Crane Intensity and Block Stowage Strategies in Stowage Planning

Pacino, Dario

Published in:
International Conference on Computational Logistics

Link to article, DOI:
10.1007/978-3-030-00898-7_12

Publication date:
2018

Document Version
Peer reviewed version

Crane intensity and block stowage strategies in stowage planning

Dario Pacino[1 [0000−0002−7255−004X]]

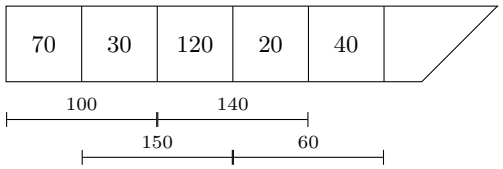
Technical University of Denmark, DTU Management Engineering, darpa@dtu.dk

Abstract. The increasing size of container vessels is raising the complexity of daily operations of both the carrier and the terminal. This paper focuses on stowage planning, the problem of assigning container to positions in a vessel. In particular, it studies the implementation of known planning strategies within an optimisation framework. Block stowage and crane intensity are presented and mathematically modelled on a simplified version of the problem. An experimental evaluation, on a large set of novel benchmark instances, shows that even in this simplified version the problem is not trivially solved. A matheuristic based on large neighbourhood search is presented, which is able to find a solution to all instances in short computational times.

1 Introduction

The container shipping industry has continuously grown in the past many years, and though it now experiencing a period with little growth [13], the complexity of the daily planning operations is still very high. The use of mega-vessels (now able of carrying more than 20,000 containers), is not only having an impact on port operations, but it is also making the cargo planning of the vessel a very complex and time-consuming task. This task is known as stowage planning, and it is often performed by the carrier a few hours before calling each port. In the past decade, the number of academic works on stowage planning has increased showing a continuous interest from the community. Solution approaches are divided between theoretical works, where a deeper understanding of specific optimisation challenges is sought (e.g.[10, 5, 12, 3]), and applied approached where heuristic and decomposition methods aim at solving rich stowage planning problems that can be implemented in practice (i.e. [1, 2, 6–8]). Theoretical works are characterised by simple definitions of the problem where e.g. only one container size is assumed and where stability constraints are ignored. Those works, however, focus on some particular combinatorial challenges such as allowing containers to be shuffled along the route or understanding computational complexity. The work presented in this paper belongs to this category, though it is motivated by more applied issues. While academic works tend to look at optimal conditions, practical stowage planners are used to work with uncertain data and rules-of-thumb. It would then be reasonable to assume that a first professional implementation of a decision support system for stowage planning would follow current planning

practices. It turns out that current planning practices face hard combinatorial problems that have not yet been studied and for which, current state-of-the-art methods are not applicable. One of such problems comes from the concept of *crane intensity*. Crane intensity is an estimation of the number of cranes that a container terminal needs to use to handle a vessel. Crane intensity is calculated by dividing the total number of container moves by the number of moves the longest crane will perform. The longest crane is the crane that has the most moves assigned. To better explain this, consider the example in Figure 1. The figure shows a vessel which is divided into bays. The number within each bay represents the number of container moves (load or discharge) that will have to be performed. Since two handling cranes cannot work on adjacent bays, the intervals below the vessel represent all the possible combination of adjacent bays and their respective workload. In this example, the longest crane has a total of 150 moves resulting in a crane intensity of 3. The longest crane can be seen as the handling operations makespan as described in [6]. In the remainder of the paper, we will refer to the longest crane as the makespan.



longest crane = 150
total moves = 450
crane intensity = $\frac{450}{150} = 3$

Fig. 1: Example of crane intensity calculation.

Crane intensity is not used by stowage planner as a KPI, it instead used as a target performance. The planners know by experience that at a given port, the vessel is usually serviced by a specific number of cranes, f.ex. 3. By forcing a stowage plan to have a crane intensity of 3, the container assignment will be forced to distribute containers along the bays such that the 3 cranes are fully utilised[1].

Another important concept is *block stowage*, which referrers to the practice of dividing a bay into logical sub-section to which only containers with the same discharge port can be assigned. This rule is used to avoid overstowage (which occurs when a container with a later discharge port is stowed over one with an earlier discharge port), and to improve container handling at the port. When containers with the same discharge port are clustered together, it is easier for the terminal to implement more advanced handling operations such as dual-cycling (interchanging load and discharge operations) and tandem lifts (moving more than one container at the time). The modelling of block stowage is already present in the scientific work of e.g [1] and [4]. To the best of the author's knowledge, no research has been published on the modelling of crane intensity and its combination to a block stowage policy.

[1] Note that this is a rule-of-thumb used by the industry.

This paper presents the Block Stowage Problem with Crane Intensity (BSPCI), a simplified stowage planning problem that only focuses on the combinatorial interplay between the block stowage strategy and the targeting of a specific crane intensity. With this new problem, we aim at finding efficient solution methods which can build the foundation for more rich problem definitions. We propose a mathematical formulation and a matheuristic based on the Large Neighborhood Search (LNS) framework. We test the mathematical formulation and the heuristic approach on a benchmark of 600 instances. The results show that the LNS is able to find solutions for all the instances where the mathematical model fails. For the remaining instances, the LNS can reach solutions that are either better or within 10% from the best-known solution in 75% of the cases.

