid |D(c_i)| > 1\}|$ denote the number of unstowed reefer containers in ρ (we only count the aft part of a 40-foot container reefer to avoid counting the container twice). Further, let $S_\rho^{UR} = |\{i \in Slots^R \,|\, |D(s_i)| > 1,\, D(s_i) \cap Cont^V = \emptyset\}|$ be the reefer slots where no virtual container will be stowed. If $S_\rho^{UR} > C_\rho^R$ then at least $S_\rho^{UR} - C_\rho^R$ extra reefer slots will stow non-reefer containers. Thus, we can tighten L_ρ^r as follows

$$L_\rho^r = \begin{cases} S_\rho^{UR} - C_\rho^R + S_\rho^{\neg R} & : \text{ if } S_\rho^{UR} > C_\rho^R \\ S_\rho^{\neg R} & : \text{ otherwise} \end{cases}$$

The pruning effect of the lower bound is achieved as usual by adding the constraint $O^r \geq L_\rho^r$.

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 $h_\rho(j) = \overline{hs}_j - h_j^s$, where h_j^s denote the height of the stowed containers in stack j. Let $M_\rho^N(j)$ and $M_\rho^{HC}(j)$ denote the maximum number of normal and high-cube containers that can be placed in stack j, respectively. We then have

$$\begin{aligned} M_\rho^N(j) &= \lfloor h_\rho(j)/h(N) \rfloor, \\ M_\rho^{HC}(j) &= \lfloor h_\rho(j)/h(HC) \rfloor, \end{aligned}$$

where $h(N)$ and $h(HC)$ denote the height of normal and high-cube containers. Let C_ρ^N and C_ρ^{HC} 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

$$\sum_{j \in Stacks} M_\rho^N(j) \geq C_\rho^N \quad \wedge \quad \sum_{j \in Stacks} M_\rho^{HC}(j) \geq C_\rho^{HC}.$$

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 in a slot 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. Now we proceed to update $\underline{hs}_j$. Clear, we can apply the bin packing propagator suggested by Paul [2004]

$$\underline{hs}_j \geq \sum_{i \in Cont} Height_i - \sum_{i \in Stacks \setminus \{j\}} \overline{hs}_i, \quad \forall j \in Stacks.$$

6 Experiments

236 *CSPUDL* instances have been derived from stowage plans provided by our industrial collaborator. Each instance corresponds to restowing a random location in one of these plans. Since the plans have been applied in real life, we can assume that the distribution of instances corresponds to what any hierarchical stowage planning system has to handle well when solving the low-level problem of assigning containers to slots in locations. To investigate the impact of different features of the instances, we have partitioned them into the classes shown in Table 5. All the experiments were run on a Linux machine with two Quad Core Opteron processors at 1.7 GHz and 8 GB of memory. The CP and IP model were implemented in Gecode 3.3 Gecode Team [2006] and CPLEX 12.1, respectively.

6.1 Impact of CP enhancements

Here we analyze the impact of the different enhancements of the CP model introduced in Section 5. We define four CP models. The *basic* model includes only the constraints and objectives of the *CSPUDL* (33 - 52). A simple branching

Class	40'	20'	Reefer	HC	DSP > 1	# Inst.
1	*					13
2		*				22
3	*	*				13
4	*			*		78
5	*	*		*		36
6	*		*	*		15
7	*	*	*	*		14
8	*			*	*	14
9	*	*		*	*	16
10	*		*	*	*	8
11	*	*	*	*	*	6

Table 5: Grouping of Instances. The first column is the group index. Column 2, 3, 4, 5, and 6 define the characteristics of the group instances in terms of the presence of 40', 20', reefer, and high-cube containers and whether more than one discharge port is represented. Column 7 is the number of instances in the group.

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 5.1. Its branching strategy is similar to the *basic* model, but additionally, 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 5.2 and the lower bounds of Section 5.3, respectively.

Since *CSPUDL* instances must be solved fast, we set a runtime limit of one second. The solver can return an optimal solution before that, but after one second it must return its current solution. The results are summarised in table 6. As expected, the total number of instances solved and proven optimal increases for each extension of the basic

Class	**Basic**		**Improved**		**Branching**		**Advanced**		
	sol	opt	sol	opt	sol	opt	sol	opt	all
1	13	9	13	11	13	13	13	13	13
2	19	12	18	11	20	19	20	20	18
3	12	8	12	9	13	13	13	13	12
4	64	9	65	48	75	70	75	74	58
5	19	4	25	16	32	27	33	31	13
6	12	1	14	6	15	14	15	14	12
7	10	0	9	2	9	4	9	8	5
8	11	2	11	4	13	13	13	13	7
9	13	3	11	2	13	8	13	13	8
10	5	1	7	3	8	5	8	8	6
11	4	1	5	2	5	4	5	5	4
Total	182	50	190	114	216	190	217	212	156

Table 6: Number of instances solved and proven to optimality by the CP models. The last column is the number of instances solved by all four models.

model. A more careful inspection of the table shows that this does not apply to all classes individually, but overall the impact of the model improvements are quite similar for each class.

