Grasp for the slot planning problem

Francisco Parreño*, Dario Pacino+, and Ramón Álvarez-Valdés**

*University of Castilla-La Mancha, Department of Mathmatics
+Technical University of Denmark, Department of Transport
**University of Valencia, Department of Statistics and Operations Research
*francisco.parreno@uclm.es, +darpa@transport.dtu.dk,**ramon.alvarez@uclm.es

Abstract. In this paper, we address the slot planning phase, in which the containers assigned to a container ship location have to be stowed, satisfying many conditions related to the way in which containers have to be stacked, the weight distribution and the specific conditions regulating the containers with dangerous products.
In order to solve in an efficient way problems of realistic size, we have developed a Greedy Randomized Adaptive Search Procedure (GRASP) algorithm, including two constructive methods, several randomization strategies, and several improvement moves. The algorithms have been tested on a set of real-world instances. The results show that the algorithm works quite well for a wide range of real instances.

Keywords: Container Vessel Stowage Planning, Slot Planning, Heuristic algorithm, GRASP

1 Introduction

Over the past two decades, the demand for cost efficient containerized transportation has seen a continuous increase. In order to answer this demand, shipping companies have deployed larger container vessels, that nowadays can transport up to 18,000 containers. These vessels sail from port to port loading and unloading thousands of containers. Minimizing the time a vessel stays at port involves, among other aspects, an efficient stowage plan, a plan describing where each container should be loaded in the vessel. A feasible stowage plan has to satisfy many different constraints, high-level constraints ensuring that the vessel is stable and seaworthy, and low-level constraints concerning the way in which each container is loaded to a position into the vessel.

The state-of-the-art stowage planning follow a 2-phase hierarchical decomposition of the problem [2,1]. In the first phase, called the *Master Planning*, containers are distributed to subsections of the vessels called *locations* (a term used in [2]). The distribution of containers must satisfy seaworthiness requirements: the *center of gravity* of the vessel must be within limits to ensure that e.g. *trim* and *draft* are feasible, and that *shear and bending moments* are at within tolerance. One of the main objectives of this phase is the minimization of the *hatch-overstowage*, meaning the number of containers on-deck that need to be re-handled due to handling activities under-deck. Since this is not the focus of this work, we refer the reader to [2] for a detailed description. The second phase of the decomposition, *Slot Planning*, refines the container distribution and identifies the exact position of each container in the vessel locations. This phase is concerned with the low-level constrains, regarding the physical position of the containers, e.g. ensuring that weight and height capacities are satisfies, and that *reefers* (refrigerated containers) are assigned to positions where a power outlet is available.

In Figure 1, it is possible to see that Master Planing requires a *loadlist*, *port* and *vessel data*. The loadlist includes the containers to load at the current port and a forecast of the ones to be loaded at later ports. Vessel and port data include information about e.g. the layout of the ship and the depth of the ports of call. If the intended output is a class-based stowage plan (where only container types are taken into account), it is possible to solve an independent Slot Planning Problem for each location of the vessel (subsections of vessel). Though this might seem a simplification of the problem, the reason for pursuing this road is rooted in the container terminal part of the optimization. Given a class-based stowage plan, terminals can optimize the load sequencing of the containers and thus further reduce the ship's time at port.

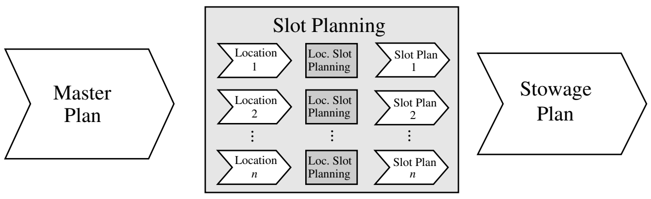

Fig. 1. The master planning and slot planning decomposition of stowage planning

Since solving the stowage planning problem over multiple ports requires the use of forecasts, stowage planners wish to be able to analyze different forecast scenarios. They thus require that the solution time does not exceed 10 minutes. According to [2,5] this requirement only leaves 1 second to solve each Slot Planning Problem.

2 Background and Problem Definition

Container vessels are ships specially designed for the transportation of large amounts of containers with a small crew. Containers are metallic boxes designed to withstand significant outer forces. They are particularly robust to high vertical compression, which allows the creation of high stacks. All containers are fitted with corner castings designed to support the container's weight, and to which security fittings can be attached. ISO standard containers are usually 20', 40' or 45' long. ISO containers are 8' wide and 8.6' tall with the exception of *high-cube* containers which are 1 foot taller. Longer containers, such as the 45' containers are equipped with two extra sets of castings at a 40' distance. The extra castings allow the longer containers to be stacked on top of 40' containers. No castings, however, exist at the 20' position, which means that 20' containers cannot be stacked on top of longer containers.

