Constraint-Based Local Search for Container Stowage Slot Planning

Pacino Dario, and Rune Møller Jensen

***Abstract*—Due to the economical importance of stowage planning, there recently has been an increasing interest in developing optimization algorithms for this problem. We have developed 2-phase approach that in most cases can generate near optimal stowage plans within a few hundred seconds for large deep-sea vessels. This paper describes the constrained-based local search algorithm used in the second phase of this approach where individual containers are assigned to slots in each bay section. The algorithm can solve this problem in an average of 0.18 seconds per bay, corresponding to a 20 seconds runtime for the entire vessel. The algorithm has been validated on a benchmark suite of 133 industrial instances for which 86% of the instances were solved to optimality.**

***Index Terms*—container-stowage, constraint-based local search, slot planning**

I. Introduction

DUE to globalization, the world economy depends on cost efficient and reliable container shipping. A container vessel sails on a closed loop service like a city bus. A key objective is to reduce the time spent in each port. This not only saves terminal fees and fuel (bunker) due to lower speeds, but also provides buffer time to deliver a reliable service. Loading and unloading operations in a container terminal are performed by quay cranes. Since the number of quay cranes available is typically fixed by contract, the main instrument for a liner shipper to reduce the port time is to require that the terminal discharges and loads containers according to a time efficient stowage plan.

Stowage plans, however, are hard to produce in practice. First, they are made under time pressure by human stowage coordinators just hours before the vessel calls the port. Second, deep-sea vessels are large and often require thousands of container moves in a port. Third, complex interactions between low-level stacking rules and high-level stress limits and stability requirements make it difficult to minimize the makespan of cranes and, at the same time, avoid that containers block each other (*overstowage*). Finally, according to our industrial partner, runtimes of more than 10 minutes are impractical, since stowage coordinators possibly need to run several forecast scenarios.

In [1] we presented an approach to stowage planning based on a 2-phase hierarchical decomposition that in most cases is able to produce near optimal stowage plans for large deep-sea vessels within a few hundred seconds. As depicted in Figure 1, first the *multi-port master planning phase* decides how many containers of each type to stow in a set of bay sections of the vessel using an Integer Programming (IP) model. Based on this distribution, a complete stowage plan for the current port is generated in the *slot planning phase* where individual containers are assigned to specific positions on the vessel for each bay section. This method is able to generate phase-wise optimal stowage plans for 75% of the tested instances in less than 400 seconds.

In this paper, we describe the constrained-based local search algorithm that is used as a heuristic fallback of an exact constraint programming model [2], [3] to solve the slot planning phase. Since the slot planning phase consists of up to 100 slot planning instances (one for each bay section) and since master planning historically has been time consuming, we aim at solving individual slot planning instances within 1 second in order to achieve total runtimes of less than 10 minutes as needed by the industry. First the Contraint Programming (CP) model attempts finding optimal solutions within 1 second, should it fail (or should the instance be infeasible), the Constraint Based Local Search (CBLS) algorithm presented in this paper is used to find (near-)optimal solutions heuristically, and can also handle infeasibilities by rolling out containers. We have evaluated the CBLS algorithm experimentally using 133 real slot planning instances provided by our industrial partner. We compare our results with the ones from the CP model presented in [2]. Each instance is solved in an average of 0.18 seconds. Thus, if only using the CBLS algorithm for slot planning, the total time needed for the slot planning phase is less than 20 seconds in average.

The remainder of the paper is organized as follows. Section II describes the problem. Section III introduces related work. Section IV presents the approach taken. Section V and VI present the experiments and draw conclusions.

II. Background and Problem Statement

A liner shipping vessel is a ship that transports box formed containers on a fixed cyclic route. Containers typically have a width of 8 feet, and a length of either 20 or 40 feet. There exists, however, longer containers such as 45 and 50-foot. Standard containers are 8.6 feet high with the exception of some higher 40-foot containers called *high-cube* containers that are 1 foot taller. Some containers are refrigerated and require access to special power plugs. Other special types of containers are pallet-wide containers, where a standard European pallet can be stored, and IMO containers, which are used to store dangerous goods. In addition, there are out-of-gauge (OOG) containers with cargo sticking out in the top

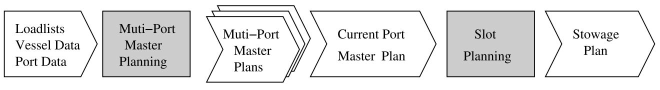

Fig. 1. Hierarchical decomposition of stowage planning into master and slot planning.