We use the subset of 156 instances solved by all four CP models to compare their runtime and optimality characteristics. The left graph of Figure 3, shows the runtime of the models for each instance. We have sorted the instances such that the expected runtime dominance between the models is clearly observable. This dominance is also reflected in the total runtime for the 156 instances which was 110.75, 58.98., 22.45, and 9.45 seconds for the basic, improved, branching, and advanced model, respectively. The right graph shows the optimality gap of 39 out of the 156 instances that at least one model solved suboptimally. Again, we have sorted the instances to highlight a quite robust optimality dominance between the models. An investigation of the runtime and optimality characteristics of each instance class did not show any significant difference.

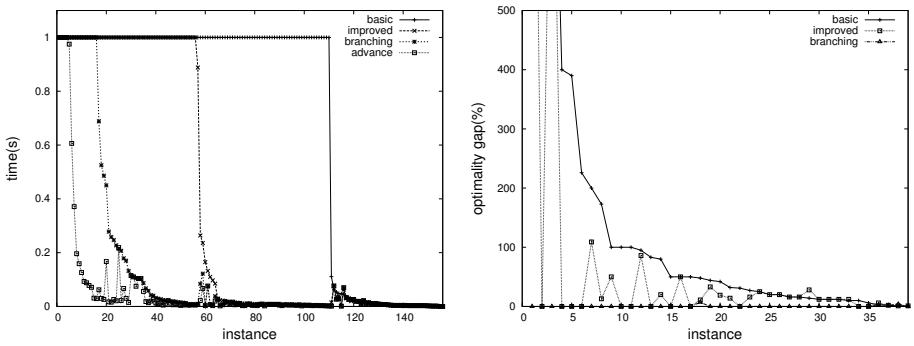

Figure 3: Runtime (left) and optimality dominance (right) of the four CP models.

6.2 Comparing the performance of our IP and CP models

In this section we compare the performance of the IP and CP model using a one second and 10 seconds runtime limit. Table 7 summarizes the performance of the models over the instance classes. For the one second time limit

Class	both	IP	CP	none
1	13	0	0	0
2	13	1	7	1
3	7	0	6	0
4	67	1	8	2
5	11	0	22	3
6	15	0	0	0
7	4	0	5	5
8	7	1	6	0
9	6	0	7	4
10	7	0	1	0
11	4	0	1	1
Total:	154	3	63	16

Class	both	IP	CP	none
1	13	0	0	0
2	19	2	1	0
3	12	0	1	0
4	76	1	1	0
5	25	3	8	0
6	15	0	0	0
7	7	2	2	3
8	10	1	3	0
9	12	1	3	1
10	8	0	0	0
11	4	0	1	1
Total:	201	10	20	5

Table 7: For both tables, column 2 is the number of instances solved by at least one of the models. Column 3 and 4 is the number of instances solved just by the IP and CP model. Column 5 is the number of instances solved by neither of the two models. The left and right table shows the results for the experiment with a one and 10 seconds runtime limit.

experiment, a total of 154 instances are solved by both models. All solutions produced by the CP model are optimal. The IP model produced 12 suboptimal solutions with optimality gap ranging from 90% to 2400%. The suboptimal instances have in common that high-cube containers are present and the number of discharge ports is greater than one. For the 10 seconds time limit experiment, the number of instances solved by both models increased considerably (47 instances). The number of suboptimal solutions was also reduced for the IP model (from 12 to 6) , but there were still five instances that remain with a high gap (from 800% to 2400%).

Figure 4 compares the response time of the two models for the two experiments. A total time of 5.2 (38.3) and 54.8 (409.6) seconds was used by the CP and IP model to solve all 154 (201) instances of the one (10) second experiment, respectively.

7 Literature review

Stowage planning for container vessels is a recognized problem in the literature, but it has not received as much attention as one would expect from its economic impact. Most approaches fall into two main categories: approaches addressing the complete problem in a single phase, and approaches decomposing the problem hierarchically into a number of sub-problems that individually can be solved using different methods.