The remainder of the paper is organised as follows. Section 2 formally introduces the problem and the compact formulation, followed by Section 3 where the design of the matheuristic is presented. Section 4 discussed the computational results before conclusions are drawn in Section 5.

2 Background and problem definition

A container vessel is a commercial vehicle designed to sail with standardised cargo containers. The most common containers (ISO containers) are 8' wide, 8.6' high, and 20', 40' or 45' long. There exists also a number of containers with such as refrigerated containers, tanks for special cargo. Each container is stowed on a cargo hold called a bay. A bay can hold multiple containers arranged in stacks. A cell indicates the position of a 40' long container and it is identified by a stack number and a tier number (the vertical index in the stack). The stowage of containers in a bay is subject to a number of physical constraints, i.e. maximum weight limits, availability of power plugs, and capacity limits. Aside from these stacking constraints, a loaded vessel must also be seaworthy, meaning that it must be stable while sailing, that it does not run aground when calling a port etc. We refer the reader to [6, 7] for a detailed description of all the constraints governing stowage planning in practice. In this paper, we focus on a simplified version of the problem where only the main combinatorial aspects regarding crane intensity are taken into account.

We assume the bays of the vessel to be composed of a number of blocks. Each block can be seen as a logical grouping of stacks (not necessarily adjacent). The vessel travels on a predefined route, where containers can be loaded or discharged. The number of containers to transport from each port to any other port is known in advance. A block is only allowed to stow container destined to the same port. The port assignment of the block is not predetermined and is thus a part of the decision. All containers are of the same size and are coupled with an origin/destination port. The ship is assumed empty in the first port and after it arrives at the last port. At each port, all the containers destined to that port are discharged, and container destined to the following ports are loaded. At each port, the total number of operations in a bay is given by the number of load and discharge operations to be performed on that bay. Container moves

are performed by the cranes of the container terminal, and no two cranes can work on adjacent bays. We disregard stacking and stability constraints aside from limiting the capacities of the blocks.

The BSPCI aims at finding an assignment of containers to blocks throughout the entire route. The assignment has to minimise the sum of the absolute difference between the found and the given crane intensity at each port.

Let us describe the problem more formally by first introducing the mathematical notation.

Sets

$P = \{1, 2, \ldots, n\}$	The set of visited ports, where n is the last port.
$P_i^j \subseteq P$	The set of ports between port $i \in P$ and $j \in P$.
C	The set of blocks.
B	The set of adjacent bay pairs.
$C_b \subset C$	The set of blocks belonging to the same pair of adjacent bays $b \in B$.
$\mathcal{T}$	The set of all pair of origin/destination ports $\{(i,j) \vert i,j \in P, i < j\}$.

Coefficients

q_c	The capacity of block $c \in C$.
tms_i	The target makespan at port $i \in P$.
$\overline{c}_i$	The cost for exceeding the target makespan at port $i \in P$.
$\underline{c}_i$	The cost for not reaching the target makespan at port $i \in P$.
t_{ij}	The number of containers to transport from port $i \in P_1^{n-1}$ to port $j \in P_{i+1}^n$.
$\hat{t}_{ij} = \sum_{k \in P_1^i} t_{kj}$	The total number of j-containers on board upon leaving port $i \in P$. A j-container is a container destined to port $j \in P$.

Decision variables

$x_{ij}^c \in \mathbb{Z}_+$	The number of j-containers ($j \in P_{i+1}^n$) loaded in block $c \in C$ at port $i \in P_1^{n-1}$
$y_{ij}^c \in \mathbb{B}$	A binary variable equal to 1 if at least one j-container is stowed in block $c \in C$ upon leaving port $i \in P_1^{n-1}$
$z_i \in \mathbb{R}_+$	The makespan at port $i \in P$
$\delta_i^{\in} \mathbb{R}_+$	Auxiliary variable equal to the difference between z_i and the number of operations in the adjacent bays $b \in B$ performed at port $i \in P$
$\beta_i^b \in \mathbb{B}$	Auxiliary variable equal to 1 if and only if $\delta_i^b > 0$ for $b \in B$
$u_i \in \mathbb{R}_+$	A variable equal to the deviation from the target makespan if, at port i, z_i is strictly less than tms_i and 0 otherwise
$o_i \in \mathbb{R}_+$	A variable equal to the deviation from the target makespan if, at port i, z_i is strictly greater than tms_i and 0 otherwise

