Maritime Economics & Logistics (2021) 23:94–122
https://doi.org/10.1057/s41278-020-00172-5

ORIGINAL ARTICLE

A heuristic and a benchmark for the stowage planning problem

Rune Larsen[1] · Dario Pacino[1]

Accepted: 8 October 2020 / Published online: 21 October 2020

Abstract
The stowage planning problem has recently gained the attention of a number of academic researchers. Unfortunately, many of the published works are either based on oversimplified assumptions or on confidential data. This practice hinders the research field from growing. In this paper, we present a novel set of realistic vessel data along with a set of benchmark instances. Moreover, a formal definition of a single-port stowage planning problem, based on the current state of the art, is presented. The proposed optimisation problem is solved using a variant of the adaptive large neighbourhood search framework, where novel repair and destroy methods are presented. Computational results show that the solution approach is able to find high-quality seaworthy stowage plans within 60 s.

Keywords Stowage planning · Liner shipping · Benchmark · ALNS · Optimisation

1 Introduction

Container shipping operates on cyclical routes with published schedules. The cargo is transported on standard-sized metallic containers, which can carry both dry goods, liquids (in tank containers) and cargo with temperature requirements (in refrigerated containers). Each container is 8 foot wide and 8′6 foot high, although there are containers that are 9′6 foot high, called high-cube. Most containers are 20′, 40′ or 45′ long, and refrigerated containers (reefers) require power connections. To take advantage of the standardisation of containers, specialised vessels are used. Figure 1 shows the layout of a container vessel. As depicted, a vessel is divided into sections called bays. Each bay is divided into an above- and below-deck part, divided by a hatch cover (a watertight structure). Each bay is composed of a number of stacks. Container positions within a stack are called cells. A cell can hold either two 20- or

✉ Dario Pacino
darpa@dtu.dk

[1] DTU Management, Technical University of Denmark, Building 358, 2800 AkademivejKgs. Lyngby, Denmark

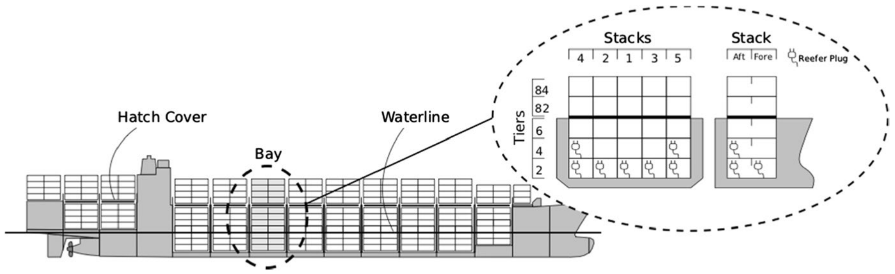

Fig. 1 Vessel longitudinal layout and bay details

one 40-foot container and are vertically indexed by tiers. Longer (e.g. 45-foot) containers are commonly stowed above-deck. Depending on the size of the vessel, each bay can have between one and four hatch covers. The set of stacks above, and below, each hatch cover are often grouped into logical partitions used during planning, and they are referred to as locations or blocks (see e.g. Ambrosino et al. 2018; Christensen and Pacino 2017). The industry uses a special numbering system to identify bays, tiers and stacks. Since this is not relevant here, we refer the interested reader to Ambrosino et al. (2004) for details.

Given a set of containers to load on a vessel (a loadlist), stowage planning is the problem of finding a profitable assignment of containers to cells ensuring the seaworthiness of the vessel (a stowage plan). Moreover, a stowage plan should minimise the time the vessel spends at port. Estimating port time in advance is not easy since it depends on how containers are handled by the terminal. Different kinds of approximations are used, e.g. in Ambrosino et al. (2004) it is assumed that a single loading crane is used, while in Pacino et al. (2011) a lower bound on the minimum port stay is optimised. A common estimator is the minimisation of overstowage, which happens when a container destined to a later port must be shifted in order to reach a container that has to be unloaded at the current port.

Stowage planners have traditionally performed their work manually. Currently, most stowage planning software available does not include any form of automated planning, although steps are currently taken in this direction (e.g. Doe 2018). Such automations are, however, becoming increasingly important as the size of the vessels, and their cargo volumes, grow. To boost research and encourage the development of innovative solutions, the problem of data availability must be overcome. With this paper we wish to start this innovative process and provide the means for stowage planning research to reach higher levels of excellence. This paper makes four major contributions. First, we provide vessel data representing three container vessels with capacity between 7000 and 15,000 TEUs. The data provide enough information to model vessel stability as seen in Pacino et al. (2012). The vessel profiles are obtained by extrapolating and simplifying data from industrial vessel profiles. The provided data does not correspond to any existing commercial vessel, but it still retains the stability quality of a real vessel. Second, a set of benchmark instances is made available. The instances are designed to resemble real-life stowage scenarios and are based on the demand distributions presented in Christensen and

Pacino (2017). Third, we present a mathematical formulation for a single port stowage planning problem where the objectives and constraints from Pacino et al. (2012) are modelled in detail. Finally, we propose a solution approach based on the combination of adaptive large neighbourhood search (ALNS) and dynamic programming to solve the proposed stowage planning problem efficiently. The computational results show that high-quality solutions can be found for all instances within a runtime of 60 s. The solution of each tested instance has resulted in seaworthy stowage plans.

The remainder of the paper is organised as follows: Section 2 gives an overview of the available literature, followed by Sect. 2.1 where the mathematical formulation and the details of the optimisation problem are described. The solution method is presented in Sect. 2.2. The proposed benchmark instances are described in Sect. 3, followed by the computational results in Sect. 4. Finally, conclusions are drawn in Sect. 5.

2 Literature review

Although it is still a small research community, the number of publications related to stowage planning has grown substantially in the past decade. Stowage planning research has had a rough start when one of the first mathematical models presented (Botter and Brinati 1991) could not find feasible solutions even for small-scale instances. Since then, the focus has been either on simplified versions of the problem (Avriel and Penn 1993; Avriel et al. 1998; Dubrovsky et al. 2002; Ding and Chou 2015; Roberti and Pacino 2018), on the computational complexity of its components (Avriel et al. 2000; Tierney et al. 2014) or on decomposition approaches (Wilson and Roach 1999; Kang and Kim 2002; Ambrosino et al. 2006; Li et al. 2017; Chao and Lin 2019).

The community has, however, been plagued by lack of a common benchmark (mainly due to the commercially sensitive nature of the required data). Some benchmarks are available, but they are related either to sub-problems (Delgado et al. 2012; Parreño et al. 2016) or to simplifications of the problem (Roberti and Pacino 2018; Pacino 2018). This lack of data has created two sub-communities, one focusing on studying single combinatorial issues on oversimplified stowage planning definitions (based on Avriel et al., 1998), and one aiming at studying rich problem definitions which are closer to actual industrial problems.

The first publications showing positive results on rich problem definitions were based on hierarchical decomposition approaches. In Ambrosino et al. (2006), we see the modelling of both 20- and 40-foot containers, weight limitations on the stacks and an initial attempt at vessel stability modelling. Vessels up to ca. 600 TEUs can be stowed using a mixture of mathematical programming and local search. Later on, a more advanced modelling of the vessel's stability constraints was presented in Pacino et al. (2011, 2012), where it was shown that detailed stability calculations can be linearised and included in linear mathematical formations, and that the decomposition approaches have the potential to solve industrial-size instances with vessels of up to 15,000 TEUs. The authors proposed a decomposition where a master planning phase would result in a distribution of containers to subsections of the vessel that guaranties seaworthiness (over several ports) and minimises time in port. A slot planning phase subsequently identifies the exact location of each container. As vessel stability data could not be publicly distributed, extensions of this work have been focusing on solution methods for the slot planning phase (Delgado et al. 2012; Parreño et al. 2016).

Recent work (Chou and Fang 2018) has also analysed the expertise of stowage planners. Another consequence of the lack of data is the adoption of simplified problem definitions (Ambrosino et al. 2004) as the standard for heuristic searches (Li 2012; Cruz-Reyes et al. 2013), where the quality of results can hardly be evaluated due to a lack of a common benchmark. It was not until 2018 (Ambrosino et al. 2018) that we saw the adoption of some of these more advanced stability considerations. In the past two decades, three patents have also been filed: one based on the use of evolutionary algorithms (Davidor 1998), another based on an integer programming framework (Guilbert and Paquin 2008), and one on case-based reasoning (Nugroho 2006). However, the quality of these approaches has not been published.

To the best of the authors' knowledge, the work presented in this paper is the first successful attempt at solving a large-scale rich stowage planning problem without a decomposition. In contrast to current research trends, we are focusing on a single port, meaning that we do not take into account cargo forecasts. On the other hand, a set of heuristic rules are used to achieve robust solutions.

A parallel community studies the capacity utilisation of container vessels. Among these, two works are particularly relevant to the stowage planning community (Delgado 2013; Christensen and Pacino 2017). In particular, Christensen and Pacino (2017) present a solution method that could be adapted to stowage planning where a block stowage strategy is used. Of interest are also the works where integrated solutions are sought between stowage planning and container terminal optimisation (Imai et al. 2006; Monaco et al. 2014; Iris et al. 2018), and the studies on the impact of IMO containers (Ambrosino and Sciomachen 2018; Parreño et al. 2016). Furthermore, the work of Helo et al. (2018) suggests a set of business-related key performance indicators (KPIs) for stowage planning, while also giving a broad presentation of the stowage planning process.