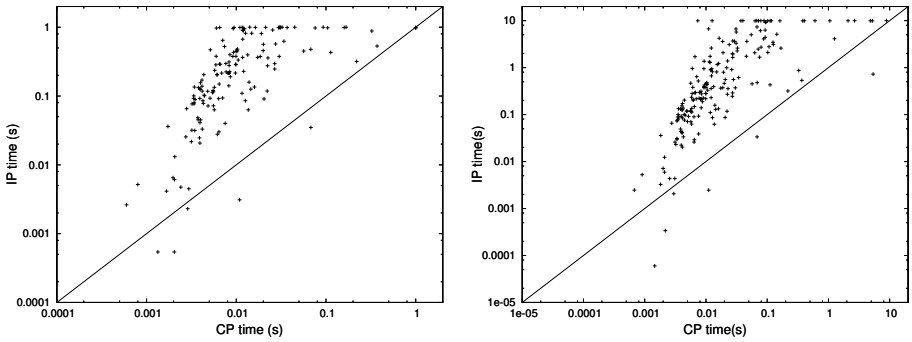

Figure 4: Runtime comparison of the CP and IP model for the experiments with one second (left) and 10 seconds (right) runtime limit.

One-phase approaches are characterized by models that introduce decision variables or similar for the assignment of each container to a slot. Initial attempts of using traditional 0-1 IP models have not been able to scale to the size of modern vessels (e.g., Botter and Brinati [1992], Avriel et al. [1998]). Heuristic approaches were then formulated. Botter and Brinati (Botter and Brinati [1992]) proposed a heuristic enumeration with the ability of generating stowage plans for a vessel of 740 TEUs, while Avriel et al. (Avriel et al. [1998]) introduced the *suspensory heuristic* that was evaluated on 300 randomly generated instances with vessel capacity ranging from 300 to 1700 TEUs. At the same time, Ambrosino and Sciomachen (Ambrosino and Sciomachen [1998]) introduced a constraint satisfaction formulation for generating stowage plans for a complete vessel. The model was formulated according to the state-of-the-art Prolog-based solution methods by the time, but only scaled to very small feeder vessels.

Attempts to generate stowage plans using one-phase approaches then moved completely towards heuristic approaches. Dubrovsky and Penn (Dubrovsky and Penn [2002]) introduced a genetic algorithm model similar to Avriel et al.'s. This approach produced stowage plans for randomly generated problems of about 1000 TEUs in 30 minutes. More recently, two heuristic approaches for generating stowage plans were introduced by Ambrosino et al. Ambrosino et al. [2010]. The first approach was a constructive heuristic that produced stowage plans following some rules extracted from the author's previous work Ambrosino et al. [2009]. The second was an ant colony optimization model that according to their experiments could stow a medium size vessel of 5632 TEUs in 139.4 seconds in average.

Initial decomposition approaches hierarchically divide the problem in two levels. At the first level, the problem of distributing containers among different sections of the ship is solved, whilst at the second level, specific slots are found for containers within each section independently, following the distribution generated by the first level. Wilson and Roach (Wilson and Roach [2000]) introduced the first model of a hierarchically decomposition solving a vessel of size 696 TEUs in approximately 90 minutes. Kang and Kim (Kang and Kim [2002]) proposed a similar decomposition approach that iteratively improved the quality of the stowage plan. According to the computational results, they could solve random instances of up to 4000 TEUs in about 11 minutes. Ambrosino et al. (Ambrosino et al. [2006], Ambrosino et al. [2009]) proposed a three-phase heuristic where the problem also was hierarchically decomposed in two levels. However, after solving the two levels of the decomposition, a third phase post-optimizes the stowage plan in order to improve stability conditions. Two vessels of 198 and 2124 TEU capacity were stowed in their experimental section in 24.5 and 74.7 seconds, respectively. Gumus et al. (Gumus et al. [2008]) introduced a four-level decomposition approach that they claimed to be scalable and modular, but no computational results were given. Finally, an automatic stowage system was introduced in Yoke et al. (Yoke et al. [2009]) where the process of generating stowage plans was consecutive rather than hierarchical and each phase considered different constraints of the problem. A vessel with 5000 TEUs capacity was used in their experiments.

Since all the work described above present approaches for generating complete stowage plans, there has not previously been published an independent model and experimental analysis of the sub-problem of assigning individual containers to slots in vessel bays. Wilson and Roach (Wilson and Roach [2000]) briefly described a tabu search algorithm for solving a version of this sub-problem that must have included reefer slots, length restrictions and also

considered minimizing overstowage and avoiding discharge port mixing of stacks. They implemented a tabu search approach and claimed that near optimal solutions could be computed fast. But they only described experimental results for generating a complete stowage plan for a single vessel. Kang and Kim (Kang and Kim [2002]) described an enumeration approach for solving a very simple version of the problem where only overstow minimization and sorting of 40-foot containers after weight was considered. As for Wilson and Roach, no independent experimental evaluation of the algorithm was provided. Ambrosino et al. (Ambrosino et al. [2009]) described a 0-1 IP model for stowing individual vessel bays optimally. The model minimized the time for stowing containers. 20 and 40-foot containers were considered and containers were sorted according to weight in each stack. The experimental section considered generating a complete stowage plan for a 198 and 2124 TEU container vessel where the biggest bay had a capacity of 20-120 TEUs. No computational time was provided for solving these sub-problems and the bays were assumed only to hold containers to a single discharge port.

