Research Article

Solution Strategies for a Multiport Container Ship Stowage Problem

Consuelo Parreño-Torres ¹, Ramon Alvarez-Valdes ¹, and Francisco Parreño²

¹University of Valencia, Department of Statistics and Operations Research, Burjassot, Valencia, Spain

²University of Castilla-La Mancha, Department of Mathematics, Albacete, Spain

Correspondence should be addressed to Consuelo Parreño-Torres; consuelo.parreno@uv.es

Received 21 November 2018; Accepted 15 May 2019; Published 23 May 2019

Academic Editor: Alessandro Formisano

Copyright © 2019 Consuelo Parreño-Torres et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The multiport container ship stowage problem consists in determining the position of the containers on board a ship along its route with the objective of minimizing the number of unproductive moves required in the loading and unloading operations at each port. This paper presents an integer programming model for the problem and proposes several sets of valid constraints that bring its LP-relaxation closer to an integer solution. Moreover, it presents a GRASP algorithm that generates stowage plans with a minimal number of unproductive moves in a high percentage of medium and large-size instances. An extended computational analysis has been performed in which, to the best of the authors' knowledge, the efficiency of integer programming models for the problem is tested for the first time. With respect to GRASP, the computational results show that it performs well on different sized datasets.

1. Introduction

Seaborne trade accounts for nearly 80% of world trade and is one of the most efficient modes of transportation [1]. About 90% of nonbulk cargo worldwide is transported by container [2]. Since its development, containerization has played an everincreasing role, due to its important advantages, reducing packaging and deterioration of goods and increasing productivity in the intermodal transportation of products. Global containerized trade expanded at a rate of 3.1% in 2016, with volumes attaining an estimated 140 million Twenty-foot Equivalent Units (TEUs) [1].

The high volumes reached have produced an increase in ship size, which still continues. Modern container vessels can handle up to 20,000 TEUs. Currently, the world's largest container ship, the *OOCL Hong Kong*, has capacity for 21413 TEUs [5]. For these large vessels to achieve economies of scale, they must be loaded at near full capacity. These vessels sail from port to port loading and unloading thousands of containers. Minimizing the time a vessel spends in port requires an efficient stowage plan, a plan that describes in which position each container should be loaded into the

vessel. This plan has to satisfy high-level constraints, related to the stability of the ship, as well as low-level constraints related to the way in which containers have to be stacked, the weight distribution, and the specific conditions regulating containers with dangerous products. The main objective is to load all the assigned containers on board if they feasibly fit into the ship. If this is not possible, then the objective is to load the maximum number of containers. A second very important objective is to minimize the number of unproductive moves in the loading/unloading operations. These moves, also called reshuffles, arise at a port when some containers going to later ports have to be unloaded to gain access to containers that must be unloaded at that port and are situated below them. Once all the containers going to this port have been unloaded, the reshuffled containers are loaded again into the ship, together with all the containers that had to be loaded at this port.

In this paper we study the multiport container ship stowage problem, considering the whole route of the ship and the different sets of containers which must be loaded/unloaded at each port on the route. The problem consists in determining the position of the containers loaded

at each port so as to minimize the number of unproductive moves necessary to unload the containers at their port of discharge.

The contributions of our study are basically of two types. Firstly, we have developed a new integer linear model, as an alternative to the existing models by Avriel et al. [3] and by Ding and Chou [4], requiring fewer variables and obtaining better results concerning the number of optimal solutions obtained and the quality of feasible solutions when the time limit is reached without proving optimality. Nevertheless, as the computational study shows the limits of mathematical models for solving large-size instances, in the second part of the study, we have developed a metaheuristic GRASP algorithm, going one step further than the constructive algorithms proposed so far, and including randomization strategies and a local search tailored to the problem. An extensive computational study shows that GRASP obtains much better results than previous procedures, generating stowage plans with a minimal number of reshuffles in a high percentage of medium and large-size instances.

The structure of the paper is as follows. The relevant literature on stowage problems is reviewed in Section 2. Section 3 presents a detailed definition of the problem. In Section 4 we present a new mathematical formulation and propose several families of valid inequalities. The GRASP metaheuristic is presented in Section 5 by describing its randomized constructive algorithm and the local search. The computational experience carried out is summarized in Section 6. Finally, conclusions and future work are highlighted and discussed in Section 7.

2. Previous Work

Academic work on stowage planning problems has increased notably in recent years. The studies can be classified into two main groups: single-port and multiport problems.