Manuscript received December 08, 2011. This research is sponsored in part by the Danish Maritime Fund under the BAYSTOW project.
D. Pacino and R. M. Jensen are with the IT-University of Copenhagen. [dpacino,rmj]@itu.dk

or at the side (e.g., a yacht) and non-containerized break-bulk like windmill wings.

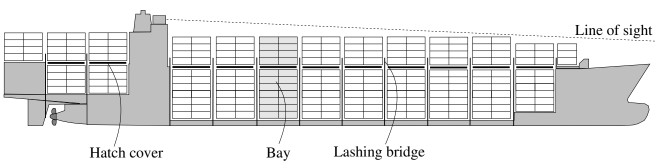

Fig. 2. The arrangement of bays in a small container vessel.

The cargo space of a vessel is composed of a number of *bays*, which are a collection of container *stacks* along the length of the ship. Each bay is divided into an *on-deck* and *below-deck* part by a *hatch cover*, which is a flat water tight structure that prevents the vessel from taking in water. An overview of a vessel layout is shown in Figure 2.

Figure 3 shows how each stack is composed of two Twenty foot Equivalent Unit (TEU) stacks and one Forty foot Equivalent Unit (FEU) stack, which hold vertically arranged *cells* indexed by *tiers*. The TEU stack cells are composed of two *slots*, which are the physical positioning of a 20-foot container. The *aft* slot refer to the position toward the stern of the vessel, while *fore* slots are allocated on the bow side. Some of the cells have access to power plugs.

The loading and unloading of containers are carried out by *quay cranes* that can access the stacks individually. Some cranes can lift two 20-foot containers at the same time, but they only have access to the container on top of the stack.

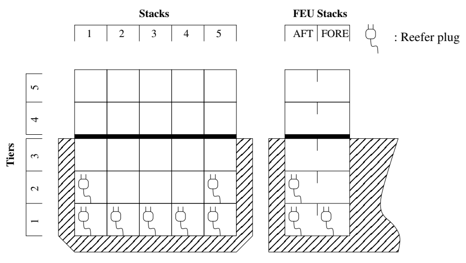

Fig. 3. A bay structure seen from behind (left) and from the side (right).

The primary objective of stowage planning is to minimize port stay. An important secondary objective is to generate stowage plans that are robust to changes in the cargo forecast. The reason is that the number of containers of each type to load in downstream ports is only fairly accurately known a few ports ahead.

When a set of containers to stow in a bay has been decided, the positioning of these containers has limited interference with containers in other bays. For this reason, it is natural to divide the constraints and objectives of the stowage planning problem into high-level inter bay constraints and objectives and low-level intra bay constraints and objectives.

High-level constraints mainly consider the stability of the vessel as defined by its trim, metacentric height, and stress moments such as shear, bending and torsion. In addition, any distribution of containers to bays must satisfy weight and volume capacity limits as well as capacity limits of the different container types. High-level objectives include the minimization of the crane makespan and of the overstowage between on and below deck.

Since we focus on slot planning in this paper, we describe the low-level constraint and obecjtives in more detail. Low-level constraints are mainly stacking rules. Containers must be stowed forming stacks, taking into consideration that two 20-foot containers cannot be stowed on top of a 40-foot container (this is due to the lack of supports in the middle of a 40-foot container). Each stack has a maximum allowed height and weight which cannot be exceeded. Each cell can have restrictions to which kind of containers it can hold. Some cells can be, for example, reserved to only 40-foot containers. Reefer containers require power, and can only be stowed in cells with access to special power plugs. Dangerous goods (IMO containers) must be stowed following predefined security patterns, while OOG containers can only be stowed where sufficient space is available. On-deck special rules also apply due to the use of lashing rods. On deck container weight must decrease with stack height. Wind also imposes special stacking patterns on deck and line of sight rules restrict the stack heights.

Low-level objectives reflect rules of thumb from stowage coordinators in order to get stowage plans that are robust to changes in forecasted demands. The objectives include maximizing the number of empty stacks, clustering of container's discharge port in stacks, minimizing the number of reefer slots used for non-reefer containers, and minimizing overstowage between containers in the same stack.

Taking all industrial constraints and objectives into account is desirable, however, it makes an academic study of the problem impractical. to that end, we have together with our industrial partner defined a representative problem where the number of constraints and objectives is reduced but the structural complexity is kept.