With the presented notation, the BSPCI can be formulated as the following mixed-integer program:

$$z^* = \min \sum_{i \in P_1^n} (\bar{c}_i o_i + \underline{c}_i u_i) \tag{1}$$

$$\text{s.t.} \sum_{k \in P_1^i} \sum_{c \in C} x_{kj}^c = \hat{t}_{ij} \qquad (i,j) \in \mathcal{T} \tag{2}$$

$$y_{ij}^c \leq \sum_{k \in P_1^i} x_{kj}^c \leq q_c y_{ij}^c \qquad (i,j) \in \mathcal{T} \quad c \in C \tag{3}$$

$$\sum_{j \in P_{i+1}^n} y_{ij}^c \leq 1 \qquad i \in P_1^{n-1} \quad c \in C \tag{4}$$

$$\delta_1^b + \sum_{c \in C_b} \sum_{j \in P_2^n} x_{1j}^c = z_1 \qquad b \in B \tag{5}$$

$$\delta_n^b + \sum_{c \in C_b} \sum_{i \in P_1^{n-1}} x_{in}^c = z_n \qquad b \in B \tag{6}$$

$$\delta_i^b + \sum_{c \in C_b} \left(\sum_{r \in P_1^{i-1}} x_{ri}^c + \sum_{j \in P_{i+1}^n} x_{ij}^c \right) = z_i \quad i \in P_2^{n-1} \quad b \in B \tag{7}$$

$$\delta_i^b \leq M \beta_i^b \qquad i \in P \quad b \in B \tag{8}$$

$$\sum_{c \in \hat{C}} \beta_i^c \leq |\hat{C}| - 1 \qquad i \in P \tag{9}$$

$$z_i + u_i - o_i = tms_i \qquad i \in P \tag{10}$$

$$x_{ij}^c \in \mathbb{Z}_+ \qquad (i,j) \in \mathcal{T} \quad c \in C \tag{11}$$

$$y_{ij}^c \in \mathbb{B} \qquad (i,j) \in \mathcal{T} \quad c \in C \tag{12}$$

$$z_i \in \mathbb{R}_+ \qquad i \in P \tag{13}$$

$$\delta_i^b \in \mathbb{R}_+ \qquad i \in P \quad b \in B \tag{14}$$

$$\beta_i^b \in \mathbb{B} \qquad i \in P \quad b \in B \tag{15}$$

$$u_i, o_i \in \mathbb{R}_+ \qquad i \in P \tag{16}$$

The objective function (1) aims to minimize the weighted sum of the number of ports where the makespan is not equal to the target makespan. Since all containers must be stowed, the total number of moves is constant and the crane intesity measure can be translated to a target makespan. Constraints (2) guarantee that all container transports are satisfied. Constraints (3) are block capacity constraints and act as on-off constraints for the y-variables based on the values of the x-variables. Constraints (4) are block stowage constraints ensuring that each block can contain only containers with the same discharge port. Constraints (5)-(9) set all z-variables equal to the makespan of the number of operations for each port $i \in P_1^n$; in particular, constraints (5) concern port 1, constraints (6) port n, and constraints (7) ports 2 to $n-1$. Constraints (8)-(9) ensure that at least one β-variable is equal to 0 for each port i, thus setting the corresponding δ_i^c equal to zero and the makespan (i.e., variable z_i) equals the number of operations in

the subset of four consecutive blocks. Constraints (10) set u_i and o_i equal to the difference between z_i and tms_i for each port $i \in P$. Constraints (11)-(16) define the range of the decision variables.

3 LNS based matheuristic

In Section 4, we show that the proposed formulation is not applicable to efficiently solve the BSPCI, thus heuristic methods are sought. It is important to note, however, that a number of instances can indeed be solved by the mathematical formulation. This insight has inspired us to use a mathematical-based heuristic to solve the BSPCI. We adopted the LNS framework where, given an initial solution, at each iteration, a part of the current solution is destroyed using a destroy operator. A repair heuristic is then used to rebuild the solution. This process is iterated until a termination criterion is met. Since the LNS framework is well-known, we refer the reader to [9] for a more in-depth description of the framework and its extension. The remainder of the section will, instead, present how each of the main LNS components has been adapted to solve the BSPCI.

3.1 Initial solution

Finding an initial solution to the BSPCI is not trivial. Given that blocks have different capacities, it is not simple to analytically identify the number of blocks needed, moreover, this decision is made more difficult by the fact that a discharge port will also need to be assigned. We propose a 2-phase approach where first a mathematical model identifies the number of blocks to be used and, subsequently, a heuristic procedure assigns containers to the blocks.