8 Conclusions

In this paper we have presented the first independent study of a class of important sub-problems for hierarchically decomposed methods to stowage planning that assigns containers to slots in sections of vessel bays. We have introduced an accurate model of the problem called *CSPUDL* that has been validated by the industry. We have developed a CP and IP model to solve the *CSPUDL* optimally. The CP model works well on the practical instances considered. It is demonstrated that this CP model performs better than the (basic) IP model on these instances. 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 *CSPUDL* to include over deck locations and special containers such as out-of-gauge, pallet-wide, and containers with dangerous goods.

9 Acknowledgements

We would like to thank Associate Professor Christian Schulte and PhD student Mikael Lagerkvist from the KTH Royal Institute of Technology for their fruitful suggestions on the Constraint Programming model introduced in this report.

References

Daniela Ambrosino and Anna Sciomachen. A Constraint Satisfaction Approach for Master Bay Plans. *Maritime Engineering and Ports*, 36, 1998.

Daniela Ambrosino, Anna Sciomachen, and Elena Tanfani. Stowing a conteinership: the master bay plan problem. *Transportation Research Part A: Policy and Practice*, 38(2):81–99, 2004.

Daniela Ambrosino, Anna Sciomachen, and Elena Tanfani. A decomposition heuristics for the container ship stowage problem. *Journal of Heuristics*, 12(3), 2006.

Daniela Ambrosino, Anna Sciomachen, Davide Anghinolfi, and Massimo Paolucci. A new three-step heuristic for the master bay plan problem. *Maritime Economics and Logistics*, 11(1):98–120, March 2009.

Daniela Ambrosino, Davide Anghinolfi, Massimo Paolucci, and Anna Sciomachen. An Experimental Comparison of Different Heuristics for the Master Bay Plan Problem*. In *Experimental Algorithms*, pages 314–325, 2010.

Mordecai Avriel, Michal Penn, Naomi Shpirer, and Smadar Witteboon. Stowage planning for container ships to reduce the number of shifts. *Annals of Operations Research*, 76(55-71), 1998.

Mordecai Avriel, Michal Penn, and Naomi Shpirer. Container ship stowage problem: complexity and connection to the coloring of circle graphs. *Discrete Applied Mathematics*, 103:271–279, 2000.

R.C. Botter and M.A. Brinati. Stowage container planning: A model for getting an optimal solution. *Proceedings of the Seventh International Conference on Computer Applications in the Automation of Shipyard Operation and Ship Design*, VII(C):217–229, 1992.

Opher Dubrovsky and Gregory Levitin Michal Penn. A genetic algorithm with a compact solution encoding for the container ship stowage problem. *Journal of Heuristics*, 8(585-599), 2002.

Gecode Team. Gecode: Generic constraint development environment, 2006. Available from `http://www.gecode.org`.

Peer Giemesch and Andreas Jellinghaus. Optimization models for the containership stowage problem. 2003.

Mehmet Gumus, Philip Kaminsky, Erik Tiemroth, and Mehmet Ayik. A multi-stage decomposition heuristic for the container stowage problem. In *Proceedings of the 2008 MSOM Conference*, 2008.

P. Van Hentenryck and J. P. Carrillon. Generality vs. specificity: an experience with ai and or techniques. In *Proceedings of the National Conference on Artificial Inteligence (AAAI)*, pages 660–664. ACM press, 1988.

J.G. Kang and Y.D. Kim. Stowage Planning in Maritime Container Transportation. *Journal of the Operations Research society*, 53(4):415–426, 2002.

Shaw Paul. A constraint for bin packing. In *Proceeding of Principles and practice of constraint programming*, volume 3258 of *Lecture Notes in Computer Science*, pages 648–662. Springer, 2004.

Gilles Pesant. A regular language membership constraint for finite sequences of variables. In *Proceeding of Principles and practice of constraint programming*, volume 3258 of *Lecture Notes in Computer Science*, pages 482–495. Springer, 2004.

B. Smith. Modelling. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 11. Elsevier, 2006.

I. D. Wilson and P. Roach. Container Stowage Planning: A Methodology for Generating Computerised Solutions. *Journal of the Operational Research Society*, 51(11):248–255, 2000.

Malcolm Yoke, Hean Low, Xiantao Xiao, Fan Liu, Shell Ying Huang, Wen Jing Hsu, and Zhengping Li. An Automated Stowage Planning System for Large Containerships. In *In Proceedings of the 4th Virtual International Conference on Intelligent Production Machines and Systems*, 2009.