Here we give a formal definition of this representative problem using the IP model given in [2]. Given a set of stacks S and the sets of tiers per stacks T_s of the bay section for slot planning, we indicate a cell by a pair of indexes $s \in S$ and $t \in T_s$. Let $x_{stc} \in \{0,1\}$ be a decision variable indicating if the cells indexed by $s \in S$ and $t \in T$ contains the container $c \in C$, where C is the set of all containers to be stowed in the bay section.

$$\frac{1}{2}\sum_{c\in C^{20}} x_{s(t-1)c} + \sum_{c\in C^{40}} x_{s(t-1)c} - \sum_{c\in C^{40}} x_{stc} \geq 0 \quad \forall s \in S, t \in T_s \tag{1}$$

$$\sum_{c\in C^{20}} x_{s(t-1)c} + \sum_{c\in C^{40}} x_{stc} \leq 1 \quad \forall s \in S, t \in T_s \tag{2}$$

$$\frac{1}{2}\sum_{c\in C^{20}} x_{stc} + \sum_{c\in C^{40}} x_{stc} \leq 1 \quad \forall s \in S, t \in T_s \tag{3}$$

$$x_{stc} = 1 \quad \forall (s,t,c) \in P \tag{4}$$

$$\sum_{s\in S}\sum_{t\in T} x_{stc} = 1 \quad \forall c \in C \tag{5}$$

$$\sum_{c'\in C^{20}} x_{stc'} - 2x_{stc} \geq 0 \quad \forall s \in S, t \in T_s, c \in C^{20} \tag{6}$$

$$\sum_{c\in C} R_c x_{stc} - \mathcal{R}_{st} \leq 0 \quad \forall s \in S, t \in T_s \tag{7}$$

$$\sum_{t\in T_s}\sum_{c\in C} W_c x_{stc} \leq \mathcal{W}_s \quad \forall s \in S \tag{8}$$

$$\sum_{t\in T_s}\left(\frac{1}{2}\sum_{c\in C^{20}} H_c x_{stc} + \sum_{c\in C^{40}} H_c x_{stc}\right) \leq \mathcal{H}_s \quad \forall s \in S \tag{9}$$

$$\sum_{t'=1}^{t-1}\sum_{d'=2}^{d-1}\sum_{c\in C} A_{cd'}x_{st'c} - 2(t-1)\delta_{std} \leq 0 \qquad \forall s\in S, t\in T_s, d\in D \tag{10}$$

$$A_{cd}x_{stc} + \delta_{std} - o_c \leq 1 \quad \forall s\in S, t\in T_s, c\in C \tag{11}$$

$$e_s - x_{stc} \geq 0 \quad \forall s\in S, t\in T_s, c\in C \tag{12}$$

$$p_{sd} - A_{cd}x_{stc} \geq 0 \quad \forall s\in S, t\in T, d\in D, c\in C \tag{13}$$

Constraints (1) and (2), where $C^{20} \subseteq C$ and $C^{40} \subseteq C$ are respectively the set of 20-foot and 40-foot containers, forces containers to form valid stacks where 20-foot containers cannot be stowed on top of 40-foot onces. Constraint (3) ensures that either 20-foot or 40-foot containers can be stowed in a cell but not both at the same time. The set P of containers already onboard is composed of triples (s,t,c) indicating that container $c \in C$ is stowed in stack $s \in S$ and tier $t \in T_s$. Constraint (4) forces those containers to their preassigned cell. Cells holding 20-foot containers must be synchronized, meaning that they cannot hold only one 20-foot container. Such contraint is handled by inequality (6). Constraint (7), where $R_c \in \{0,1\}$ indicates if container $c \in C$ is a reefer container and $\mathcal{R}_{st}$ holds the reefer capacity of a cell, enforces the reefer cells capacity. Given W_c as the weight of container $c \in C$, constraint (8) limits the weight of a stack $s \in S$ to not exceed the maximum weight $\mathcal{W}_s$. Similarly the height limit is enforced by constraint (9), where H_c indicates the height of container $c \in C$ and $\mathcal{H}_s$ is the maximum height of stack $s \in S$. Constraint (10) defines the indicator variable δ_{std} which indicates if a container below the cell in $s \in S$ and $t \in T_s$ is to be unloaded before port $d \in D$, where D is the set of discharge ports. The constant $A_{cd} \in \{0,1\}$ indicates if container $c \in C$ must be discharged at port $d \in D$. Constraint (11) then uses the δ_{std} variable to count overstowage into the variable o_c. The number of non-empty stacks is stored in the variable e_s via the constraint (12). The number of discharge ports used within a stack is then calculated into the variable p_{sd} in constraint (13).