Since it is reasonable to assume that many blocks in a container vessel will have the same capacity, let Q be the set of available block capacities, and C_q be the number of blocks of capacity $q \in Q$. We define a j-block to be a block assigned to only hold j-containers (where $j \in P$ and a j-container is a container destined to port j). The decision variable of the model, $x^q_{ij} \in \mathbb{Z}_+$, identifies the number of j-blocks with capacity q to be added to the j-blocks used in ports previous to i. The model is then formulated as follows:

$$\max \sum_{i \in P_1^{n-1}} \sum_{j \in P_{i+1}^n} \sum_{q \in Q} x^q_{ij} \tag{17}$$

s.t.

$$\sum_{h \in P_1^i} \sum_{q \in Q} q x^q_{hj} \geq \hat{t}_{ij} \quad \forall (i,j) \in \mathcal{T} \tag{18}$$

$$\sum_{h \in P_1^i} \sum_{j \in P_{i+1}^n} x^q_{hj} \leq C_q \quad \forall i \in P_1^{n-1}, q \in Q \tag{19}$$

$$x^q_{ij} \in \mathbb{Z} \quad \forall q \in Q, (i,j) \in \mathcal{T} \tag{20}$$

The objective function (17) maximise the number of used blocks. It is not strictly necessary to solve the model as an optimisation problem, but we believe that this will give more flexibility during the subsequent heuristic search since containers can be distributed to more blocks thus making it easier to stay within the target makespan. Constraints (18) ensures that at each port we assign enough j-blocks to fulfil the container demand $\hat{t}_{ij}$. The number of used blocks is then restricted by Constraints (19). Finally, the domain of the variables is defined in Constraint (20).

The mathematical model of the first phase effectively identifies how many block to use at each port for a specific discharge destination. In the second phase, Algorithm 1 uses this information to assign containers to each block, at every port.

The algorithm assigns containers to blocks starting from the first port and continuing in order (line 2). At each port, the set of discharge ports are sorted according to the number of moves to be performed such that the discharge port with the most containers is assigned first (line 3). The actual container assignment starts in line 4. We start by assigning the total number of container move (at the current port) to an auxiliary variable L. So long as L is positive, it means that we still have containers moves to perform (line 5). We keep count of the containers destined to discharge port d to be loaded/unloaded at port p, and we keep a sorted list of blocks ($\hat{C}$). The list sorts the blocks first by descending objective cost and then by ascending capacity. A block has a positive objective cost if the block is part of the adjacent bays defining the makespan. The cost of the block is then equal to the part of the objective cost for port p. The list is composed of the available blocks for port $p \in P$ and discharge port $d \in D$ ($\hat{C}_{pd}$). The set of available blocks is computed with a simple procedure. Blocks are evaluated sequentially starting from the first one. If a block has any remaining capacity and has been assigned to discharge port d (or it has not yet been assigned) it is included in the set. We keep adding blocks to the set until we reach the amount identified in the solution of the first phase. The first block in the list is then selected for container assignment (line 7). If the block is empty, it first needs to be assigned to the selected discharge port (lines 8-9). Lines 10-13 assign containers to the block. Since a solution with minimum cost is one that reaches the target makespan, we first check if the block is affecting the makespan (line 10). If this is not the case we assign as many containers as we have available, though at most the amount needed to reach the makespan or the capacity of the block (line 11). Otherwise, we load as many containers as the capacity of the block allows. The actual assignment is performed in line 14, which the updates all the necessary variables. The main idea behind this procedure is of trying to first load containers in blocks until the makespan is reached. The remaining containers are then greedily assigned where capacity is available. Once an initial solution is found, the mathematical model is run for 10 seconds to warmstart the LNS.

Algorithm 1: 2^{nd} phase of the initial solution procedure

```
D_bp = 0;// Discharge port assignment for block b at port p
for each port p ∈ P do
    for each discharge port d ∈ D sorted by moves do
        L = t̂_pd; // Containers to load
        while L > 0 do
            Ĉ = sort(Ĉ_pd) by cost and capacity;
            b = POP(Ĉ);
            if block b is empty then
                D_bi = d   ∀i ∈ P_p^d;
            if makespan is not reached then
                L = min(ΔMk, q_b, L);
            else
                L = min(q_b, L);
            ASSIGN_ LOAD(L, b, p, d);
```

3.2 Repair operator

In our adaptation of the LNS framework, we use a single repair operator based on the mathematical model described in Section 2. As we will see in the next section, a number of destroy operators are used to select which parts of the solution have to be removed. Let $\hat{X}$ be the set of variable assignments which has to be re-evaluated, where $(c, i, j) \in \hat{X}$ represents the indexes relative to the variable assignment for block $c \in C$, at port $i \in P^1_{n-1}$ for discharge port $j \in P^n_{i+1}$. Assume that $\bar{X}$ is the set of all variable assignment indexes in the current solution, the repair operator solves the model from Section 2 with the following additional variable fixings:

$$x^c_{ij} = v^c_{ij} \quad \forall (c, i, j) \in \bar{X} \setminus \hat{X} \tag{21}$$

where v^c_{ij} is value assigned to variable x^c_{ij} in the current solution.

3.3 Destroy operators

Six destroy operators have been designed for the BSPCI, each targeting special parts of the problem. The operators are selected at random at each iteration and they can be roughly classified as random and cost-based, and are described in the following.

Random destroy This is the simplest of the destroy operators, where variable assignments are simply selected at random, thus

$$\hat{X} = \{(c, i, j) | (c, i, j) \in \bar{X}, r^c_{ij} \leq \rho_1\},$$

where $r^c_{ij} \in [0,1]$ is a random value for each variable assignment indexed by $x \in \bar{X}$, and $\rho_1 \in [0,1]$ is a parameter of the algorithm.