3 The single-port stowage planning problem

As previously mentioned, the stowage planning problem aims at finding the optimal stowage of containers on a vessel that minimises handling time in port. Such a definition is, however, too naïve compared with real-life practice. Evaluation of the quality of a stowage plan goes far beyond port-stay. Quality is defined differently, depending on where the stowage plan is evaluated within the organisation. In uptake management the goal is to accept as many bookings as possible and thus fully utilise the capacity of the vessel (Christensen and Pacino 2017). The stowage planner's main objective is to have a plan of seaworthy ship which takes into account unforeseen container loads in ports along the route. A wrong stowage decision at a given port can result in high costs in subsequent ports if e.g. a large number of overstowage moves need to be performed. From a terminal management point of view, an optimal stowage plan minimises not only time in port but also the use of terminal resources, e.g. crane and vehicle moves (Monaco et al. 2014; Iris et al. 2018). In this work we assume the point of view of a stowage planner and optimise two groups of performance KPIs. The first group has a higher priority and includes KPIs relative to the current planned port:

1. Hatch-overstowage (Pacino et al. 2011; Delgado et al. 2012)
2. Stack-overstowage (Delgado et al. 2012)
3. The number of unassigned containers (Parreño et al. 2016)
4. Makespan of adjacent bays (Pacino et al. 2011)
5. Metacentric height (based on Pacino et al. 2011)

Hatch-overstowage is the most costly kind of overstowage and requires the lifting of the hatch cover (and all the containers above it) in order to load or discharge a target container. Stack overstowage is a milder version where extra moves need to be performed due to blocking containers. The third KPI regards loadlists where not all containers can be stowed on the vessel due to stability issues. It is thus the stowage planner's job to load as much as possible. Finally, the makespan of adjacent bays is a lower bound on the time to complete the actual work by the quay cranes, and thus it is an indication of terminal efficiency when handling the ship.

In Pacino et al. (2011) it was stressed how important it is to take the subsequent container movements into account, and consequently plan multiple ports at once. This, however, requires a high-quality forecast which is currently not available in the industry. Stowage planners rely on their experience and the currently available bookings. Our work focuses on planning a single port at the time, but it accommodates future cargoes with a set of secondary KPIs:

6. Block purity (inspired by Pacino 2018)
7. Number of empty stacks (Delgado et al. 2012)
8. Number of non-reefers in reefer slots (Delgado et al. 2012)
9. Number of containers below deck

Block purity measures the distinct number of ports of destination of the containers within a block. A block is a logical partition of stacks corresponding to all the cells that are either above or below a hatch cover. Stowing containers in blocks results in more terminal-friendly stowage plans. KPIs 7 and 8 aim at finding robust plans. Keeping stacks empty leaves space for loading containers at the next port, avoiding the unnecessary blockage of reefer plugs. This opens the possibility of loading more reefer containers at later ports. The last KPI is based on a general rule used by stowage planners, aiming to achieve seaworthy plans. Favouring containers below deck will reduce the centre of gravity of the vessels, resulting in better stability and reducing the risk of hatch-overstowage in later ports.

Following Monaco et al. (2014) and Parreño et al. (2016), we work with container classes rather than individual containers (note, however, that the class definition can easily be expanded to individual containers). Let T be the set of container classes defined as $T = \{3,6,9,14,21,27\} \cup \{20,40\} \cup \{\mathrm{DC},\mathrm{RC},\mathrm{HC},\mathrm{HR}\}$. T is thus the cross product of container weights (tons), length (foot) and type: dry container (DC), reefer container (RC), high-cube container (HC), high-cube reefer (HR). The number of containers to be loaded on the vessel (the loadlist) is given by n_{pt}, where P is the set of discharge ports. The *release* (containers already onboard the vessel) is defined by the set R composed of tuples (c, l, t) where $c \in C$ is the cells, and $l \in L$ is the slot (Fore, Aft) in which a container of class $t \in T$ is stowed. With the main decision variable $x_{clt} \in \mathbb{B}$ taking value 1 iff a container of class t is stowed in cell c at position l, the following mixed-integer program formally describes the single port stowage planning problem under study (a table with the summary of the mathematical notation can be found in Appendix 1).

$$\min\ C^{\mathrm{load}} \sum_{t\in T} y_t + C^{\mathrm{hatch}} \sum_{c\in C, i\in H} ho_{ci} + C^{\mathrm{ov}} \sum_{c\in C, l\in L} o_{cl} + C^{\mathrm{Mk}} mk + C^{\mathrm{GM}} vm + C^{\mathrm{purity}} \sum_{i\in H, p\in P, q\in\{\text{above},\text{below}\}} bp_{ip}^{q} - C^{\mathrm{empty}}\left(|S| + \sum_{s\in S} e_s\right) + C^{\mathrm{reefer}} \sum_{c,\in C, l\in L} nr_{cl} - C^{\mathrm{below}} \sum_{c\in C, l\in L, t\in T} d_t \beta_c x_{clt} \tag{1}$$

s.t.

$$\sum_{t\in T} x_{c,\mathrm{AFT},t} \le 1 \ \forall c \in C \tag{2}$$

$$\sum_{t\in T^{20}} x_{c,\mathrm{FORE},t} \le 1 \ \forall c \in C \tag{3}$$

$$\sum_{t\in T^{40}} x_{c,\mathrm{FORE},t} = 0 \ \forall c \in C \tag{4}$$

$$\sum_{t\in T^{20}} \left(x_{c,\mathrm{AFT},t} - x_{c,\mathrm{FORE},t}\right) = 0 \ \forall c \in C \tag{5}$$

$$\sum_{c\in C, l\in L} x_{clt} = \sum_{p\in P} n_{pt} - y_t \ \forall t \in T \tag{6}$$

$$\sum_{c\in C_s} \left(\frac{1}{2} \sum_{t\in T^{20}, l\in L} w_t x_{clt} + \sum_{t\in T^{40}} w_t x_{c,AFT,t}\right) \le w_s^{\max 40} \ \ \forall s \in S \tag{7}$$

$$\sum_{c\in C_s} \left(\sum_{t\in T^{20}, l\in L} w_t x_{clt} + \frac{1}{2} \sum_{t\in T^{40}} w_t x_{c,AFT,t}\right) \le w_s^{\max 20} \ \ \forall s \in S \tag{8}$$

$$\sum_{c\in C_s} \left(\frac{1}{2} \sum_{t\in T^{20}, l\in L} h_t x_{clt} + \sum_{t\in T^{40}} h_t x_{c,AFT,t}\right) \le h_s^{\max} \ \ \forall s \in S \tag{9}$$

$$-r_c + \sum_{t\in TR, l\in L} x_{clt} \le 0 \ \forall c \in C \tag{10}$$

$$\sum_{t\in T} w_t x_{clt} \le \sum_{t\in T} w_t x_{c-1,l,t} \ \ \forall c \in C_s, s \in S, c > \mathrm{bot}(s), l \in L \tag{11}$$

$$x_{clt} = 1 \ \forall (c, l, t) \in R \tag{12}$$

$$\sum_{t\in T,d_t<p} x_{c'l't} \leq \delta_{csp} \ \forall s \in S, c \in C_s, c' \in C_s, l \in L, l' \in L, p \in P, c' < c \tag{13}$$

$$x_{clt} + \delta_{csd_t} \leq 1 + o_{cl} \ \forall s \in S, c \in C_s, t \in T, l \in L \tag{14}$$

$$x_{clt} + \sum_{t'\in T,d_{t'}<d_t} x_{c'l't'} \leq 1 + ho_{ci} \ \forall i \in H, c \in C_i^{\text{above}}, c' \in C_i^{\text{below}}, t \in T, l, l' \in L \tag{15}$$

$$x_{clt} + x_{c'l't'} \leq 1 + ho_{ci} \ \forall i \in H, c \in C_i^{\text{below}}, t \in T, l \in L, \left(c', l', t'\right) \in R_i^{\text{above}} \tag{16}$$

$$\sum_{s\in S_b\cup S_{b'},c\in C_s,l\in L,t\in T} x_{clt} \leq mk \ \forall \left(b, b'\right) \in A \tag{17}$$

$$x_{clt} \leq bp_{ip}^{q} \ \forall i \in H, q \in \{\text{above, below}\}, s \in S_i^q, c \in C_s, l \in L, t \in T, p \in P, d_t = p \tag{18}$$

$$x_{clt} \leq e_s \ \forall s \in S, c \in C_s, l \in L, t \in T \tag{19}$$

$$\sum_{l\in L,t\in T\setminus TR} x_{clt} \leq nr_c \ \forall c \in C, r_{cl} = 1 \tag{20}$$

Objective (1) represents a weighted sum of the KPIs presented above. The first term, with cost $\mathcal{C}^{\text{load}}$, minimises the number of non-loaded containers of class $t \in T$ (y_t). The next two terms represent the hatch-overstowage, with cost $\mathcal{C}^{\text{hatch}}$, and the stack-overstowage, with cost $\mathcal{C}^{\text{ov}}$, respectively. The former quantified by the variable h_{ci} (which is 1 iff cell $c \in C$ of hatch cover $i \in H$) holds a hatch-overstowing container. Similarly, stack-overstowage is indicated by variable o_{cl}, which is 1 iff cells c in slot $l \in L$ stows an overstowing container. The next two terms optimise the makespan, with cost $\mathcal{C}^{\text{Mk}}$ and variable mk, and the metacentric height, with cost $\mathcal{C}^{\text{GM}}$. The metacentric height is the distance between the metacentre and the centre of gravity of the vessel (points M and G in Fig. 2a). A stable vessel needs a minimum metacentric height, however, not all vessel profiles provide this value. To alleviate this issue, the vertical moment of the vessel (vm) is minimised. The remaining four terms model the secondary KPIs. Variable bp_{ip}^{q} is 1 iff the block above/below

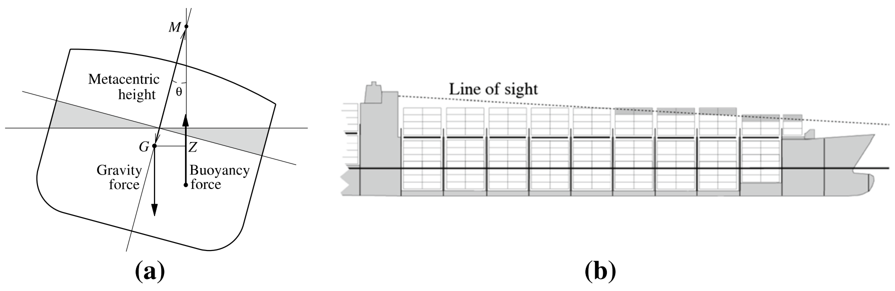

Fig. 2 Vessel stability and container stacking

($q \in \{\text{above}, \text{below}\}$) hatch cover i stows containers destined to port $p \in P$. Each distinct discharge port assigned to the block is penalised with cost $\mathcal{C}^{\text{purity}}$, thus representing the block purity KPI. Stacks stowing containers are indicated by the binary variable e_s for stack $s \in S$. The total number of empty stacks is then maximised using a cost $\mathcal{C}^{\text{empty}}$. The stowage of non-reefer containers in reefer slots is indicated by the binary variable nr_{cl} for each cell c in slot l, and has a cost $\mathcal{C}^{\text{reefer}}$. The last term corresponds to the last KPI where we want to incentivise containers going further on the route to be stowed as low as possible. This is done by imposing a negative cost $\mathcal{C}^{\text{below}}$ to each container. The cost increases linearly depending on the discharge port d_t of container class t below deck, indicated by $\beta_c \in \mathbb{B}$ for cells c.