An optimal solution to slot planning minimizes the following objective:

$$100\sum_{c\in C} o_c + 20\sum_{s\in S}\sum_{d\in D} p_{sd} + 10\sum_{s\in S} e_s + 5\sum_{s\in S}\sum_{t\in T_s}\Big(\mathcal{R}_{st}\sum_{c\in C^{40}} x_{stc}(1-R_c) + \sum_{c\in C^{20}} x_{stc}(\frac{1}{2}\mathcal{R}_{st} - R_c)\Big) \tag{14}$$

which, following the priorities of the stowage coordinators, minimizes: overstowage, the number of different discharge ports in a stack, the number of non-empty stacks and the number of reefer cells used by non-reefer container.

III. Literature Survey

The number of publications on stowage planning has grown substantially within the last few years. Contributions can be divided into two main categories: single-phase and multi-phase approaches. Multi-phase approaches decompose the problem hierarchically. 2-phase [4], [5], [6], [1] and 3-phase approaches [7], [8] are currently the most successful in terms of model accuracy and scalability. Single-phase approaches represent the stowage planning problem (or parts of it) in a single optimization model. Approaches applied include IP [9], [10], [11], [12], CP [13], [3], GA [14], [15], SA [16], placement heuristics [17], 3D-packing [18], simulation [19] , and case-based methods [20].

Slot planning models and algorithms in the work above, however, either do not solve sufficiently representative versions of the problem or lack experimental evaluation. In our previous work, we have used the same representative model as in this paper. In [3] and [2] a CP approach is used to solve this model. The CP model solved with Gecode [21] was shown to greatly outperform both the IP model given in this paper solved with CPLEX and a column generation approach. As our comparison with the CP approach in this paper shows, however, these exact methods may need long time to prove optimality.

IV. Solution Approach

As mentioned in Section I, the aim of slot planning is to stow containers within a bay section (also called a location) as fast as possible. Knowing that up to 100 slot planning instances must be solved for a stowage plan and that master planning is time consuming, it easy to argue that a 1 second time limit on slot planning is adequate.

In order to reach or even improve such high performance we propose a CBLS with a simple hill-climbing search. Details about CBLS can be found in [22], however, some of the basic principles can easily be captured by the following description of the algorithm used in this work.

Hill-climbing might seem like a simplistic approach especially when the objectives of the problem are clearly non-linear. Our choice for this search method is supported by the fact that fast solutions must be found. We concentrated our effort on developing an accurate guiding heuristic, which, in combination with incremental computations is able to drive the hill-climb towards (near-)optimal solutions. The search could be probably be improved using a metaheuristic framework. Our preliminary studies, however, show that this is non-trivial due to the structure of the problem.

We model slot planning with a set of decision variables $x_{stp} \in C \cup \{\perp\}$ defining which container is to be stowed into which slot (one cells is composed of an AFT and a FORE slot). Slots are identified by a triple (s,t,p) where $s \in S$ is the stack, $t \in T_s$ is the tier and $p \in P = \{AFT, FORE\}$ is the position within the cell. Slots without stowed containers are assigned the special value $\perp$. Unlike the original IP model, which was cells based, our CBLS model is slot based. Such a model is more robust in terms of future development when slot based constraints need to be applied.

The algorithm uses a neighborhood generated by swapping containers. A swap is an exchange of some containers between a pair of cells. Formally, a swap γ is a pair of tuples $\gamma = (\langle s,t,c\rangle, \langle s',t',c'\rangle)$ where the containers c in the cell at stack s and tier t exchange position with the containers c' in the cell at stack s' and tier t'. The sets c and c' can contain at most two containers. Swaps are implemented with two functions $swap^{20}$ for exchanging position of 20-foot containers (that is one 20-foot with another or with an $\perp$ container), and $swap^{40}$ for exchanging position of 40-foot containers (with an other 40-foot, or with a mixture of 20-foot and $\perp$ containers).

The constraints of the model are defined in terms of violations. Violations are numerical evaluations of how far the current solution is from satisfying the constraint under consideration. Once a constraint has no violation it is considered satisfied, thus a solution where all constraint have no violation is a feasible solution. Given an assignment π, let x^{π}_{stp} be the value of the decision variables for assignment π. Taking constraint (1) as an example the violations for a single slot can be defined as

$$v^{\pi}_{stp} = \sum_{t'=0}^{t-1} \neg(t(x^{\pi}_{stp} \implies \neg f(x^{\pi}_{st'AFT})) \tag{15}$$

where $t(c)$ is true if $c \in C$ is a 20-foor container, $f(c)$ is true if $c \in C$ is a 40-foot container. We also represent boolean variables as $\{0, 1\}$ and allow arithmetic operations over them. The total violation for constraint (1) is thus defined as

$$\sum_{s\in S}\sum_{t\in T}\sum_{p\in P} v^{\pi}_{stp} \tag{16}$$

Violations are also used during the search as heuristic guidance, as it will soon be described in the description of the search algorithm.