The single-port problem consists in determining the arrangement of containers in the ship at a given port without considering the loading of containers in subsequent ports. The vast majority of papers in the literature related to stowage planning belong to this group and it is usual to refer to this problem as the Master Bay Planning Problem (MBPP). For a detailed description of the MBPP and its constraints, see, for instance, Ambrosino et al. [6]. Even though this problem is difficult, it is not as hard as the multiport problem, so there is a trend of progressively including more realistic constraints. Papers related to the MBPP can be divided into two approaches: those that consider the problem as a whole and those that decompose it into several optimization subproblems. In the first approach, exact methods such as mathematical models [6] or branch and bound algorithms [7] can be found, as well as heuristic algorithms, such as Sciomachen and Tanfani [8], based on an exact method for the 3D-BPP. In the second approach, the problem is divided into two phases. First, the containers are distributed in clusters along the vessel. Then, for each cluster a specific position for each container must be found. Examples of this

approach are Ambrosino et al. [9], Pacino et al. [10], Delgado et al. [11], and Parreño et al. [12].

In the multiport problem, the stowage problem is considered as a dynamic loading and unloading problem. According to the classification by Lehnfeld and Knust [13], it is a combined problem, since it is possible to perform two types of moves. Two main approaches have been followed in recent years. On the one hand, some authors use a multiphase approach, proposing a decomposition of the problem into several optimization models that can be solved individually. This decomposition makes it possible to include realistic characteristics of the problem, in the objective function, considering costs related to nonloaded containers, reshuffles, and quay cranes, and also to include constraints concerning the stability of the ship [14–18]. On the other hand, there are several studies that follow a single-phase approach. The problem is solved as a whole, looking for loading strategies that minimize the number of reshuffles at each port on the route, thus providing a complementary view of the stowage problem, usually considering routes with many ports and ships with large numbers of containers, but disregarding stability conditions and considering just one type of container. Avriel et al. [3] presented a mathematical model and developed a constructive heuristic, the *Suspensory Heuristic* (SH), to stow containers of the same size in a ship consisting of a single rectangular bay so as to minimize the total number of unproductive moves in the whole route. The same problem was studied by Dubrovsky et al. [19], who presented a genetic algorithm that was able to incorporate constraints beyond the simplified problem and performed nearly as well as the SH. More recently, Ding and Chou [4] have proposed a new heuristic algorithm that is shown to perform better than the SH in an extensive computational study, and also, in an online supplement, an integer linear model. Our study belongs to this class of single-phase approaches to multiport stowage problems, and in addition to a new integer linear model, we develop a more efficient GRASP metaheuristic algorithm.

3. Problem Description

Let us consider a ship traveling along a trade route consisting of N ports. All containers are the same size and no stability conditions are considered, so the ship can be seen as having a single bay with R rows and C columns. The position of each container is described by the pair (r, c) where $r \in \{1, \dots, R\}$, $r = 1$ is the bottom row, and $c \in \{1, \dots, C\}$, $c = 1$ is the first column on the left. We consider sets $\mathcal{N} = \{1, \dots, N\}$, $\mathcal{R} = \{1, \dots, R\}$, and $\mathcal{C} = \{1, \dots, C\}$ to simplify notation. Note that here we are following the notation proposed by Avriel et al. [3] and Ding and Chou [4], although in other stowage problems horizontal levels are referred as tiers and vertical levels as rows or stacks.

The loading/unloading operations along the route are defined by an $N \times N$ transportation matrix T whose input (i, j) is the number of containers originating at port i with destination port j . It is a nonnegative upper triangular matrix, so $T_{ij} = 0$ for all $i \geq j$. We assume, as has been done in previous studies [3, 4], that all transportation matrices are

feasible, so all the containers to be shipped can be stowed at every port according to (1).

$$\sum_{k=1}^i \sum_{j=i+1}^N T_{kj} \leq R \cdot C \quad \forall i \in \mathcal{N} : i < N; \quad (1)$$

The vessel starts its route at port 1, and sequentially visits ports 2, 3, ..., N . At each port $i = 1, \dots, N - 1$, containers with destination i are unloaded and containers bound for destinations $j = i + 1, \dots, N$ are loaded. A container whose destination is port j is called a j -container. At the last port N , the vessel is emptied. Therefore, it can be seen as a circular route in which port 1 is the same as port N . If there is a j -container in position (r, c) and there is a k -container, $k < j$, in some position (r', c) with $r' < r$, the first one is said to be a

$$x_{ijrc} = \begin{cases} 1, & \text{if a } j\text{-container is stowed in slot } (r, c) \text{ after loading operations at port } i \\ 0, & \text{otherwise} \end{cases}$$

$$y_{ijrc} = \begin{cases} 1, & \text{if a } j\text{-container stowed in slot } (r, c) \text{ at port } i - 1 \text{ is unproductively moved at port } i \\ 0, & \text{otherwise} \end{cases}$$

According to the problem description, the ship is emptied after the unloading operations at port N . Therefore from port $N - 1$ to port N there are only N -containers in the ship, and there will be no reshuffles at port N . Moreover, since we assume that the number of containers to be loaded on board is not greater than the number of available locations, the exact position of each container when the ship leaves port $N - 1$ is irrelevant. Under these assumptions, neither variables $x_{(N-1)Nrc}$ nor variables y_{Nrc} need to be considered.