Constraints (2–4) ensure that a cell can only hold either one 40-foot container or two 20-foot containers, where T^{20} and T^{40} are the sets of 20- and 40-foot container classes, respectively. Forty-foot containers are assumed to be always assigned to an Aft slot [see (2) and (4)]. Furthermore, we assume that a 20-foot container cannot be stowed alone in a cell, thus a matching container must be found. This is ensured by constraint (5). Constraint (6) defines the y_t variables by collecting the number of containers of class $t \in T$ that could not be loaded from the loadlist. The loadlist is represented by n_{pt}, which indicates the number of containers of class t with discharge port $p \in P$ to be stowed in the vessel (including the release containers).

Constraints (7) and (8) model the weight limits of the stacks. Each stack has 8 corner castings to support the stowed containers, however, only 4 of them are used when 40-foot containers are stowed. For that reason, the weight of a stack is distributed differently, and thus two stack weight limits exist: $w_s^{\max 40}$ and $w_s^{\max 20}$ for each stack $s \in S$ (see Pacino et al. 2011 for more information). The weight of a container class is represented by w_t, and C_s is the set of cells belonging to stack s. The maximum height of a stack ($h_s^{\max}$) is ensured by constraint (9), where h_t is the height of container class t. This height limit is established below deck by the hatch cover, and above deck by the captain's line-of-sight or the crane height of the next port of call (see Fig. 2b). Ensuring that reefer containers are stowed where electric power is available is done using constraint (10), where TR is the set of reefer container classes, and r_c is the number of power plugs available in cell c (0,1, or 2). In order to ease container lashing, we enforce a sorting on the weight of the stowed containers with constraint (11), where bot(s) indicates the cell at the bottom of stack s. Constraint (12) ensures that release containers are assigned to their current stowage position.

Constraints (13–20) define the auxiliary variables used for the definition of the objective function. Constraint (13) defines the variable δ_{csp}, which is 1 iff a container with discharge port less than $p \in P$ is stowed in any cell of stack $s \in S$ below cell $c \in \mathcal{C}_s$. The discharge port of a container class $t \in T$ is defined by d_t. This auxiliary variable is then used in constraint (14) to identify overstowing containers. Should an overstowing container be found in cell c in slot l, the variable o_{cl} takes the value of 1, and 0 otherwise. Hatch-overstowage is modelled using two constraints. Constraint (15) assigns ho_{ci} the value of 1 iff cell c, above hatch cover $i \in H$ (C_i^{above}), stows a container that overstows any container in the cells below the hatch cover (C_i^{below}). This is, however, not enough as the hatch cover might be forced to be lifted should we assign containers below deck under a hatch cover holding release containers. This is modelled with constraint (16), where R_i^{above} is the set of release containers above hatch cover i.

The crane makespan is calculated with constraint (17), where the set A is composed of pairs of adjacent bays, and S_b is the set of stacks belonging to bay b. This can easily be extended to triplets of bays, should the terminal require two bays' distance between quay cranes. Constraint (18) models the block purity KPI. A block is defined by the set of stacks S_i^q above/below (q) hatch cover $i \in H$. The constraint forces the variable bp_{ip}^q to be 1 if any container with discharge port $p \in P$ is stowed in a cell belonging to the considered block. With constraint (19) we assign the value 1 to the variable e_s, should any container be stowed in stack $s \in S$, thus identifying used stacks. The stowage of non-reefer containers in reefer slots is modelled with constraint (20), where r_{cl} indicates the presence of a power plug in cells $c \in C$, slot $l \in L$.

The model presented until now, deals only with the basic stacking constraints and the objectives. Following are the constraints needed to ensure the seaworthiness of the stowage plan:

$$\sum\nolimits_{t \in T, l \in L, s \in S_b, c \in C_s} w_t x_{clt} = w_b^{\text{bay}} \quad \forall b \in B \tag{21}$$

$$\sum\nolimits_{b \in B} \left(w_b^{\text{const}} + w_b^{\text{bay}} \right) = w^{\text{disp}} \tag{22}$$

$$\sum\nolimits_{o \in O} \lambda_o = 1 \tag{23}$$

$$\sum\nolimits_{o \in O, o > 1} \text{disp}_{o-1} \lambda_o \leq w^{\text{disp}} \leq \sum\nolimits_{o \in O, o > 1} \text{disp}_o \lambda_o \tag{24}$$

$$\sum\nolimits_{b \in B} \text{lcg}_b \left(w_b^{\text{const}} + w_b^{\text{bay}} \right) = lm \tag{25}$$

$$\sum\nolimits_{o \in O} \text{lcg}_o^{\min} \text{disp}_o \lambda_o \leq lm \leq \sum\nolimits_{o \in O} \text{lcg}_o^{\max} \text{disp}_o \lambda_o \tag{26}$$

$$\sum\nolimits_{b \in B} \left(vcg_b w_b^{\text{const}} + \sum\nolimits_{s \in S_b, c \in C, l \in L, t \in T} vcg_s w^t x_{clt} \right) = vm \tag{27}$$

$$\sum\nolimits_{s \in S, c \in C, l \in L, t \in T} \text{tcg}_s w_t x_{clt} = tm \tag{28}$$

$$\text{tcg}^{\min} \sum\nolimits_{o \in O} \text{disp}_o \lambda_o \leq tm \leq \text{tcg}^{\max} \sum\nolimits_{o \in O} \text{disp}_o \lambda_o \tag{29}$$

$$by_b \geq \left(\frac{\text{disp}_o - w^{\text{disp}}}{\text{disp}_o - \text{disp}_{o-1}} \right) \text{buoy}_{bo-1} + \left(1 - \frac{\text{disp}_o - w^{\text{disp}}}{\text{disp}_o - \text{disp}_{o-1}} \right) \text{buoy}_{bo} - \left(1 - \lambda_o \right) M \tag{30}$$

$$by_b \leq \left(\frac{\text{disp}_o - w^{\text{disp}}}{\text{disp}_o - \text{disp}_{o-1}} \right) \text{buoy}_{bo-1} + \left(1 - \frac{\text{disp}_o - w^{\text{disp}}}{\text{disp}_o - \text{disp}_{o-1}} \right) \text{buoy}_{bo} + \left(1 - \lambda_o\right) M \tag{31}$$

$$\text{shear}_b^{\min} \leq \sum\nolimits_{b'=1}^{b} \left(w_{b'}^{\text{const}} + w_{b'}^{\text{bay}} - by_b \right) \leq \text{shear}_b^{\max} \ \ \forall b \in B \tag{32}$$

$$\sum\nolimits_{b'=1}^{b} \left|\text{lcg}_1 - \text{lcg}_b\right| \cdot \left(w_{b'}^{\text{const}} + w_{b'}^{\text{bay}} - by_b \right) \leq \text{bend}_b^{\max} \ \ \forall b \in B \tag{33}$$

$$x_{clt} \in \mathbb{B} \ \forall c \in C, l \in L, t \in T \tag{34}$$

$$y_t \in \mathbb{N} \ \forall t \in T \tag{35}$$

$$ho_{ci} \in \mathbb{B} \ \forall c \in C, i \in H \tag{36}$$

$$o_{cl} \in \mathbb{B} \ \forall c \in C, l \in L \tag{37}$$

$$bp_{ip}^{q} \in \mathbb{B} \ \forall i \in H, p \in P, q \in \{\text{above,below}\} \tag{38}$$

$$e_s \in \mathbb{B} \ \forall s \in S \tag{39}$$

$$nr_{cl} \in \mathbb{B} \ \forall c \in C, l \in L \tag{40}$$

$$mk \in \mathbb{R}^+ \tag{41}$$

$$vm \in \mathbb{R}^+ \tag{42}$$

$$by_b \in \mathbb{R}^+ \ \forall b \in B \tag{43}$$

$$\delta_{csp} \in \mathbb{B} \ \forall c \in C, s \in S, p \in P \tag{44}$$

$$w_b^{\text{bay}} \in \mathbb{R}^+ \ \forall b \in B \tag{45}$$

$$w^{\text{disp}} \in \mathbb{R}^+ \tag{46}$$

$$\lambda_o \in \mathbb{B} \ \forall o \in O \tag{47}$$

$$tm \in \mathbb{R} \tag{48}$$

The weight of a bay $b \in B$, represented by the variable w_b^{bay}, plays an important role when calculating the stability of the vessel; this is defined in constraint (21) where S_b is the set of stacks belonging to bay b. Constraint (22) defines the displacement of the vessel (variable w^{disp}) as the total weight of the cargo and the weight of the vessel itself. Notice that the weight of the vessel (lightship) is represented as a constant weight at each bay (w_b^{const}). The stability of a containership is calculated on the basis of a set of data points called hydrostatic data. Hydrostatic data functions as a lookup table, where, given the displacement and the centre of gravity of the vessel, it is possible to read trim (the angle in which the ship sits in the water), draft (the depth reached by the vessel) and buoyancy (the water displacement).