Objectives are defined, in terms of violations like the constraints. A solution minimizing the objective violations is optimizing the objective. Violations for the overstowage objective (11) are, for example defined for each slot as

$$\begin{aligned} o^{\pi}_{stp} = \exists t' \in \{0, ..., t-1\}, p' \in P.& \\ dsp(x^{\pi}_{stp}) > dsp(x^{\pi}_{st'p'})& \end{aligned} \tag{17}$$

where $dsp(c)$ indicates the discharge port of container c. The total numer of overstowing containers thus is

$$\sum_{s\in S}\sum_{t\in T}\sum_{p\in P} o^{\pi}_{stp} \tag{18}$$

Other constraints and objectives are defined in a similar fashion. A comprehensive definition is found in [23].

A. A 3-Phase approach

Constraint satisfaction and objective optimization, tend to drive the search in opposite directions, ultimately generating a poor heuristic. For this reason we decided to split the main search into a feasibility and optimality phase. We start with a greedy initial solution, generated by relaxing the stack height and weight constraint. Containers are then stowed using the lexicographical order (reefers $\prec$ discharge port $\prec$ 20-foot container), designed to optimize the objective function. Containers that cannot be stowed, due to the constraints, are then sequentially stowed at the end of the procedure. This, *placement heuristic*, produces high quality solution that are slightly infeasible. Such solutions become the input of the *feasibility phase* where only the violation of the constraints are minimized. Feasible solutions are then passed to the *optimality phase* which will now only consider feasible neighborhoods and optimize the objective functions. Both the feasibility and optimality phases use a *min/max heuristic* where swaps are chosen by selecting a slot with the maximum number of violations and swapping it with a slots producing the minimum violations (in other words, the most improving swap). This is how violations become the essential building blocks of the search heuristic.

Algorithm 1 shows the details of the search. The placement heuristic is used in line 1 to generate and initial solution π. Lines 2-6 are the implementation of the feasibility phase.

Algorithm 1: SolveLocation()

```
1  $\pi$ = placementHeuristic();  /* Heuristic Placement */
2  while $\neg$satisfy(constraints) do
3      $\pi' \leftarrow \pi$;                    /* Feasibility phase */
4      select $s_1 \leftarrow$most violated slot do
5          select $s_2 \leftarrow$most improving slot to swap do
6              $\pi \leftarrow swap(s_1, s_2)$;
7          if $\pi'$=$\pi$ then perform side move
8  while there is objective improvement do
9      $\pi' \leftarrow \pi$;                    /* Optimality phase */
10     select $s_1 \leftarrow$most objective violating slot do
11         select $s_2 \leftarrow$most objective improving feasible slot to swap do
12             $\pi \leftarrow swap(s_1, s_2)$;
13         if $\pi' = \pi$ then $\pi$ =perform tie-breaking swap on $\pi$
14 return $\pi$;
```

While the constraints are not satisfied (line 2), we perform swaps (line 6) selecting the most violated slot (line 4) and the most improving slot (line 5). If the swap generates a non improving (but not worse) solution, we accept the solution as a side move. Side moves are allowed only after a number of failed improving tentatives.

Lines 8-13 are the implementation of the optimality phase. With π now being a feasible solution, we perform feasible swaps (line 12) so long as an objective improvement can be generated. Similarly to the feasibility phase, lines 10-11 select the most violating and most improving slots. The selection, however, is limited to feasible swaps (swaps that do not generate any violations on the constrains). Should the swap generate a non-improving solution, a limited number of side moves are allowed where solutions are chosen using tie-breaking rules (line 13). For the overstowage objective (11) the tie-breaking is defined as the number of containers overstowed by each container, which will make the algorithm choose a container that overstows many containers over one that only overstows a single container. For the non-empty stack objective (12) the tie is broken by the number of containers in the stack, so that the search rather swaps containers that are in almost empty stacks. The clustering objective (13) calculates the tie-breaker by summing the quadratic product of the number of containers with the same discharge port, which will favor those stacks that have the most containers with the same discharge port.