Aside from the standard and high-cube containers, there are a number of specialized containers for different kinds of cargo. Fruits and vegetables, for example, must be transported in refrigerated containers called *reefers*. Containers transporting dangerous goods are called IMO containers. Depending on the nature of the cargo, a special IMO code is assigned to the container. Strict separation rules apply to IMO containers.

The layout of a container vessel is shown in Figure 2. The figure shows how containers are arranged into storage areas called *bays*, throughout the entire vessel length. A bay is composed of a number of *cells*, each indicating a possible stowage position. Cells usually have a capacity of two Twenty Equivalent Unitss (TEUs), meaning that we can either stow two 20' containers or one 40' (or 45') container. Each TEU position within a cell is referred to as a *slot*. Slots toward the bow of the vessel are called *Fore* slots and those towards the stern are called *Aft* slots. Cells are identified by a *stack* number, indicating the horizontal position within a bay, and a *tier* number indicating the vertical position. Figure 2 also shows how only a subset of the cells have access to electric power (or *reefer plugs*).

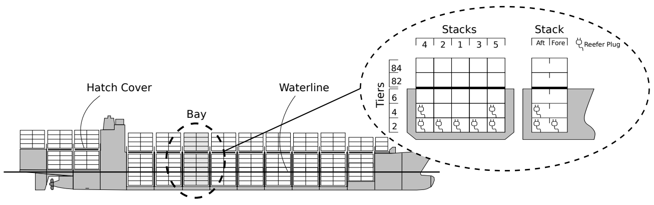

Fig. 2. The layout of a container vessel.

2.1 Slot Planning

As previously mentioned, Slot Planning assumes that a solution to Master Planning is given. We are, thus, provided with a set of containers (or container types) to be loaded into a single vessel location (subsection of the vessel). A representative Slot Planning problem was presented in [6,5]. We extend this definition by including IMO restrictions and the maximization of the loaded cargo. Thus a feasible solution must satisfy the following rules:

a) Assigned cells must form stacks (containers stand on top of each other in the stacks. They cannot hang in the air).

b) 20' containers cannot be stacked on top of 40' containers.
c) A 20' reefer container must be placed in a reefer slot. A 40' reefer container must be placed in a cell with at least one reefer slot.
d) The length constraint of a cell must be satisfied (some cells only hold 40' or 20' containers).
e) The sum of the heights and the sum of the weights of the containers stowed in a stack must be within the stack limits.
f) All containers already onboard must be stowed in their original slots and they cannot be moved to any other slots.
g) A cell must be either empty or with both slots occupied.
h) IMO rules must be satisfied.

IMO rules dictate how distant incompatible cargoes must be stowed (segregation). Since in Slot Planning we only focus on a location, it is sufficient to only concentrate on rules that allow stack segregation. Segregation that goes beyond the stack level must be handled at the Master Planning level. Without loss of generality, we have devised a representative set of rules based on 4 IMO categories and a set of rules.

An optimal Slot Plan minimizes the sum of the following objectives:

a) Minimize the number of containers out of the solution. Sometimes is not possible to stowage all the containers in the location. If F and T are the sets of 40' and 20' containers assigned to the location, respectively, a cost of 1000 $\cdot$ ($|F| + |T|$) units is paid for each container out of the solution.
b) Minimize overstows, the movements of containers to allow the unloading of other container placed below. A 100 unit cost is paid for each container overstowing any containers below.
c) Avoid stacks where containers have many different discharge ports. A 20 unit cost is paid for each discharge port included in a stack.
d) Keep stacks empty if possible. A 10 unit cost is paid for each stack used.
e) Avoid loading non-reefer containers into reefer slots. A 5 unit cost is paid for each non-reefer container stowed in a reefer slot.

3 Literature review

A growing number of works on stowage planning have been published in the past few years. The contributions can roughly be classified as belonging to multi-phase or single-phase approaches. Multi-phase approaches ([2,1,3]) are currently the best performing solutions, where the stowage planning problem is hierarchically divided into sub-problems, often into master planning and slot planning. Single-phase approaches consider the stowage planning problem as a whole. Since most of those works only have a very simplified representation of ship stability, they can also be seen as slot planning approached. Solution methods include mathematical programming [4], constraint programming [12,6], constraint-based local search [5], genetic algorithms [8], dedicated heuristics [9,11], 3D-packing [10]. Of those that are relevant for the Slot Planning Problem, in the works [9,12,8], all containers are considered to be of the same size and no distinction is made between reefer and high-cube containers. Approaches that include both 20 and 40 foot containers are [10,4], of those [10] also includes high-cubes while [4] includes reefers. To the best of the authors' knowledge, the only two works that propose a industrial strength representative Slot Planning problem are [6,5]. Both works have comparable results, and complement each other. For instances where the constraint programming approach in [6] cannot find an optimal solution, the constraint-based local search in [5] finds high-quality solutions. Both approaches, however, disregard the positioning of IMO containers and assume that the solutions of the Master Planning phase are feasible.