Random bin destroy This operator selects bins (sets of adjacent bays) at random and relaxes all the variable assignments related to the selected bins, thus

$$\hat{X} = \left\{(c,i,j)|(c,i,j) \in \bar{X}, c \in C_b, b \in B, r_b \leq \rho_2\right\},$$

where $r_b \in [0,1]$ is a random value for each bin, and $\rho_2 \in [0,1]$ is a parameter of the algorithm.

Random discharge port assignment destroy This operator targets blocks with a specific discharge port assignment. Let $d \in P$ be a discharge port selected uniformly at random. The set of relaxed assignments is then

$$\hat{X} = \left\{(c,i,j)|(c,i,j) \in \bar{X}, j = d, r_x \leq \rho_3\right\}$$

where $r_x \in [0,1]$ is a random value for each variable assignment index $x \in \bar{X}$, and $\rho_3 \in [0,1]$ is a parameter of the algorithm. We use r_x in order to limit the size of the relaxed solution.

Random Block destroy The two previous destroy operators can be seen as a version of the Shawn-removal technique [11] used in vehicle routing, where block assignments (either by discharge port or by bin association) are relaxed together. This version of the operator is more basic and only selects blocks at random, thus

$$\hat{X} = \left\{(c,i,j)|(c,i,j) \in \bar{X}, c \in C, r_c \leq \rho_4\right\},$$

where $r_c \in [0,1]$ is a random value for each block, and $\rho_4 \in [0,1]$ is a parameter of the algorithm.

Random Port destroy This operator aims at re-optimising the portion of the solution related to a specific port. Given a port p selected uniformly at random,

$$\hat{X} \subseteq \left\{(c,i,j)|(c,i,j) \in \bar{X}, i = p, r_x \leq \rho_5\right\},$$

where $r_x \in [0,1]$ is a random value for variable assignment index, and $\rho_5 \in [0,1]$ is a parameter of the algorithm.

Cost based bin destroy This and the next three destroy operators are cost based versions of the operators we have already seen. This particular case is an extension of the *Random bin destroy* operator. The aim is to make the random selection biased toward bins that, if changed, might have an impact on the objective function. For each bin $b \in B$ at every port $i \in P$, we calculate an impact factor $f_{bi} = \frac{z_{bi}}{z_i}$ where z_{bi} is the total number of moves to be performed in bin b at port i. In order to mitigate the impact of the bias, we also draw a random number $r_b \in [0,1]$ to be combined with the impact factor. The set of relaxed variable assignments is then

$$\hat{X} = \left\{(c,i,j)|(c,i,j) \in \bar{X}, c \in C_b, b \in B, \frac{f_{bi} + r_b}{2} \leq \rho_6\right\}$$

Cost based discharge port assignment destroy Using the same cost impact factor as in the previous operator (f_{bi}), this destroy operator implements a cost based version of the *Random discharge port assignment destroy* by defining

$$\hat{X} = \left\{ (c,i,j) | (c,i,j) \in \bar{X}, j = d, \frac{\max_{b \in B_c} (f_{bi}) + r_x}{2} \leq \rho_7 \right\}$$

where B_c is the set of bins including block c.

Cost based block destroy This destroy operator is very similar to the previous one, with the difference that we do not restrict the variable assignment selection to specific discharge ports. More formally the set of variable assignments to relax is

$$\hat{X} = \left\{ (c,i,j) | (c,i,j) \in \bar{X}, \frac{\max_{b \in B_c} (f_{bi}) + r_x}{2} \leq \rho_8 \right\}.$$

Cost based move destroy This is another relational destroy operator. The main idea is to first relax variables assignments of the bin that determines the makespan at a port. Let $p^M \in P$ be a randomly selected port with a positive impact on the objective function, and $b^M \in B$ be the bin that defined the makespan at port p^M. The first set of variable assignment to relax is then indexed by

$$\hat{X}^B = \left\{ (c,i,j) | (c,i,j) \in \bar{X}, i = p^M, c \in C_b^M \right\}.$$

In order to improve the solution we now need to relax variable assignments that will allow us to either add or remove containers for the b^M bin. This can be achieved by relaxing blocks that have the same discharge port as those in the bin b^M, and which have available capacity. Let $D(x)$ be the discharge port assigned to block c of the triplet $x = (c,i,j)$. The variable assignment we want to relax are the indexed by

$$\hat{X}^R = \bigcup_{d \in \{D(x) | x \in \hat{X}'\}} \left\{ (c,i,j) | (c,i,j) \in \bar{X}, i = p^M, j = d, \sum_{j \in P_i^i} x_{ij}^c < q_c \right\}.$$

Finally we also include variable assignments for blocks that are empty

$$\hat{X}^E = \left\{ (c,i,j) | (c,i,j) \in \bar{X}, i = p^M, \sum_{j \in P_i^i} x_{ij}^c = 0 \right\}$$

and the union of all these sets defines the set of variable assignments to relax

$$\hat{X} = \hat{X}^B \cup \hat{X}^R \cup \hat{X}^E.$$

3.4 Acceptance and termination criteria

A new solution s' is accepted if its objective value ($f(s')$) is better than that of the current solution s. Non-improving solutions that have the same objective value as the current solution are accepted only if they have not been visited before. An hash-key is generated for each solution and used to individually identify already visited configurations. A time limit of 300 seconds has been selected as termination criteria for the heuristics.