B. Incremental computations

In order to efficiently compute and evaluate the neighborhoods, we designed incremental calculations for each constraint and objective. Incremental calculations (or delta evaluations) allow the algorithm to evaluate and apply swaps efficiently without having to recompute all of the constraint and objective violations. A delta evaluation is based on a simple construction/destruction principle, where the violations of the original assignment are first removed from the total violation, to then be reinserted according to the new assignment. A delta evaluation can be made volatile, thus only be used as an evaluation, or persistent if the change has to be applied to the solution.

Consider a single 20-foot to 20-foot swap $\gamma = (\langle s, t, c\rangle, \langle s', t', c'\rangle)$ between two distinct stacks. Starting from stack s we first remove the known violations of

TABLE I
Test Set Characteristics. THE FIRST COLUMN IS AN INSTANCE CLASS ID. COLUMN 2, 3, 4, AND 5 INDICATE WHETHER 40-FOOT, 20-FOOT, REEFER, AND HIGH-CUBE CONTAINERS ARE PRESENT. COLUMN 6 INDICATES WHETHER MORE THAN ONE DISCHARGE PORT IS PRESENT. FINALLY, COLUMN 7 IS THE NUMBER OF INSTANCES OF THE CLASS.

Class	**40'**	**20'**	**Reefer**	**HC**	**DSP>1**	**Inst.**
1	√					6
2		√				18
3	√	√				4
4	√			√		42
5	√	√		√		27
6	√		√	√		8
7	√	√	√	√		7
8	√			√	√	7
9	√	√		√	√	10
10	√		√	√	√	2
11	√	√	√	√	√	2

container c. Then we look at all the containers stowed above it, removing one violations for each of the ones that only overstow container c. Now all the violations connected to container c are removed. Swapping c with c' we now have to do a similar but opposite operation where we add 1 violation to all container that only overstow c' in stack s at tier t and 1 violation if c' is overstowing any container below it. The same operation is then performed on the other stack s' where c and c' exchange roles. It is easy to show that given the correct data structures such operations can be performed linearly with the size of the tiers under consideration. More details and formal definitions for all constraints and objectives as well as incremental evaluations can be found in [23].

V. COMPUTATIONAL RESULTS

The algorithm has been implemented in C++ and all experiments have been conducted on a Linux system with 8 Gb RAM and 2 Opteron Quad Core CPUs, each running on 1.7 GHz and having 2Mb of cache. The slot planning instances have been derived from real stowage plans used by our industrial collaborator for deep-sea vessels.We use test data set of 133 instances, including locations between 6 TEUs and 220 TEUs. Table I gives a summarized overview of these instances. Notice that many instances only have 1 discharge port, this is a feature of the instances generated after a multi-port master planning phase, thus it reflects the actual problem we are facing.

The experimental results on the test data set, shown in Table II(a), show the percentage of solutions solved within a specific optimality gap (optimal solutions have been generated using the constraint programming algorithm described in [2]). It is easy to see that only few instances diverge from near optimality and that in 86% of the cases the algorithm actually reached the optimal solution. Studying the algorithm performance closer, we were able to gain some insight on the quality of the different phases. The heuristic placement does not take the height and weight constraint into account, leaving space for improvements. However Table II (c) and (d) show that the solution found by the heuristic placement is not far from being feasible in most of the cases. This is supported by the fact that often feasibility is reached within 20 iterations and that 61% of the time, the objective value is not compromised. The quality of the objective value of the first feasible solution is also optimal in 74% of the

TABLE II
Algorithm Analysis (A) COST GAP BETWEEN RETURNED SOLUTION AND OPTIMAL SOLUTION. (B) COST GAP BETWEEN FIRST FEASIBLE SOLUTION AND OPTIMAL SOLUTION. (C) NUMBER OF ITERATIONS NEEDED TO FIND THE FIRST FEASIBLE SOLUTION. (D) WORSENING OF THE COST OF THE HEURISTIC PLACEMENT WHEN SEARCHING FOR THE FIRST FEASIBLE SOLUTION.