Thus, it is important to identify the displacement level of the vessel, compared to the hydrostatic data. This is done with constraints (23) and (24). With the first, (23), we define an auxiliary binary variable λ_o for each displacement data point $o \in O$, and enforce that only one of them can be active. Constraint (24) is then used to set λ_o at the correct hydrostatic data point, where disp_o is the displacement value of data point o. Since we assume that the vessel needs to be at port with trim zero, we can constrain the longitudinal centre of gravity of the vessel to be within a certain bound. To do so, we first calculate the longitudinal moment of the vessel (lm) with constraint (25), where lcg_b is the longitudinal centre of gravity of bay b. Then, with constraint (26), we force lm to be within the bounds dictated by the maximum and minimum centre of gravity ($\text{lcg}_o^{\max}, \text{lcg}_o^{\min}$) for the reached data point o. Constraints (27) and (28) define the vertical (vm) and transversal moment (tm) of the vessel, respectively. Here vcg_b represents the vertical centre of gravity of the constant weight of bay b, while vcg_s and tcg_s are the initial vertical and transversal centre of gravity of stack $s \in S$. The vertical moment is part of the objective function (1), while the transversal moment must be kept within bounds ($\text{tcg}^{\max}, \text{tcg}^{\min}$) with constraint (29).

Constraints (30) and (31) define the by_b variable representing the buoyancy at bay $b \in B$, where buoy_{bo} is the buoyancy at bay b for the hydrostatic data point $o \in O$, and M is a large number. These constraints make a linear interpolation of the hydrostatic data in order to achieve a precise buoyancy value. Constraints (32) and (33) are used to keep shear forces and bending limits at bay. Shear forces are the opposing forces that affect the structure of the vessel. These opposing forces are the difference between the accumulated weight of the cargo and the buoyancy. Bending moments, on the other hand, are the forces tending to bend the structure of the vessel. Constraint (32) calculates the shear force at each bay b and limits it to be within the bounds $\text{shear}_b^{\max}$ and $\text{shear}_b^{\min}$. In a similar matter, constraint (33) calculates the bending moment at bay b, and limits it to be below $\text{bend}_b^{\max}$. Constraints (34–48) define the domain of the variables.

4 Solving the stowage planning problem

The mathematical model presented in the previous section is but a formal description of the problem. It has not been possible to solve the model for even the smallest of instances in our benchmark. The sheer size of the problem does not even allow the calculation of a lower bound within 1 h of computation. In order to solve the problem, we propose a novel adaptive large neighbourhood search (ALNS). The

ALNS was initially applied to vehicle routing problems (Ropke and Pisinger 2006), and later it was successfully used in a wide range of other applications (e.g. Iris et al. 2017; Muller et al. 2012). Given an initial solution, the ALNS is based on the principle of destruction and reconstruction of the solution. A set of destroy methods are used to relax the variable assignment of a part of the solution. The resulting partial solution is then re-constructed using one or several repair algorithms. Since the ALNS is a well-known algorithm, we will focus on describing the components designed for solving the stowage planning problem. The interested reader can find details of the algorithm in the original publication (Ropke and Pisinger 2006).

4.1 The concept of candidate stacks

The creation of an initial solution and the subsequent use of the repair methods are based on the concept of candidate stacks. Consider a vessel consisting of a set of empty stacks. To find a stowage plan, all containers in the loadlist are used to generate all possible stack configurations. All that remains to be done is to select which stack configuration belongs to which stack in the vessel such that all constraints are fulfilled. Naturally, this is not a viable solution as there exists an exponential number of stack configurations. However, the concept can still be used.

In order to reduce the number of generated candidate stacks, we enforce a sorting on the containers to load. Consider the example in Fig. 3, where three containers need to be loaded into a single stack. The number of each container represents an arbitrary ordering. Starting from container 1, only one feasible stack can be generated; the one where the container is at the bottom. Next, we move to container 2. Here we add two new candidate stacks, one with container 2 at the bottom, and one where container 2 is in the same stack as container 1. We continue the same procedure with container 3 and obtain a total of seven candidate stacks. Out of these seven candidates stacks, we can select one which can be part of our solution.

Not all possible candidate stacks need to be generated, as a large number of them can be discarded due the constraints imposed by the problem. We generate the set of candidate stacks using a standard labelling algorithm (a form of dynamic programming). Each label *l* has the following layout:

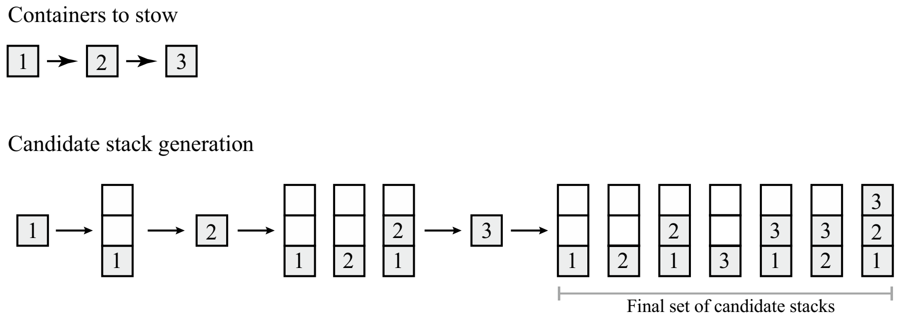

Fig. 3 Example of candidate stack generation

$$\left[W_l^{40}, W_l^{20\text{Fore}}, W_l^{20\text{Aft}}, h_l, C_l^{\text{Fore}}, C_l^{\text{AFT}}, R_l^{\text{Fore}}, R_l^{\text{AFT}} \middle| \text{dipo}_l, \text{prev}_l\right]$$

where the first three hold the accumulated weight of a stack on the 40-foot holds, and the 20-foot holds, respectively. The next component indicates the accumulated height, while the four last components of the first part of the layout are the capacity count and the reefer count. The second part of the layout indicates the discharge port of the latest inserted container, while the last component is a link to the previous label.

As mentioned above, we assume a container ordering when generating labels. We use a lexicographical order where containers are sorted first by length (20- before 40-foot), then by weight (heavy first) and finally by discharge port (descending). The label expansion, and consequently the candidate stack generation, is better explained using an example. We extend the example from Fig. 3 and assume all containers to be 40-foot. Following the described sorting, the three containers have the following characteristics:

Container 1: Weight: 18 tons. Destination port 2.
Container 2: Weight: 16 tons. Destination port 3.
Container 3: Weight: 16 tons. Destination port 2.

Label generation always starts with a zero-label representing a candidate stack with no containers:

$$1 : [0,0,0,0,0,0,0,0|0,0]$$

The number before the label represents its ID. We now extend this label with the first container:

$$2 : [18,9,9,8'6'',1,1,0,0|2,1]$$

Two more labels are then generated when the second container is introduced:

$$3 : [16,8,8,8'6'',1,1,0,0|3,1]$$
$$4 : \left[34,17,17,17',2,2,00\middle|3,2\right]$$

Label 3 is similar to label 2 and represents a candidate stack holding only container 2. Label 4 is more interesting as it represents the candidate stack which includes both containers; here we can see how the components of the label represent the accumulated weight, height and capacity of the candidate stack. Notice also that label 4 refers back to label 2 ($\text{prev}_4 = 2$), allowing us to trace back to all the containers stowed in the candidate stack represented by this label. Finally, we extend the set of labels in the same manner with container 3:

$$5 : [16,8,8,8'6'',1,1,0,0|2,1]6 : \left[34,17,17,17',2,2,0,0\middle|2,2\right]$$
$$7 : [32,16,16,17',2,2,0,0|2,3]$$
$$8 : [50,25,25,25'6'',3,3,0,0|2,4]$$

Notice how label 8 traces back to container 1 through label 4 and 2. The candidate stacks are simply lists of containers and do not refer to any particular stack of the vessel, however, we can limit the list of containers based on e.g. the stack with the most capacity, as we know that it would be impossible to assign more containers in any of the stacks of the vessel. The same reasoning can be made for the weight and height limits.

4.2 Stack matching and ship stability

Once a list of candidate stacks is available, it needs to be matched with the actual stacks of the vessel, which means finding stacks that can accommodate the container stacking patterns created in the candidate stack list. Obviously, not all patterns can be matched. Assume e.g. a vessel composed of only one stack that can hold at most three 40-foot containers. If a release container is already present in the stack, it would be infeasible to match label 8, as it will result in four stowed containers. The full matching procedure is outlined in Algorithm 1.

Algorithm 1 Candidate stack matching

```
 1: fail=0
 2: while there are unassigned containers AND fail!=failMax
 3:     $Stacks$ = select $n$ stacks at random
 4:     $Conts$ = select $m$ unassigned containers
 5:     generateLabels($Conts$)
 6:     $score_{sl}$ = score of matching stack $s$ with label $l$
 7:     $(s,l)$ = best stack/label matching
 8:     if $solution(s,l)$ is improving
 9:         commit($solution(s,l)$)
10:     else
11:         fail++
12:     end
13: end
14: update vessel status
```

The aim of the algorithm is to stow all the unassigned containers through the use of the candidate stacks. It starts by initialising a fail flag (line 1) used to terminate the algorithm (e.g. if it is not possible to load all containers). The procedure then iterates (line 2) as long as there are unassigned containers, or a fail limit is reached. Next, we generate the set of candidate stacks (line 5). We do this only using a random subset of stacks and containers to limit the size of the set (line 3–4). For each of the selected stacks and each of the candidate stacks we calculate a matching score. The score is based on the impact the resulting stack will have on the solution in terms of objective function, but also on the impact the new stack has on the stability of the vessel (line 6). Out of all the candidate stacks, we select the match with the best score (line 7). The solution resulting from this match is evaluated (line 8). If it improves the current objective, it is accepted (line 9), if not, the fail flag is incremented by 1 (line 11). Once the procedure has finished matching all possible stacks (line 13), the state of the vessel is updated (line 14). This means that values related to the stability of the vessel are updated based on the newly created solution.