4 Constructive algorithm

In order to solve the slot planning problem of a location we are given the list of containers assigned to this location by the solution of the master planning problem as well as the list of containers already stowed in this location with their positions.

We follow an iterative process in which we combine two elements: a list $Cont$ of the containers still to be stored, initially the complete list of containers, a list S of stacks and, for each stack $s \in S$, a list C_s of cells in which it is possible to store a container.

At each step, we first choose a stack from S and the first tier available in this stack. At this tier we have a cell C_s, which could contain two 20 containers or a 40 container. Once we have chosen the tier, we choose the container to be stored from among the containers in $Cont$ which could be feasibly stored in it. The process goes on until all the containers have been stored or there is not any usable space left.

Step 0: Initialization

We have the list $Cont$ ordered by port, length, reefer, height and weight, each factor in non-increasing order except the length: we want to store first 20' containers and afterwards 40' containers. Reefer and high cube containers are before standard containers in the ordered list $Cont$ because we want to consider them first when choosing the container to store.

In the list S the partially filled stacks appear first and then the empty stacks. Both lists are ordered by their number of available cells in non-increasing order. This ordering tries to minimize the number of stacks being used.

Step 1: Choosing the stack and tier

We choose the first stack in the list S and its first empty tier. We do not change the stack until it is full or we cannot store more containers in it.

Step 2: Choosing the container

Once a tier has been chosen, we have the characteristics of this tier, and the conditions determined by the containers previously stored in the stack. For instance, if we have in the previous tier a 40' container, it is not possible to store 20' containers. Other aspects to be considered are the maximum weight and height left for the stack. Then we try to pack the first container on the list satisfying the constraints.

We store a 40' High Cube container when that does not reduce the number of possible tiers of this stack if there are more standard containers of this port to be stored.

If the cell has power supply we store a reefer container, if it is possible, in this tier.

If we are going to store a 20' container in this tier we check if it is possible to store a second 20' container in the other cell of this tier. We cannot leave odd empty cells.

When the tier belongs to the upper half of the stack, we have the list *Cont* ordered by port, length, reefer, height and weight, but in this case the weights appear in non-decreasing order. We want to store in each stack heavyweight containers in the lower half, and lightweight containers in the upper half. The idea is to maintain a similar average weight in each stack.

Before we store a container we check if it is possible according to the IMO constraints imposed on the stack by previously stored IMO containers.

Step 3: Updating the lists

We remove from the list the container already stored and update the values of the total weight and height of the chosen stack. This process goes on until there is no container on the list or there is no possible cells in which to store any remaining container.

Parallel constructive algorithm

In the previous constructive algorithm, once a stack is open we store all the containers that we can in this stack before considering a new stack on the list S. We call this strategy a *sequential* construction. However, we have studied an alternative in which instead of choosing a stack and fill it completely, we take a different stack every time we store a container. We call this strategy a *parallel* construction. This alternative is not good for minimizing the number of stacks used because it keeps open all the available stacks, but it produces diverse solutions and that would be useful in the iterative procedure described later.

5 GRASP algorithm

GRASP is an iterative procedure that combines a constructive phase and an improving phase. In the constructive phase the solution is built step by step, adding one element to a partial solution. For selecting the element to add, a greedy strategy is used, but the selection is not deterministic because a random strategy is used to provide diversity to the procedure. The improvement phase consists of a local search. GRASP has been successfully applied to many combinatorial optimization problems.

5.1 Randomizing the constructive algorithm

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

In Step 2, we maintain two lists of stacks, first the open stacks, those which are already partially filled, and then the empty stacks, but instead of having the lists ordered by non-increasing number of available cells, we assign to each stack a number taken at random from 0 to its number of available cells and order each list according to these random numbers, using the same order previously defined for cells.

In Step 3, we consider two ways of ordering the containers. The first one, already described in the constructive algorithm, orders the containers by port, length, reefer, height and weight. The other alternative orders them by port, length, reefer, weight and height.