Opt. Gap		**Opt. Gap (Feas.)**		**Feas. Iter.**		**Feas. Worse**	
Gap.	Freq.	Gap.	Freq.	It. ≤	%	Obj %	%
0%	86%	0%	74%	0	29%	-20%	2%
1%	2%	5%	6%	5	23%	-10%	2%
2%	2%	10%	2%	10	18%	-5%	4%
3%	2%	15%	5%	15	8%	0%	61%
4%	1%	20%	3%	20	6%	5%	8%
10%	1%	25%	3%	25	3%	10%	2%
15%	4%	35%	3%	30	6%	20%	7%
20%	1%	40%	1%	35	1%	> 20%	15%
30%	2%	> 100%	3%	40	2%		
				45	2%		
				50	1%		
				65	2%		
(a)		(b)		(c)		(d)	

cases (Table II(b)), suggesting that the heuristic placement procedure is performing well. The results also point to the fact that the optimality phase improves only a limited number of instances, which however is important especially in the cases where the optimality gap after the feasibility phase is more than 20%.

The limited improvement of solutions in the optimality phase probably happens because the search does not allow for a large degree of diversification. Preliminary tests using tabu search have shown that local diversification often was unsuccessful to escape a local minimum due to large structural differences between the local minimum and an optimal solution. A possible approach to solve this problem could be to less closely follow the heuristic in the initial placement and the search procedures. This method, however, would probably be more expensive in terms of runtime performance.

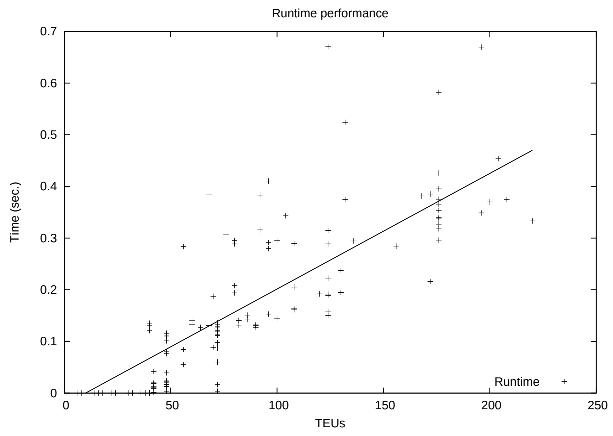

Fig. 4. Execution time as a function of instance size measured in TEUs.

The average runtime of the instances is 0.18 seconds, with a worst case of 0.65 seconds. Figure 4 shows the runtime of the algorithm as a function of the size of the instance measured in TEUs. As depicted, the execution time scales well with the instance size. Figure 5 compares the execution time between our algorithm and the complete constraint programming approach used for generating optimal solutions. When generating the instance set for investigating the optimality gap shown in table II, we excluded instances that

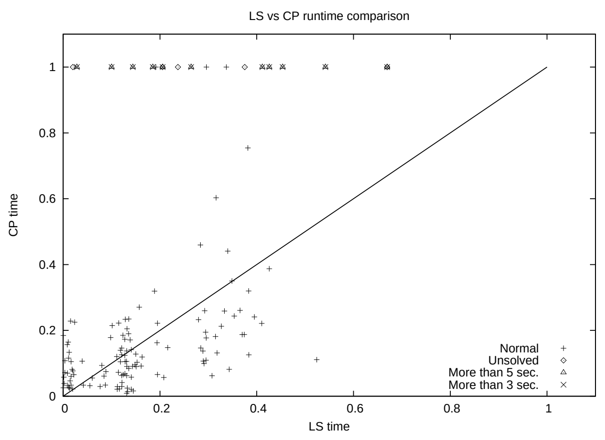

Fig. 5. Execution time comparison between our algorithm and the complete constraint programming approach used to generate optimal solutions.

were not solvable by the CP approach within 160 seconds. However, in a comparison between the two approaches these instances are particularly interesting.

As depicted, the CP approach is highly competitive within the set of instances that it can solve in 160 seconds. However our approach can solve the problematic instances for CP fast as well. This result indicates that the under-constrained nature of the problem might force exact methods to spend an excessive amount of time proving optimality. A pragmatic solution would be to execute the two approaches in parallel and rely on the optimal CP solution for the locations where it can be generated fast.

VI. Conclusion

We have developed an accurate slot planning model together with a major liner shipping company and implemented a constraint-based local search algorithm to solve them. Our experimental results show that these problems in practice are easy even though the general problem is NP-Complete. Our work contributes to our current understanding of stowage planning. It shows that if these problems are combinatorially hard in practice, the main challenge is to distribute containers to locations in vessel bays rather than assigning them to individual slots in these bays.

References

[1] D. Pacino, A. Delgado, R. M. Jensen, and T. Bebbington, "Fast generation of near-optimal plans for eco-efficient stowage of large container vessels," in *ICCL*, 2011, pp. 286–301.