4 Computational results

The LNS matheuristic and the mathematical formulations have been tested on a 2.30 GHz Intel Xeon E5 Processor with 128GB of RAM. The heuristic has been implemented using C++ and all models have been solved using CPLEX version 12.8.

The experiments are based on a randomly generated set of 600 benchmark instances composed of 8 vessels with a capacity to carry from 1,200 to 18,000 containers (see Table 1 for details). For each of these vessels, an instance group is generated assuming a route visiting 5, 10, 15, 20 and 25 ports. For each of these groups, sub-groups are created to target specific origin/destination patterns: long distance, short distance and mixed distances as defined in [3]. Five instances are then generated for each sub-group[2]. According to our industrial collaborator, exceeding the target makepasan should be double as expensive as not reaching it, we have thus assigned $\overline{c}_i = 2$ and $\underline{c}_i = 1$ for all ports $i \in P$.

	Vessels							
	A	B	C	D	E	F	G	H
Bays	10	10	18	18	20	20	24	24
Bay capacity	120	210	180	300	420	600	624	750
Larger block capacity	80	140	120	200	280	400	416	500
Smaller block capacity	40	70	60	100	140	200	208	250
Vessel capacity	1,200	2,100	3,240	5,400	8,400	12,000	14,976	18,000

Table 1: Vessels' capacity. Each bay is assumed to be composed of two blocks. A larger one representing the outer stacks, and a smaller one representing the central stacks.

4.1 Evaluation of the compact model

The large set of benchmark instances has been designed to evaluate the impact each instance feature has on the solution of the compact formulation. With a

[2] The instances can be obtained upon request to the author.

time limit of one hour, the model is able to find optimal solutions for only 20 instances. Feasible solutions are found for 431, while 149 instances are not solved. Though the formulation is able to find a large number of feasible solutions, the average gap to the lower bound is ca. 65%. Figure 2 shows a histogram of the gap distribution among the solved instances, where it is possible to see that most solutions have more than 50% gap.

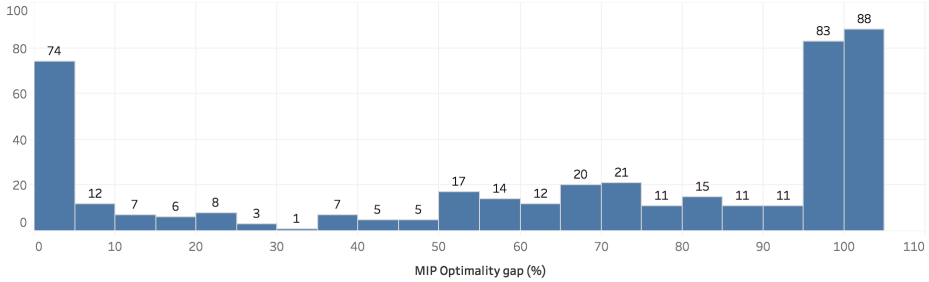


Fig. 2: Histogram representing the distribution of the gap between feasible solutions and CPLEX lower bound.

Figures 3 and 4 show the distribution of feasible, optimal and unsolved instances with respect to vessel size and number of visited ports, respectively. As expected, the larger the vessels and the number of visited ports the harder the problem it is to solve with the compact formulation. Tests have also been run to identify patterns between the size of the problem and the optimality gap, however, they did not produce any results worth mentioning.

More interesting are the results obtained when the objective weights are changed. Figure 5 presents an histogram for different combinations of cost coefficients. E.g. column "U:1 O:2" represents a cost coefficient assignment of $\underline{c}_i = 1$ and $\overline{c}_i = 2$ for all ports $i \in P$. As expected, simple feasibility problems (U:0 O:0) are easier to solve and the more components we add to the objective function the harder it is to find optimal solutions. For all of the presented combinations there exists a number of instances for which a solution is never found.

4.2 Evaluation of the LNS

Compared to the compact formulation, the LNS algorithm is able to find feasible solutions for all problem instances i.e 149 instances more than the mathematical model. For the instances where the compact formulation finds optimal solutions, the matheuristic is able to match those with an average optimality gap of 0.2%. Figure 6 depicts an analysis of the quality of the solutions found by the LNS and the feasible solutions found by the compact formulation. As is can be seen, in most cases, the two solutions differ only by a few hundred containers during the entire voyage. There do exist outliers where either approach finds much better