To randomize these orderings we have used two alternatives. The first one is the introduction of a noise in the weight of the container, similarly to what we did with the stacks, the new weight of each container is a random value between 0 and the original weight of the container, and the list of containers is reordered with these new weights at each iteration. This method randomizes the choice of container only at the level of weight or height, but the order of port, length and reefer is kept fixed.

The second randomization method considers all the remaining containers, ordered by non-increasing value, and takes a random sample. The size of this sample is determined by a parameter δ that indicates the proportion of the containers to be sampled, so each container has a probability δ to be chosen. We have a parameter δ with $0 < \delta < 1$, where $delta = 1$ is the deterministic algorithm and δ=0 is completely random.

In order to introduce even more diversity in the solutions, we use two more randomizing procedures. As we have developed two constructive methods, sequential and parallel, at each iteration we choose randomly one of them to build the solution.

5.2 Determining the parameter δ

We have used reactive GRASP, proposed by Prais and Ribeiro[7], in which δ is taken at random from the set of discrete values $\{0.1, \dots, 0.9\}$. As we have two different methods (sequential and parallel) to build the solution, it is possible that one value of delta can have good values for the sequential but bad values for the parallel, so we keep two lists of values for δ, one for the sequential case and other for the parallel.

5.3 Improvement methods

The first improvement consists of removing the last containers stored by the randomized algorithm and store them again with the deterministic algorithm.

We have a parameter that determines how many containers to remove from the solution. We choose a random percentage $\gamma \in (35, 95)$. We keep the first γ% of the containers and remove the others. To complete the solution we only use the sequential constructive algorithm. This method is based on the fact that wrong selections in the final stages of the randomized constructive process may produce bad solutions and therefore significant improvements may be obtained by doing the final part of that process again with the deterministic procedure.

The second movement consist of removing a set of stacks of the solution and store them again with the deterministic algorithm. We select randomly a number of stacks of the solution and remove the stacks stored in the last place.

6 Computational experiments

The algorithms were coded in C++ and run on a Intel Core Duo 2.93 GHz with 4 GB of RAM. For the GRASP algorithms we established a stopping criterion of a maximum of 20000 iterations or 4000 iterations without improving the best solution known.

We have two set of instances. The first set, Set I is that used by Delgado et al. [6], composed of 236 slot planning instances derived from complete stowage plans provided by a shipping company. Each instance has been generated by restowing a random location in one of the stowage plans. Since the plans have been applied in real life, we can assume that the containers have been assigned to locations according to the preferences of stowage coordinators.

In the second set, Set II, there are 425 instances also generated from real slot planning instances. Some of these instances have IMO containers and in general they are more difficult because in some of them it is not possible to store all the containers assigned to each location. There are in total 4 groups: Weight, HC, Capacity and IMO. Each group is aimed at targeting a specific problem property, e.g. in the weight group all instances are forced to have a total weight of containers equal to a specified percentage of the total weight capacity.

Table 1 shows the average values (in thousands) of the objective function for six versions of the constructive and GRASP algorithms described in previous sections: the constructive parallel algorithm, the constructive sequential algorithm, the randomized constructive algorithm using random noise (Noise), the randomized constructive algorithm using random samples (Sample), the GRASP algorithm (Randomized constructive + Improvements), the Reactive GRASP algorithm (the value of parameter δ is adjusted reactively) and the average time in seconds for the reactive GRASP algorithm.

The last column shows the average time for the Reactive GRASP algorithm.

We can see that the sequential construction outperforms the parallel for all types of problems. We can also see that the method based on random samples works better than the version using random noise. The effect of the improvements is quite considerable and the reactive technique provides better results than the non-reactive version. The reactive algorithm, which is the most complex procedure, requires just one second on average.

Type	**#**	**Parallel**	**Sequential**	**Noise**	**Sample**	**GRASP**	**Reactive**	**Time**
Set I	236	659	320	99	104	24	24	0.63
Capacity	75	6988	3831	2315	1680	1533	1370	0.95
HC	125	9027	5212	1862	931	844	648	1.16
IMO	150	5695	3168	892	653	621	529	1.04
Weight	75	5279	3370	1623	1239	947	915	0.97
Overall	661	**4627**	**2636**	**1037**	**693**	**590**	**510**	**0.90**

Table 1. Results for constructive and GRASP algorithms

We can compare our results with the constraint programming from Delgado et al. [6] because we are using the same instances and solving the same problem. Table 2 compares the results of the GRASP algorithm running one and ten seconds and the CP model by Delgado et al. [6] running also one and ten seconds of CPU time. In order to have similar computational times we have modified the stopping criterion, decreasing the number of iterations by 10. Then we have a maximum of 2000 iterations or 400 iteration without improving the solution. The table shows the values grouped by category. For each method, the first column indicates the number of problems solved (%Sol), the second column the number of problems solved optimally (%Opt), and the third the total time for all the instances in the group (Time). We can see that our GRASP algorithm obtains higher percentages of found solutions and similar results in the percentages of optimal solutions.