[2] A. Delgado, R. M. Jensen, K. Janstrup, T. H. Rose, and K. H. Andersen, "A constraint programming model for fast optimal stowage of container vessel bays," *European Journal of Operations Research*, 2011, [Accepted for publication].

[3] A. Delgado, R. M. Jensen, and C. Schulte, "Generating optimal stowage plans for container vessel bays," in *Proceedings of the 15th Int. Conf. on Principles and Practice of Constraint Programming (CP-09)*, ser. LNCS Series, vol. 5732, 2009, pp. 6–20.

[4] I. D. Wilson and R. P., "Principles of combinatorial optimization applied to container-ship stowage planning," *Journal of Heuristics*, no. 5, pp. 403–418, 1999.

[5] J. Kang and Y. Kim, "Stowage planning in maritime container transportation," *Journal of the Operations Research Society*, vol. 53, no. 4, pp. 415–426, 2002.

[6] W.-Y. Zhang, Y. Lin, and Z.-S. Ji, "Model and algorithm for container ship stowage planning based on bin-packing problem," *Journal of Marine Science and Application*, vol. 4, no. 3, 2005.

[7] D. Ambrosino, D. Anghinolfi, M. Paolucci, and A. Sciomachen, "An experimental comparison of different heuristics for the master bay plan problem," in *Proceedings of the 9th Int. Symposium on Experimental Algorithms*, 2010, pp. 314–325.

[8] M. Yoke, H. Low, X. Xiao, F. Liu, S. Y. Huang, W. J. Hsu, and Z. Li, "An automated stowage planning system for large containerships," in *In Proceedings of the 4th Virtual Int. Conference on Intelligent Production Machines and Systems*, 2009.

[9] R. Botter and M. A. Brinati, "Stowage container planning: A model for getting an optimal solution," in *Proceedings of the 7th Int. Conf. on Computer Applications in the Automation of Shipyard Operation and Ship Design*, 1992, pp. 217–229.

[10] D. Ambrosino and A. Sciomachen, "Impact of yard organization on the master bay planning problem," *Maritime Economics and Logistics*, no. 5, pp. 285–300, 2003.

[11] P. Giemesch and A. Jellinghaus, "Optimization models for the containership stowage problem," in *Proceedings of the Int. Conference of the German Operations Research Society*, 2003.

[12] F. Li, C. Tian, R. Cao, and W. Ding, "An integer programming for container stowage problem," in *Proceedings of the Int. Conference on Computational Science, Part I*. Springer, 2008, pp. 853–862, LNCS 5101.

[13] D. Ambrosino and A. Sciomachen, "A constraint satisfaction approach for master bay plans," *Maritime Engineering and Ports*, vol. 36, pp. 175–184, 1998.

[14] Y. Davidor and M. Avihail, "A method for determining a vessel stowage plan, Patent Publication WO9735266," 1996.

[15] O. Dubrovsky and G. L. M. Penn, "A genetic algorithm with a compact solution encoding for the container ship stowage problem," *J. of Heuristics*, vol. 8, pp. 585–599, 2002.

[16] M. Flor, "Heuristic algorithms for solving the container ship stowage problem," Master's thesis, Technion, Haifa, Isreal, 1998.

[17] M. Avriel, M. Penn, N. Shpirer, and S. Witteboon, "Stowage planning for container ships to reduce the number of shifts," *Annals of Oper. Research*, vol. 76, pp. 55–71, 1998.

[18] A. Sciomachen and 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*, vol. 14, pp. 251–269, 2003.

[19] W. C. Aye, M. Y. H. Low, H. S. Ying, H. W. Jing, and Z. Min, "Visualization and simulation tool for automated stowage plan generation system," in *Proceedings of the International MultiConference of Engineers and Computer Scientists 2010 (IMECS 2010)*, vol. 2, Hong Kong, 2010, pp. 1013–1019.

[20] S. Nugroho, "Case-based stowage planning for container ships," in *The Int. Logistics Congress*, 2004.

[21] Gecode Team, "Gecode: Generic constraint development environment," 2006, available from `http://www.gecode.org`.

[22] L. Michel and P. V. Hentenryck, "A constraint-based architecture for local search," *ACM SIGPLAN Notices*, vol. 37, no. 11, pp. 101–110, nov 2002.

[23] D. Pacino and R. M. Jensen, "A 3-phase randomized constraint based local search algorithm for stowing under deck locations of container vessel bays," IT-University of Copenhagen, Tech. Rep. TR-2010-123, 2010.