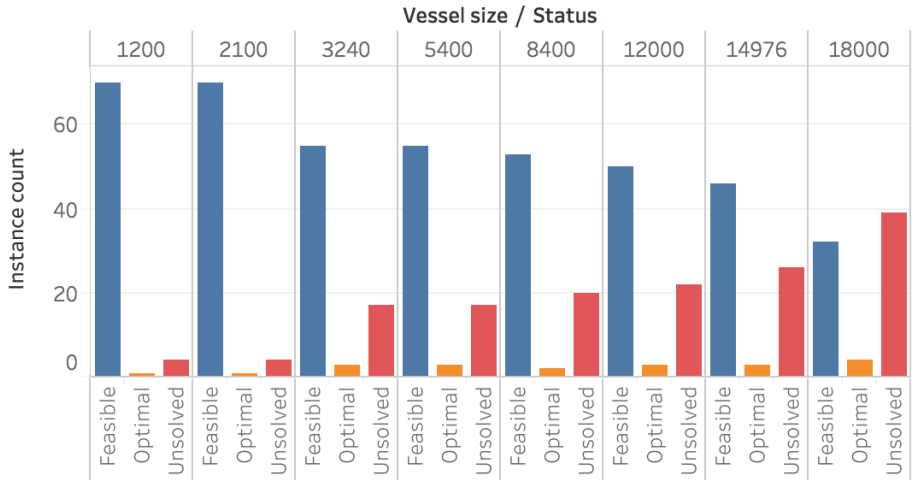


Fig. 3: Solved solutions by vessel size.

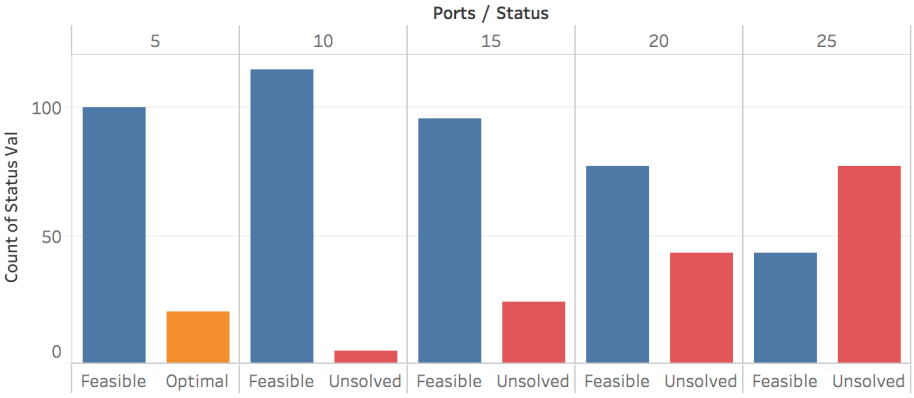


Fig. 4: Solved solutions by number of visited ports.

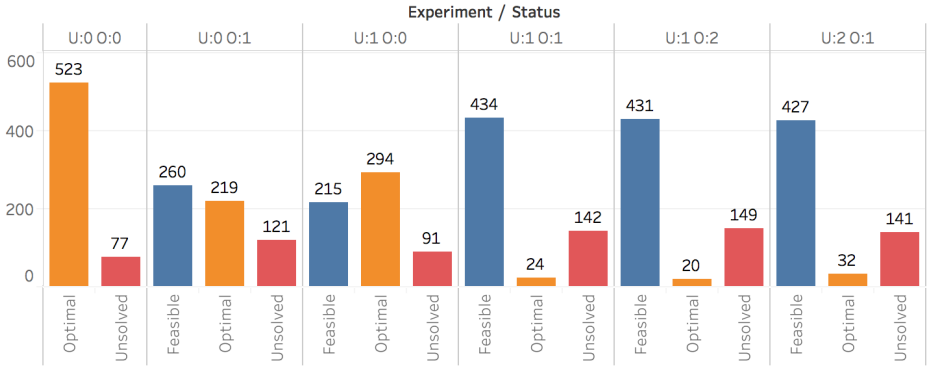


Fig. 5: Sensitivity analysis of the objective cost coefficients.

solutions than the other, which is to be expected due to the computational complexity of the problem. Notice, however, that the LNS is an anytime algorithm that is able to solve all the 149 instances that the compact formulation could not.