To better understand the matching algorithm, consider the example in Fig. 4. To simplify the explanation we assume a vessel composed of 2 stacks which has one release container in the second stack. Figure 4 shows two solution examples: (a) matches the last candidate stack with the first stack from the vessel, while (b) matches two candidate stacks, and shows how stacks with release containers can be combined with candidate stacks.

4.3 Matching score calculation

The score calculation used in Algorithm 1 consists of two parts and can be formally described as follows:

$$\text{score}_{sl} = \sum_{i=1}^{9} \omega_i \Delta \text{Obj}_i + \sum_{j=1}^{7} \omega_{9+j} \vartheta_j$$

where ΔObj_i is the impact the container matching will have on each of the nine objective components described earlier, and where ϑ_j is an indication of the changes in vessel stability. The seven stability indicators are: maximum and minimum transversal moment, maximum and minimum longitudinal moment, maximum and minimum shear, and maximum bending. The scalar vector $\bar{\omega}$ is a parameter indicating the weight of each score component. Below we explain how this can be used within the solution method. The impact of the objective components can be trivially computed. The same cannot be said about the stability constraints, as changes in one stack have an impact on the entire vessel. For this reason, we compute an approximation using the following:

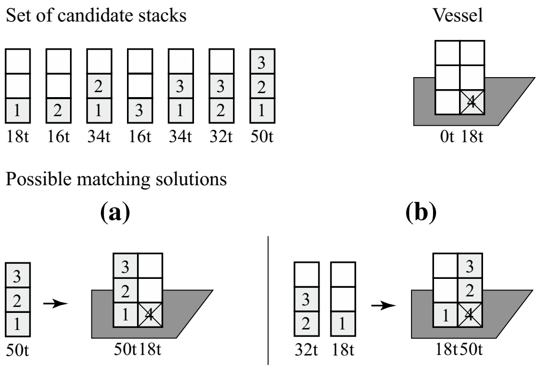

Fig. 4 Candidate stacks matching example

$$\vec{s} = \begin{array}{c}\text{Limits}\\ \begin{pmatrix} -tm^+ \\ tm^- \\ -lm^+ \\ lm^- \\ 0 \\ 0 \\ 0 \end{pmatrix}\end{array} + \begin{array}{c}\text{Ship}\\ \begin{pmatrix} tm \\ -tm \\ lm \\ -lm \\ 0 \\ 0 \\ 0 \end{pmatrix}\end{array} + \begin{array}{c}\text{Bay}\\ \begin{pmatrix} 0 \\ 0 \\ lm_b \\ -lm_b \\ -\Delta\text{shear}^+ \\ \Delta\text{shear}^- \\ -\Delta\text{bending}^+ \end{pmatrix}\end{array} \cdot \begin{array}{c}\text{Stack}\\ \begin{pmatrix} tm_s \\ -tm_s \\ 1 \\ 1 \\ 1 \\ -1 \\ 1 \end{pmatrix}\end{array} \cdot \begin{array}{c}\text{Candidate Stack}\\ \begin{pmatrix} \Delta w \\ \Delta w \\ \Delta w \\ \Delta w \\ \Delta w \\ \Delta w \\ \Delta w \end{pmatrix}\end{array}$$

$$\bar{\vartheta} = \max(\vec{0}, \vec{s})$$

We create an impact vector by multiplying the weight change in the candidate stack (Δw) with the transversal moment of the stack (tm_s), the longitudinal moment of the bay (lm_b) and the slack in the shear and bending limits at the bay ($\Delta\text{shear}_b^+, \Delta\text{shear}_b^-, \Delta\text{bending}_b^+$). Adding the resulting vector to the transversal and longitudinal moment of the vessel (tm, lm), and the limits on these moments ($tm^\pm, lm^\pm$), we obtain the vector $\vec{s}$. Positive elements of this vector indicate a violation of vessel stability, while negative values indicate slack in the limits. In order to calculate the actual $\bar{\vartheta}$ score we need to collect all the positive values. $\bar{\vartheta}$ is an approximation since we are only calculating a local and not a global shear and bending impact.

4.4 Creating an initial solution

The creation of the initial solution and the repair methods explained below are based on the matching algorithm (Algorithm 1) just described, where the set of containers to load corresponds to the entire loadlist. As a particularity, the creation of the initial solution utilises a customised scalar vector $\bar{\omega}$ during the score calculation:

$$\bar{\omega} = \begin{matrix}[10 \cdot \mathcal{C}^{\text{hatch}}, \mathcal{C}^{\text{ov}}, 1000 \cdot \mathcal{C}^{\text{load}}, \mathcal{C}^{mk}, \mathcal{C}^{\text{GM}}, 10 \cdot \mathcal{C}^{\text{purity}}, \mathcal{C}^{\text{empty}}, \mathcal{C}^{\text{reefer}}, 10 \cdot \mathcal{C}^{\text{below}}, \\ 0.1, 0.1, 0.1, 0.1, 5, 50, 0.1]\end{matrix}$$

It should be noted that emphasis is given to the hatch overstowage, block purity objective and the below-deck objective, as they are harder to fix incrementally. In order not to overly constrain the initial solution, a low impact is given to the limits of the transversal, longitudinal and bending moment. The limits of the shear forces do, however, need to be taken into consideration so that the solution can easily become feasible in the next iterations of the heuristic. For this procedure we restrict the algorithm to work with a set of 50 (n) stacks and 25 (m) containers.

4.5 Destroy methods

Destroy methods focus on portions of the solution that will most likely need to be changed in order to achieve a solution quality improvement. Initially, attention was paid to methods targeting the feasibility of the stability constraints, however, this turns out not to be a problem, and thus the methods described below focus only on reaching quality improvements.

4.6 Destroy *n* stacks

The n stacks are chosen uniformly at random. Only stacks stowing containers to load can be selected. This means that a stack consisting of only release containers is not part of the candidate list. This ensures that enough empty stacks are available for re-allocation. We consider three versions of this destroy, one for each $n \in \{1,3,6\}$.

4.7 Destroy small stacks

This destroy method is limited to only *small* stacks, i.e. stacks that hold at most four non-release containers. One stack is selected uniformly at random. Then, the method sequentially goes through all the small stacks in the same bay and selects stacks until at least five containers have been selected. Should the bay not have enough small stacks, the search continues to the next bays. This allows combining small stacks into bigger ones.

4.8 Destroy similar stacks

One stack is selected uniformly at random. The method then selects $n = 6$ random stacks of which the top container has the same discharge port as the one from the first selected stack. This aims at allowing the combination of stacks with the same discharge ports.

4.9 Destroy *n* containers

Stacks are chosen uniformly at random until at least n non-release containers have been removed from the solution. Compared with the previous destroy method, this one ensures that enough containers are available for re-allocation. This is especially important when the stacks are composed of few containers. We consider two versions of this destroy, one for each $n \in \{10,20\}$.

4.10 Destroy hatch cover

Selects a hatch cover uniformly at random, and unassigns all non-release containers above or below the hatch cover up to a maximum of $n = 10$ containers. The main purpose of this destroy method is to facilitate the reduction of hatch overstowage.

4.11 Repair methods

Similar to the way we created the initial solution, we use the matching algorithm (Algorithm 1) as a repair procedure. The difference is that now we do not have the entire loadlist, but only the set of containers that have been relaxed from the solution by one of the destroy methods, or containers that remain unassigned. We use a total of four repair methods. All of them use the original score weight on the objectives but have a dominating cost on the stability scores unless otherwise stated:

$$\bar{\omega} = [\ldots, 1000, 1000, 1000, 1000, 50000, 500000, 1000]$$

With this score weight we ensure that once a feasible solution is reached, the algorithm will most likely not revert towards infeasible solutions.

4.12 No overstowage repair

This repair only allows the generation of candidate stacks that do not possess overstowing containers.

4.13 Overstowage repair

The same as the above, but allowing the generation of candidate stacks with overstowage. To speed up the repair operation, only 20 containers (m) at a time are considered during the candidate stack generation.

4.14 Relax LCG repair

This repair method uses the same setting as the one above, but with the difference that, here, the longitudinal moment limits are down-prioritised, thus having a score of 0.01. This is necessary to diversify the search.

4.15 Relax stability repair

This repair method is the same as the one above, but with the difference that, here, all stability factors have a score of 0.01. This ensures that good solutions can be found even though reaching them might require temporarily breaking the stability constraints.

4.16 Acceptance criteria, termination and restart strategy

The implemented ALNS uses the standard acceptance criteria from simulated annealing. With a starting temperature of 500 and an ending temperature of 1, we use a geometric cooling schedule that ensures the convergence of the algorithm within a prescribed time limit. Let T_{start} and T_{end} be the start and end temperature. It is possible to derive the coefficient α using the following equation:

$$T_{\text{start}} \cdot \alpha^{\text{timeLimit}} = T_{\text{end}}$$

where timeLimit is the maximum runtime of the algorithm. Within each iteration the current annealing temperature t is then given by

$$t = T_{\text{start}} \cdot \alpha^{\text{currTime}},$$

where currTime is the time currently elapsed.

Moreover, the ALNS keeps track of the best solution. If the heuristic does not find improving solutions for a total of five iterations, the search restarts from the best known solution.