		Reactive						Constraint Programming					
		1 sec			10 sec			1 sec			10 sec		
Group	#	%Sol	%Opt	T	%Sol	%Opt	Time	%Sol	%Opt	T	%Sol	%Opt	T
1	13	100	92	0,8	100	92	3,2	100	100	0,1	100	100	0,1
2	22	100	100	4,2	100	100	19,0	91	91	3,6	91	91	21,6
3	13	100	100	0,0	100	100	0,0	100	100	0,5	100	100	0,5
4	78	100	97	6,8	100	100	28,4	96	95	6	99	99	19,7
5	36	100	100	6,5	100	100	28,1	92	89	7,1	92	92	39
6	15	100	100	0,7	100	100	2,7	100	93	1,2	100	100	5,4
7	14	100	79	3,5	100	79	15,1	64	57	6,8	64	64	53,5
8	14	100	79	1,6	100	86	8,4	93	93	1,5	93	93	10,5
9	17	94	71	4,3	94	82	22,9	76	76	5,2	88	88	36,5
10	8	100	63	0,8	100	75	4,8	100	100	0,7	100	100	0,7
11	6	100	50	0,6	100	50	4,7	83	83	1,3	83	83	10,3
Overall		99,6	93	31,6	99,6	94	145,6	92	90	34	94	94	198

Table 2. Comparing with Constraint Programming [6].

7 Conclusions

We have developed a GRASP algorithm that obtains good results in short computing times. When the problem is small, it seems to be more appropriate solving it by using the integer formulation that works quite well, but when the problem is larger and there are other constraints and characteristics the metaheuristic scheme outperforms the results of the exact models.

In the future we plan to solve a more general slot planning problem in which the solution of the master planning phase is not given as a set of specific containers assigned to a location, but as a number of containers of each port and length assigned to each bay.

References

1. Ambrosino, D., Paolucci, M., Sciomachen, A.:Experimental evaluation of mixed integer programming models for the multi-port master bay plan problem, Flexible Services and Manufacturing Journal,1–22, (2013).
2. Pacino, D., Delgado, A., Jensen, R.M., Bebbington, T.:Fast generation of near-optimal plans for eco-efficient stowage of large container vessels, Computational Logistics,286–301, (2011)
3. Ambrosino, D., Anghinolfi, D., Paolucci, M., Sciomachen, A.: An Experimental Comparison of Different Heuristics for the Master Bay Plan Problem, Proceedings of the 9th Int. Symposium on Experimental Algorithms,314–325, (2010)
4. Ambrosino, D., Sciomachen, A., Tanfani, E.: Stowing a Containership: the Master Bay Plan Problem, Transportation Research, 38, 81–99, (2004)
5. Pacino, D,. Jensen, R.M.: Constraint-Based Local Search for Container Stowage Slot Planning, Proceedings of The International MultiConference of Engineers and Computer Scientists, Lecture Notes in Engineering and Computer Science, 1467–1472, (2012)
6. Delgado, A,. Jensen, R.M, Janstrup, K., Rose, T.H, Andersen, K.H.: A constraint programming model for fast optimal stowage of container vessel bays,European Journal of Operational Research, 220(1),251–261, (2012)
7. Prais, M., Ribeiro, C.C.: Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment, INFORMS Journal on Computing, 12(3), 164–176, (2000)
8. Dubrovsky, O. and Levitin, G. and Penn, M.: A Genetic Algorithm with a Compact Solution Encoding for the Container Ship Stowage Problem, Journal of Heuristics, 8, 585–599, (2002)
9. Avriel, M.,Penn, M., Shpirer, N., Witteboon, S.: Stowage planning for container ships to reduce the number of shifts, Annals of Operations Research, 76, 55–71, (1998)
10. Sciomachen, A. and Tanfani, A.: The master bay plan problem: a solution method based on its connection to the three-dimensional bin packing problem, (IMA) Journal of Management Mathematics, 14, 251–269, (2003)
11. Ding, D., Chou, M.C.: Stowage Planning for Container Ships: A Heuristic Algorithm to Reduce the Number of Shifts,European Journal of Operational Research (2015)
12. Ambrosino, D., Sciomachen, A. : A constraint satisfaction approach for master bay plans, Water studies series, 175–184, (1998)