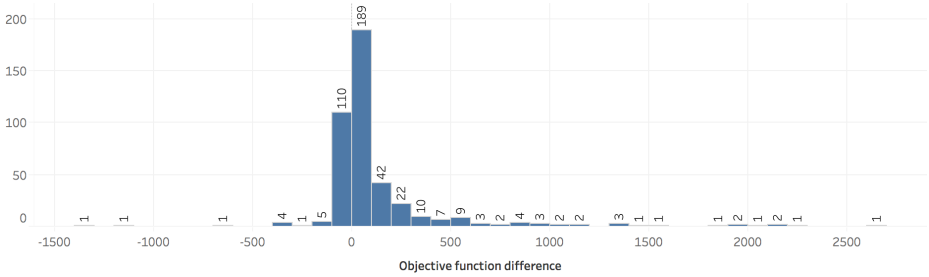
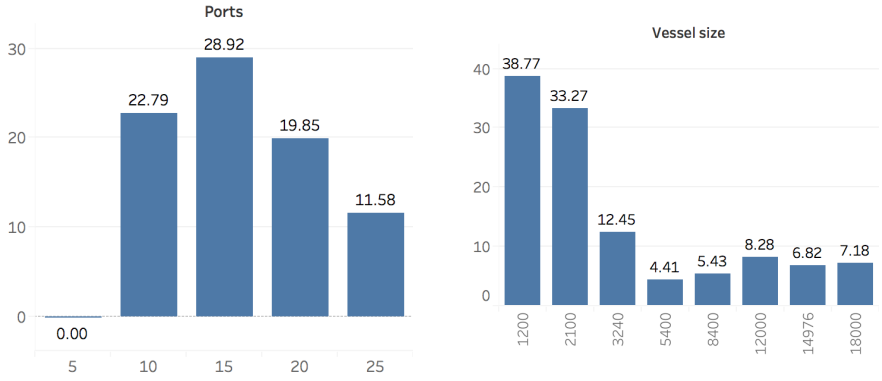


Fig. 6: Histogram over difference in objective function between the LNS and the campact model.

A more in-depth look at the experimental results confirms that the LNS becomes more efficient as the mathematical formulation starts degrading. Figure 7 shows two bar charts with the y-axis indicating the average gap between the LNS and the solution found the by the mathematical model. The changes are shown as a function of the vessel size (Figure 7a) and the number of visited ports (Figure 7b). Here we see the opposite tendency than the one shown in Section 4.1. As the size of the problem increases, the LNS reduces its gap since the compact formulation has a harder time finding solutions. Note that due to the outliers the average gap is not a good indication of the quality of the LNS. Figure 7 can be reproduced using the median as a measure, which will result in the same conclusion but where the highest median gap is only 10%.



(a) Based on number of visited ports. (b) Based on vessel size.

Fig. 7: LNS average gap to the solution found by the compact model.

5 Conclusions

In this paper, we have presented a new variation of the stowage planning problem which includes block stowage and crane intensity strategies. Though the problem was greatly simplified, due to its combinatorial nature, it has been proven hard to solve. A compact formulation was presented and a matheuristic based on the LNS framework was implemented to solve the problem. Experiments on a randomly generated set of benchmark instances have shown that the LNS can be used to quickly find solution comparable to those of the mathematical formulation. Further research is, however, needed to both improve the LNS approach and to extend it to a more rich problem definition.

Acknowledgments

The author would like to thank the Danish Maritime Foundation for supporting this research under the project 2015-119 DTU Transport, Dynastow. Thanks are also due to Roberto Roberti for the fruitful discussions about the mathematical formulations.

References

1. Ambrosino, D., Paolucci, M., Sciomachen, A.: A MIP Heuristic for Multi Port Stowage Planning. Transportation Research Procedia **10**, 725–734 (jan 2015)
2. Ambrosino, D., Paolucci, M., Sciomachen, A.: Computational evaluation of a MIP model for multi-port stowage planning problems. Soft Computing **21**(7), 1753–1763 (apr 2017)
3. Avriel, M., Penn, M., Shpirer, N., Witteboon, S.: Stowage planning for container ships to reduce the number of shifts. Annals of Operations Research **76**(0), 55–71 (1998)

4. Christensen, J., Pacino, D.: A matheuristic for the Cargo Mix Problem with Block Stowage. Transportation Research Part E: Logistics and Transportation Review **97**, 151–171 (jan 2017)
5. Ding, D., Chou, M.C.: Stowage planning for container ships: A heuristic algorithm to reduce the number of shifts. European Journal of Operational Research **246**(1), 242–249 (oct 2015)
6. Pacino, D., Delgado, A., Jensen, R.M., Bebbington, T.: Fast Generation of Near-Optimal Plans for Eco-Efficient Stowage of Large Container Vessels. In: Computational Logistics, pp. 286–301. Springer, Berlin, Heidelberg (2011)
7. Pacino, D., Delgado, A., Jensen, R.M., Bebbington, T.: An Accurate Model for Seaworthy Container Vessel Stowage Planning with Ballast Tanks. In: Computational Logistics, pp. 17–32. Springer, Berlin, Heidelberg (2012)
8. Parreño, F., Pacino, D., Alvarez-Valdes, R.: A GRASP algorithm for the container stowage slot planning problem. Transportation Research Part E: Logistics and Transportation Review **94**, 141–157 (oct 2016)
9. Pisinger, D., Ropke, S.: Large neighborhood search. In: Handbook of metaheuristics, pp. 399–419. Springer (2010)
10. Roberti, R., Pacino, D.: A decomposition method for finding optimal container stowage plans (2018), accepted manuscript to appear in Transportation Science
11. Shaw, P.: Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems, pp. 417–431. Springer Berlin Heidelberg (1998)
12. Tierney, K., Pacino, D., Jensen, R.M.: On the complexity of container stowage planning problems. Discrete Applied Mathematics **169**, 225–230 (may 2014)
13. UNCTAD: Review of maritime transport 2017 (2017)