5 Benchmark instances

Based on our experience from industrial collaborations, we have created a public benchmark. Working with vessel data requires a high degree of knowledge about the architecture of the vessel. The calculation of stress forces is done at specific points called frames, while the calculation of the buoyancy of the vessel is done through another set of points called stations. These are just two examples showing the level of detail that is needed in order to handle the implementation of stowage planning software. To make the data more accessible, we have undertaken most of the data processing work and now we are able to deliver vessel profiles where all stability constraints are related to the weight of the bays of the vessel. We have done so by creating fictitious vessel profiles and approximating their hydrostatic data (stability data) using real data from similar vessel layouts. The resulting vessels are not real but do hold all the same characteristics as real vessels. Three vessel profiles have been generated, and for each of them we generate nine instances; three with low utilisation, three with medium utilisation and three with high utilisation. The loadlists were generated using the realistic container distribution data presented in Christensen and Pacino (2017). Since container vessels are never empty, we launched a simulation round where we used our heuristic to stow the vessel. After the vessel has visited at least seven ports, we collect the containers on board the vessel and assign them to the release. This is necessary as randomised container generation would most likely result in infeasible or highly unrealistic instances. The entire benchmark set can be found at [https://doi.org/10.11583/DTU.9916760], where it will be maintained and extended. Table 1 presents the characteristics of each generated instance.

6 Computational results

The developed solution approach was implemented in Linux using C++, and all experiments have been run on a Core i7-4790 K 4 GHz with 16 GB of RAM. To improve the performance of the ALNS, we implemented a simple parallelisation scheme. A total of 4 parallel threads run the ALNS independently. At the end of each iteration, the best known solution is shared among the threads.

Table 1 Properties of the benchmark instances

Vessel	Cap	Plugs	Util	Inst	Loadlist	L. reefers	40-Foot	20-Foot	Release	Total
Small	7300	770	Low	1	1953	126	760	433	2583	4536
				2	1447	59	608	231	2858	4305
				3	1528	45	662	204	3403	4931
			Medium	4	2399	111	999	401	1896	4295
				5	2060	59	875	310	3343	5403
				6	1791	28	797	197	3730	5521
			High	7	1634	31	672	290	3746	5380
				8	2403	51	962	479	3372	5775
				9	1098	18	394	310	4662	5760
Medium	9920	921	Low	10	1519	23	620	279	3864	5383
				11	2206	113	930	346	2822	5028
				12	2120	37	873	374	4670	6790
			Medium	13	3399	107	1396	607	4334	7733
				14	3613	59	1478	657	4636	8249
				15	5273	221	2234	805	1764	7037
			High	16	3590	34	1605	380	5762	9352
				17	3350	31	1498	354	5910	9260
				18	3136	22	1425	286	6364	9500
Large	15,370	1144	Low	19	2486	76	1026	434	6304	8790
				20	1931	41	798	335	5700	7631
				21	3039	158	1220	599	6118	9157
			Medium	22	3920	74	1675	570	6140	10,060
				23	3673	59	1511	651	7038	10,711
				24	971	11	349	273	8434	9405
			High	25	8365	383	3544	1277	3734	12,099
				26	4896	88	1961	974	7350	12,246
				27	4745	96	1914	917	6992	11,737

Vessel vessel type, *Cap.* capacity (TEUs), *Plugs* number of reefer plugs, *Util.* level of vessel utilisation, *Inst.* instance number, *Loadlist* number of containers to load (TEUs), *L. reefer* number of reefer containers in the loadlist (units), *40-Foot* number of 40-foot containers in the loadlist, *20-Foot* number of 20-foot containers in the loadlist, *Release* number of release containers (TEUs)

The costs associated with the objective function have been assigned following suggestions from literature, and they have been adjusted by qualitatively assessing the stowage solutions:

$$\mathcal{C}^{\text{hatch}} = 100, \mathcal{C}^{\text{ov}} = 100, \mathcal{C}^{\text{load}} = 1000, \mathcal{C}^{\text{reefer}} = 5, \mathcal{C}^{\text{empty}} = 10,$$
$$\mathcal{C}^{\text{purity}} = 20, \mathcal{C}^{\text{below}} = 0.5, \mathcal{C}^{mk} = 1, \mathcal{C}^{\text{GM}} = 0.0001.$$

Table 2 provides an overview of the results obtained from solving the benchmark instances with our ALNS. All results have been obtained within a runtime limit of 60 s, and for all instances it has been possible to find solutions that do not break any

Table 2 ALNS results on benchmark instances

Inst.	Obj.	UL	OV	HO	ES	MK	BP	NR	FB	VM
1	40,609.61	1	81	291	97	167	381	389	13,009	1.52E+06
2	46,749.47	1	65	371	125	149	332	357	10,639	1.45E+06
3	54,414.43	0	95	419	65	203	350	448	11,863	1.53E+06
4	29,306.02	1	97	162	128	246	344	396	11,126	1.43E+06
5	72,428.63	0	93	573	25	191	398	536	9838	1.67E+06
6	113,179.96	7	165	838	12	167	434	602	12,073	1.79E+06
7	81,919.94	0	207	557	25	258	422	529	11,485	1.69E+06
8	149,884.30	40	194	841	3	289	438	592	11,537	1.74E+06
9	273,579.36	156	139	966	12	162	416	605	8977	1.81E+06
10	17,011.68	1	11	144	190	170	314	479	13,212	1.73E+06
11	20,369.48	0	92	115	215	210	354	407	15,347	1.68E+06
12	40,246.87	0	63	318	112	198	384	587	15,497	2.02E+06
13	138,880.51	1	60	1265	55	348	477	619	14,584	2.40E+06
14	181,337.19	1	295	1447	8	268	510	739	16,389	2.49E+06
15	75,904.72	1	80	641	89	416	431	532	16,475	2.36E+06
16	497,520.59	280	93	2014	22	267	540	754	16,153	2.80E+06
17	488,933.06	300	52	1772	27	300	493	702	14,287	2.77E+06
18	667,474.49	438	444	1767	0	259	569	738	14,467	2.79E+06
19	39,330.37	0	105	283	218	217	517	680	23,094	3.00E+06
20	435,385.76	386	90	416	316	255	459	718	22,702	2.72E+06
21	246,178.57	140	207	816	263	370	481	892	16,524	3.21E+06
22	126,875.23	0	304	912	59	329	586	872	21,807	3.60E+06
23	112,520.97	1	150	934	91	269	571	800	24,061	3.72E+06
24	213,497.54	130	164	653	182	71	492	821	21,450	3.27E+06
25	701,912.65	476	13	2224	159	671	585	879	26,784	4.29E+06
26	336,175.82	80	68	2437	74	437	665	857	24,070	4.29E+06
27	215,336.12	1	101	1992	75	422	657	810	24,494	4.21E+06

Inst. the instance number, *Obj.* the objective value, *UL* non-loaded containers, *OV* overstowage, *HO* hatch overstowage, *ES* empty stack, *MK* makespan, *BP* Block stowage, *NR* non-reefers, *FB* favour below, *VM* vertical moment

of the stability constraints. The table not only shows the achieved objective, but also presents the values of each of the objective components. As expected, the number of containers that could not be loaded increases as the size of the loadlist grows. The stability and stress forces constraints play an important role in deciding the number of containers that can be loaded onto the vessel. Note, however, that stowage planners might be able to increase the number of loaded containers by making use of ballast tanks. Noteworthy are also the overstowage and hatch overstowage components. Real-life hatch-overstowage would tend to be completely avoided by stowage coordinators, who will use their experience to foresee future cargo loadings. Since the loadlist generation is based on the myopic view of our heuristic, the generated release containers often force hatch- and stack-overstowage.

Since it was not possible to calculate bounds or solutions from the mathematical model, we attempted to evaluate the quality of our solution method by comparing its behaviour when it is given extra time. The results are summarised in Table 3, where it is possible to see that increasing the runtime of the algorithm does indeed improve the solution. The algorithm seems to stop improving approximately after 300 s, as it can be noticed by the fact that results after 600 s are not always better. This does, however, show that the heuristic has room for improvement.

6.1 Analysis of proposed KPIs

As previously mentioned, most of the KPIs that compose the optimisation objective of the problem have already been proposed. In the following, we present an analysis of the ones we introduced in this research.

6.1.1 KPI 6: block purity

In earlier work (Delgado et al. 2012; Parreño et al. 2016), *stack purity* was introduced as a means to cluster containers with the same discharge port. This KPI, however, results in bays becoming a cluster of different discharge ports, as can be seen in Fig. 5a. Such a stowage plan is inefficient for terminal operations. Exchanging stack purity for block purity improves the quality of the generated plan by incentivising containers below/above a hatch cover to have the same discharge port (Fig. 5b). Such a stowage also increases the possibility for terminal operators to perform dual cycling operations.

Table 3 Runtime convergence analysis

Inst	60 s	300 s	600 s	Inst	60 s	300 s	600 s
1	40,609.61	38,648.62	**34,023.43**	15	75,904.72	51,537.77	**49,922.02**
2	46,749.47	42,944.91	**38,567.98**	16	497,520.59	518,695.71	**449,386.72**
3	54,414.43	**50,131.68**	55,737.28	17	488,933.06	457,460.28	**432,452.12**
4	29,306.02	25,165.98	**21,582.01**	18	667,474.49	629,568.29	**597,291.79**
5	72,428.63	60,970.98	**59,382.27**	19	39,330.37	29,060.95	**25,287.17**
6	113,179.96	96,705.41	**94,041.15**	20	435,385.76	**393,106.57**	400,454.29
7	81,919.94	74,669.77	**68,652.28**	21	246,178.57	**242,725.97**	242,912.76
8	149,884.30	100,042.69	**89,938.78**	22	126,875.23	506,743.42	**114,088.79**
9	273,579.36	252,538.68	**250,844.45**	23	112,520.97	95,920.38	**89,947.49**
10	17,011.68	11,258.20	**11,227.37**	24	213,497.54	206,549.43	**205,659.27**
11	20,369.48	20,268.23	**19,104.53**	25	701,912.65	676,800.74	**650,709.23**
12	40,246.87	37,690.87	**33,165.64**	26	336,175.82	328,059.15	**320,470.43**
13	138,880.51	**118,357.88**	129,033.49	27	215,336.12	199,564.33	**189,125.10**
14	181,337.19	154,071.97	**144,137.77**				

Inst. The instance number. The next three columns indicate the objective value achieved after 60, 300 and 600 s, respectively. The best objective is indicated in bold

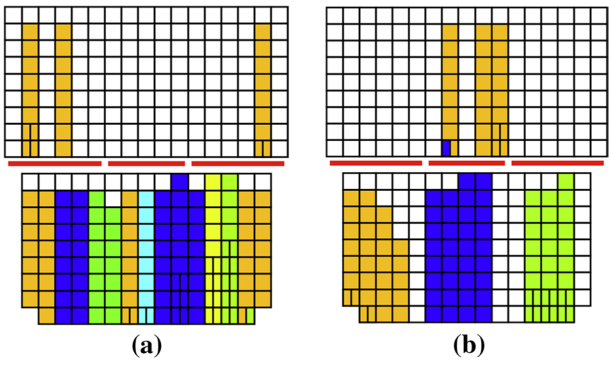

Fig. 5 Comparison of stack and block purity KPIs (colours correspond to discharge ports)

6.1.2 KPI 9: favour below

When generating stowage plans for ports which are at the beginning of a service, ample room is available for stowing containers onboard. If the heuristic is not guided correctly, it might produce plans which will force hatch-overstowage in future ports. Take for example Fig. 6a. The depicted bay is part of a stowage plan generated without favouring containers below deck (KPI 9). Notice how the containers are assigned above a hatch cover, and how the hold below is empty. At the next port, it will be very costly to stow containers below that hatch cover. Figure 6b shows the same bay when the KPI is included in the optimisation. Containers are now stowed below the hatch cover. Those which are above are containers destined for the next port, thus they will need to be discharged anyway. One must, however, be careful when assigning costs to this KPI. Figure 6c shows the same bay with increased cost, and as can be seen, forcing containers below can have a negative impact on the discharge port clustering, and thus on terminal operations.

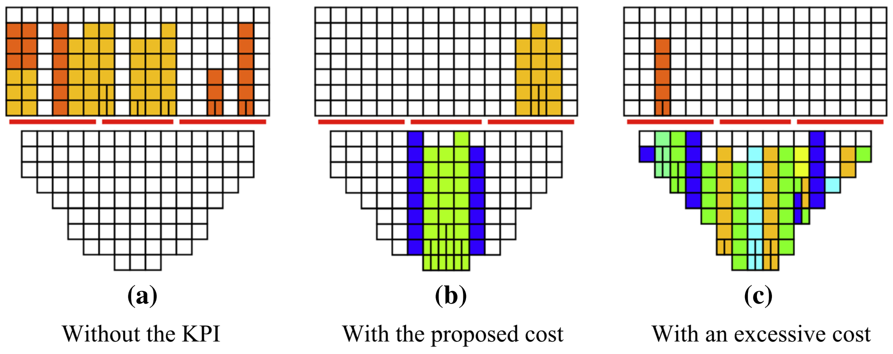

Fig. 6 Analysis of favouring containers below (colours correspond to discharge ports)

7 Conclusions

A realistic set of benchmark instances is presented to the stowage planning research community. The instance set features 3 vessel profiles and 27 novel instances. A formal description of the single-port stowage planning problem is proposed, solved using an adaptation of the ALNS heuristic framework. The solution method produces what qualitatively can be described as high-quality stowage plans within a runtime of 60 s. The computational analysis shows that the heuristic has room for improvement and provides qualitative evidence of how two new KPIs can impact the generated solutions.

From an industrial point of view, the computational results presented in this paper are promising, but the road to automated stowage planning is still long. We have demonstrated that an ALNS-based approach can be used to find good solutions. However, more research is needed to reach industrial application. Further research is needed to identify solution methods that can handle e.g. lashing forces and torsion moments. Recent literature has also included the use of ballast water as a way to achieve vessel stability. Though this is an interesting approach, very little is known on how to balance the cost of ballast water (extra weight and thus increased bunker consumption) with the cost of overstowage or of slower handling operations.

From an academic point of view, there is a need to identify a common definition of the problem in order to be able to compare solution approaches and devise proper theories. It is our hope that this article and the proposed benchmarks can contribute towards achieving this goal. A common dataset will also allow quantifiable evaluation of the difference between e.g. KPI-based approaches and multi-objective approaches.

An interesting future path of research is the evaluation of the new and established KPIs in a multi-port setting; can the current KPIs support the generation of robust stowage plans in future ports? Or is cargo forecasting an absolute necessity? In future work we plan to present a simulation study to investigate the robustness of the KPIs present in the literature and analyse the need to adjust them or to create entirely new ones.

Appendix 1: mathematical notation

Sets	
A	The set of adjacent bays
B	The set of bays
C	The set of cells
C_s	The set of cells belonging to stack $s \in S$
C_i^{above}	The set of cells above hatch cover $i \in H$
C_i^{below}	The set of cells below hatch cover $i \in H$
H	The set of hatch covers
L	The set of slots (Fore, Aft) in a cell

O	The set of linearisation points for the displacement data
P	The set of discharge ports
R	The release containers composed of tuples (c, l, t), where $c \in C$ is the cells, and $l \in L$ is the slot (Fore, Aft), where a container of class $t \in T$ is stowed
R_i^{above}	The set of release containers above hatch cover $i \in H$
S	The set of stacks
S_b	The set of stacks belonging to bay $b \in B$
S_i^q	The set of stacks above/below (q) hatch cover $i \in H$
T	The set of container classes
T^{20}, T^{40}	The set of 20- and 40-foot container classes
TR	The set of reefer container classes
Parameters	
$\mathcal{C}^{\text{below}}$	The cost of stowing containers below deck
$\mathcal{C}^{\text{empty}}$	The cost of using a stack
$\mathcal{C}^{\text{GM}}$	The cost imposed on the vertical moment
$\mathcal{C}^{\text{hatch}}$	The cost of hatch overstowage
$\mathcal{C}^{\text{load}}$	The cost of non-loading a container
$\mathcal{C}^{\text{Mk}}$	The cost of the makespan objective
$\mathcal{C}^{\text{purity}}$	The cost associated to the block purity KPI
$\mathcal{C}^{\text{ov}}$	The cost of overstowage
$\mathcal{C}^{\text{reefer}}$	The cost of assigning a non-reefer container in a reefer slot
$\text{bend}_b^{\max} \in \mathbb{R}^+$	The maximum bending force allowed for bay $b \in B$
$\beta_c \in \mathbb{B}$	Indicates whether cell $c \in C$ is below deck
$\text{bot}(s) \in C$	The cell at the bottom of stack $s \in S$
$\text{buoy}_{bo} \in \mathbb{R}$	The buoyancy in bay $b \in B$ given the hydrostatic data point $o \in O$
$d_t \in \mathbb{N}$	The discharge port of container class $t \in T$
$\text{disp}_o \in \mathbb{R}^+$	The displacement at point $o \in O$
$h_s^{\max} \in \mathbb{R}^+$	The maximum height of stack $s \in S$
$h_t \in \mathbb{R}^+$	The height of container class $t \in T$
$\text{lcg}_b \in \mathbb{R}^+$	The initial longitudinal centre of gravity of bay $b \in B$
$\text{lcg}_o^{\max} \in \mathbb{R}$	The maximum longitudinal centre of gravity for displacement point $o \in O$
$\text{lcg}_o^{\min} \in \mathbb{R}$	The minimum longitudinal centre of gravity for displacement point $o \in O$
$n_{pt} \in \mathbb{Z}^+$	The number of containers of class $t \in T$ that have discharge port $p \in P$
$r_c \in \mathbb{N}$	The number of power plugs available in cell $c \in C$
$r_{cl} \in \mathbb{B}$	Indicates with a 1 if cell $c \in C$ in slot $l \in L$ holds a power plug, and 0 otherwise
$\text{shear}_b^{\max} \in \mathbb{R}$	The maximum shear force allowed in bay $b \in B$
$\text{shear}_b^{\min} \in \mathbb{R}$	The minimum shear force allowed in bay $b \in B$
$\text{tcg}_s \in \mathbb{R}$	The initial transversal centre of gravity of stack $s \in S$
$\text{tcg}^{\max} \in \mathbb{R}$	The maximum transversal centre of gravity of the vessel
$\text{tcg}^{\min} \in \mathbb{R}$	The minimum transversal centre of gravity of the vessel
$vcg_b \in \mathbb{R}$	The vertical centre of gravity of the constant weight of bay $b \in B$
$vcg_s \in \mathbb{R}$	The initial vertical centre of gravity of stack $s \in S$
$w_b^{\text{const}} \in \mathbb{R}^+$	The constant weight in bay $b \in B$
$w_s^{\max 20} \in \mathbb{R}^+$	The maximum 20-foot weight of stack $s \in S$

$w_s^{\max 40} \in \mathbb{R}^+$	The maximum 40-foot weight of stack $s \in S$
$w_t \in \mathbb{R}^+$	The weight of container class $t \in T$
Decision variables	
$bp_{ip}^q \in \mathbb{B}$	Indicator variable that is 1 iff the block above/below ($q \in \{\text{above, below}\}$) hatch cover $i \in H$ stows containers destined to port $p \in P$
$by_b \in \mathbb{R}^+$	The buoyancy for bay $b \in B$
$\delta_{csp} \in \mathbb{B}$	Indicator variable that is 1 iff a container with discharge port less than $p \in P$ is stowed in any cell of stack $s \in S$ below cell $c \in \mathcal{C}_s$
$e_s \in \mathbb{B}$	Indicator variable that is 1 iff stack $s \in S$ is not empty
$\lambda_o \in \mathbb{B}$	Indicator variable that is 1 iff the displacement point $o \in O$ is active
$h_{ci} \in \mathbb{B}$	Indicator variable that is 1 iff cell $c \in C$ of hatch cover $i \in H$ holds a hatch-overstowing container
$ho_{ci} \in \mathbb{B}$	Indicator variable that is 1 iff cell $c \in C_i^{\text{above}}$ *is* above hatch cover $i \in H$
$lm \in \mathbb{R}$	The longitudinal moment of the vessel
$mk \in \mathbb{R}^+$	The makespan of the stowage plan
$nr_{cl} \in \mathbb{B}$	Indicator variable that is 1 iff cell $c \in C$ in slot $l \in L$ holds a non-reefer container
$o_{cl} \in \mathbb{B}$	Indicator variable that is 1 iff cells c in slot $l \in L$ stows an overstowing container
$tm \in \mathbb{R}$	The transversal moment of the vessel
$vm \in \mathbb{R}^+$	The vertical moment of the vessel
$w_b^{\text{bay}} \in \mathbb{R}^+$	The total weight of bay $b \in B$
$w^{\text{disp}} \in \mathbb{R}^+$	The total displacement of the vessel
$x_{clt} \in \mathbb{B}$	The main decision variable with value 1 iff a container of class $t \in T$ is stowed in cell $c \in C$ at slot $l \in L$
$y_t \in \mathbb{N}$	The number of containers of type $t \in T$ not loaded on the vessel

Appendix 2: stability values of the solution

Inst	VCG	TCG	LCG	Displacement
1	23.74	−0.03	−0.10	86,853
2	24.29	−0.02	−0.35	81,976
3	24.25	0.10	−0.20	85,368
4	23.27	0.06	−0.18	84,701
5	24.43	0.06	−0.50	90,352
6	24.56	0.10	−0.30	95,110
7	25.04	−0.09	−0.12	89,281
8	24.95	0.09	−0.46	91,352
9	25.33	−0.06	−0.40	92,749
10	24.73	0.09	−0.23	95,330
11	23.69	0.00	−0.41	97,569
12	25.03	−0.05	−0.30	106,073
13	25.35	−0.01	−0.27	119,359
14	26.66	−0.07	−0.41	116,936

Inst	VCG	TCG	LCG	Displacement
15	25.07	0.02	−0.27	119,410
16	26.14	−0.03	−0.15	131,316
17	26.01	0.10	−0.13	130,592
18	26.51	−0.02	−0.33	129,072
19	28.45	−0.05	0.27	144,018
20	28.86	0.09	0.42	132,076
21	28.92	0.04	0.38	148,691
22	29.95	−0.07	0.05	156,645
23	29.26	−0.06	0.32	164,713
24	29.32	0.00	0.38	148,672
25	29.38	0.00	0.30	183,135
26	29.41	0.10	0.30	183,017
27	29.20	0.05	0.26	181,691

Inst. The instance number, *VCG* the vertical centre of gravity of the vessel, *TCG* the transversal centre of gravity of the vessel, *LCG* the longitudinal centre of gravity of the vessel, *Displacement* the displacement weight of the vessel

References

Ambrosino, D., Paolucci, M. and Sciomachen, A. 2018. Shipping Liner Company Stowage Plans: An Optimization Approach, in Żak, J., Hadas, Y., and Rossi, R. (eds) *Advances in Intelligent Systems and Computing*. Cham: Springer International Publishing (Advances in Intelligent Systems and Computing): 405–420.

Ambrosino, D., and A. Sciomachen. 2018. *A shipping line stowage-planning procedure in the presence of hazardous containers*, 1–22. Maritime Economics & Logistics: Palgrave Macmillan.

Ambrosino, D., A. Sciomachen, and E. Tanfani. 2004. Stowing a containership: The master bay plan problem. *Transportation Research Part A: Policy and Practice* 38 (2): 81–99.

Ambrosino, D., A. Sciomachen, and E. Tanfani. 2006. A decomposition heuristics for the container ship stowage problem. *Journal of Heuristics* 12 (3): 211–233.

Avriel, M., and M. Penn. 1993. Exact and approximate solutions of the container ship stowage problem. *Computers and Industrial Engineering* 25 (1–4): 271–274.

Avriel, M., M. Penn, and N. Shpirer. 2000. Container ship stowage problem: Complexity and connection to the coloring of circle graphs. *Discrete Applied Mathematics* 103 (1–3): 271–279.

Avriel, M., M. Penn, N. Shpirer, and S. Witteboon. 1998. Stowage planning for container ships to reduce the number of shifts. *Annals of Operations Research* 76 (1–4): 55–71.

Botter, R. and Brinati, M. 1991. Stowage Container planning: a model for getting an optimal solution. *IFIP TC5/WG 5.6 7th Intl Conf on Computer Applications in the Automation of Shipyard Operation and Ship Design*. 10–13.

Chou, C.C., and P.Y. Fang. 2018. *Applying expert knowledge to containership stowage planning: An empirical study*, 1–24. Maritime Economics & Logistics: Palgrave Macmillan.

Chao, S.-L., and P.-H. Lin. 2019. *Minimizing overstowage in master bay plans of large container ships*, 1–23. Maritime Economics & Logistics: Palgrave Macmillan.

Christensen, J., and D. Pacino. 2017. A matheuristic for the Cargo Mix Problem with Block Stowage. *Transportation Research Part E: Logistics and Transportation Review.* 97: 151–171.

Cruz-Reyes, L., H.P. Hernández, P. Melin, H.H.J. Fraire, and O.J. Mar. 2013. Constructive algorithm for a benchmark in ship stowage planning. In *Recent advances on hybrid intelligent systems*, 393–408. Berlin: Springer.

Davidor Y. 1998. Method for determining a stowage plan. US Patent US5809489A, filed the 21/06/1996, published the 15/09/1998.

Delgado, A. 2013. Models and Algorithms for Container Vessel Stowage Optimization. Ph.D. Thesis, IT-University of Copenhagen, Denmark.

Delgado, A., R. Jensen, and K. Janstrup. 2012. A constraint programming model for fast optimal stowage of container vessel bays. *European Journal of Operational Research* 220 (1): 251–261.

Ding, D., and M.C. Chou. 2015. Stowage planning for container ships: A heuristic algorithm to reduce the number of shifts. *European Journal of Operational Research* 246 (1): 242–249.

Doe, R. 2018. Stowman [s] - stowage planning software, https://www.interschalt.com/software/stowage-planning.html. Accessed 17 June 2018

Dubrovsky, O., G. Levitin, and M. Penn. 2002. A genetic algorithm with a compact solution encoding for the container ship stowage problem. *Journal of Heuristics* 8 (6): 585–599.

Guilbert, N., Paquin, B. 2008. Container vessel stowage planning, EP Patent EP1909221A1, filed the 06/10/2006, published the 09/04/2008.

Helo, P., Paukku, H., & Sairanen, T. 2018. Containership cargo profiles, cargo systems, and stowage capacity: Key performance indicators. *Maritime Economics & Logistics*: 1–21.

Imai, A., K. Sasaki, E. Nishimura, and S. Papadimitriou. 2006. Multi-objective simultaneous stowage and load planning for a container ship with container rehandle in yard stacks. *European Journal of Operational Research.* 171 (2): 373–389.

Iris, Ç., D. Pacino, and S. Ropke. 2017. Improved formulations and an Adaptive Large Neighborhood Search heuristic for the integrated berth allocation and quay crane assignment problem. *Transportation Research Part E: Logistics and Transportation Review* 105: 123–147.

Iris, Ç., J. Christensen, D. Pacino, and S. Ropke. 2018. Flexible ship loading problem with transfer vehicle assignment and scheduling. *Transportation Research Part B: Methodological.* 111: 113–134.

Kang, J.-G., and Y.-D. Kim. 2002. Stowage planning in maritime container transportation. *Journal of the Operational Research Society* 53 (4): 415–426.

Li, J., Zhang, Y. and Ji, S. 2017. A two-phase approach for inland container ship stowage on full route on Yangtze River, in *2017 4th International Conference on Transportation Information and Safety, ICTIS 2017 - Proceedings*: 799–805.

Li, K. 2012. Modelling and Tabu search heuristic for solving container stowage planning problem, in *Proceedings of the 2012 24th Chinese Control and Decision Conference, CCDC 2012*: 2676–2680.

Monaco, M.F., M. Sammarra, and G. Sorrentino. 2014. The terminal-oriented ship stowage planning problem. *European Journal of Operational Research.* 239 (1): 256–265.

Muller, L.F., S. Spoorendonk, and D. Pisinger. 2012. A hybrid adaptive large neighborhood search heuristic for lot-sizing with setup times. *European Journal of Operational Research* 218 (3): 614–623.

Nugroho S. 2006. Method and system aiding with the establishment of stowage plans for container vessels, corresponding computer program, and corresponding computer-readable storage medium, WO Patent WO2006000462A1, filed the 27/06/2005, published the 05/01/2006.

Pacino, D. 2012. *Fast generation of container vessel stowage plans*. Ph.D. Thesis, IT-University of Copenhagen, Denmark.

Pacino, D. 2018. Crane intensity and block stowage strategies in stowage planning. In: *International Conference on Computational Logistics. Springer, Cham,* 2018: 191–206.

Pacino, D., A. Delgado, M.R. Jensen, and T. Bebbington. 2011. Fast generation of near-optimal plans for eco-efficient stowage of large container vessels. *Computational Logistics*, Part of the Lecture Notes in Computer Science book series (LNCS), 6971: 286–301.

Pacino, D., A. Delgado, M.R. Jensen, and T. Bebbington. 2012. An accurate model for seaworthy container vessel stowage planning with ballast tanks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7555 LNCS: 17–32.

Parreño, F., D. Pacino, and R. Alvarez-Valdes. 2016. A GRASP algorithm for the container stowage slot planning problem. *Transportation Research Part E: Logistics and Transportation Review* 94: 141–157.

Roberti, R., and D. Pacino. 2018. A decomposition method for finding optimal container stowage plans. *Transportation Science.* 52 (6): 1444–1462.

Ropke, S., and D. Pisinger. 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science* 40 (4): 455–472.

Tierney, K., D. Pacino, and R. Jensen. 2014. On the complexity of container stowage planning problems. *Discrete Applied Mathematics* 169: 225–230.

UNCTAD. 2017. *Review of Maritime Transport 2016*.

Wilson, I., and P. Roach. 1999. Principles of combinatorial optimization applied to container-ship stowage planning. *Journal of Heuristics* 5 (4): 403–418.