The multiport container ship stowage problem can be formulated as follows:

$$\min \sum_{i=2}^{N-1} \sum_{j=i+1}^N \sum_{r=1}^R \sum_{c=1}^C y_{ijrc} \quad (3)$$

$$x_{i-1jrc} - x_{ijrc} \leq y_{ijrc} \quad \forall i, j \in \mathcal{N} : 1 < i \leq N - 2, i < j; \forall r \in \mathcal{R}; \forall c \in \mathcal{C}; \quad (4)$$

$$x_{i-1ir-1c} + \sum_{j=i+1}^N y_{ijr-1c} \leq x_{i-1irc} + \sum_{j=i+1}^N y_{ijrc} \quad (5)$$

$$\forall i \in \mathcal{N} : 1 < i < N; \forall r \in \mathcal{R} : r > 1; \forall c \in \mathcal{C};$$

$$\sum_{r=1}^R \sum_{c=1}^C x_{1jrc} = T_{1j} \quad \forall j \in \mathcal{N} : j > 1; \quad (6)$$

blocking container and will produce a reshuffle in some port l , $l \leq k$. The objective of the problem is to minimize the number of reshuffles along the route. Note that a non-full loading transportation matrix T can always be transformed into a full-loading matrix T' , adding as many dummy containers from i to $(i + 1)$ as necessary. We can consider, without loss of generality, the inequality of (1) as an equality.

4. Mathematical Formulation

In this section we describe a new integer linear model and several classes of valid inequalities to enhance it.

4.1. An Integer Linear Model. The model involves two types of binary variables describing the configuration of the ship and the containers that are moved unproductively at each port.

$$\forall i, j \in \mathcal{N} \text{ s.t. } i \leq N - 2 \text{ and } i < j; \forall r \in \mathcal{R}; \forall c \in \mathcal{C}; \quad (2)$$

$$\forall i, j \in \mathcal{N} \text{ s.t. } 1 < i < N \text{ and } i < j; \forall r \in \mathcal{R}; \forall c \in \mathcal{C};$$

$$\sum_{r=1}^R \sum_{c=1}^C (x_{ijrc} - x_{i-1jrc}) = T_{ij} \quad (7)$$

$$\forall i, j \in \mathcal{N} : 1 < i \leq N - 2, i < j;$$

$$\sum_{j=i+1}^N x_{ijrc} \leq 1 \quad \forall i \in \mathcal{N} : i \leq N - 2; \forall r \in \mathcal{R}; \forall c \in \mathcal{C}; \quad (8)$$

The objective function (3) seeks to minimize the total number of reshuffles. Reshuffles are identified by constraints (4) and (5). Constraints (4) identify changes of destination in a given position. Constraints (5) identify unproductive moves required to unload a container, serving two purposes. On the one hand, to take into account the obvious case in which $x_{i-1ir-1c} = 1$, meaning that there is an i -container in slot $(r - 1, c)$ at port $i - 1$, the container above it in slot (r, c) must also be unloaded at port i (making $x_{i-1irc} = 1$) or there is a reshuffle and one of the variables y_{ijrc} must be 1 for some $j > i$. On the other hand, if there has been a reshuffle in the stack at any tier $r - 1$ (and then the variable y_{ijr-1c} is equal to 1 for some $j > i$) the container above it in slot (r, c) must be unloaded at port i or it will produce a new reshuffle (and some variable y_{ijrc} must be 1 for some $j > i$). In this second case, all blocking containers in the stack will be counted as reshuffles. Finally, constraints (6) and (7) fix the number of containers to be loaded at each port and constraints (8) force each location to have at most one container at each port. At the first port,

5	3	3	3	4	5	6	4	4	4
4	6	3	4	4	4	6	5	4	4
3	5	5	3	5	3	6	5	4	5
2	6	5	3	5	2	6	5	4	5
1	6	6	3	6	1	6	6	4	6
	1	2	3	4		1	2	3	4

Port 2 Port 3

FIGURE 1: Example of configurations on ports 2 and 3 to detect.

the ship is empty before loading operations, and after these operations there are T_{1j} j -containers on board (constraints (6)). At other ports, the number of j -containers loaded at each port i , T_{ij} , is equal to the number of j -containers on board after loading operations at this port i minus the number of j -containers on board before loading operations at that port, that is, minus the total number of containers whose destination is port j and were loaded at a port before i (constraints (7)).

We refer to the example in Figure 1 to better understand the model and detect the different types of reshuffles that can appear. It shows the configurations of a solution on ports 2 and 3. Each square represents a container and the number indicates the destination port. The gray-marked containers are those that are relocated on port 3. As there is a change of destination in position (3, 1) between ports 2 and 3, there has been a reshuffle at port 3 of that container and $y_{3531} = 1$ by constraints (4). We now focus on the 6-container located in slot (4, 1) at port 2. It must have been unloaded at port 3 to unload the 5-container but there is a container with the same destination in slot (4, 1) at port 3, so constraints (4) do not consider this case. It is identified by constraints (5). Since there has been a reshuffle in the stack 1 at tier 3 at port 3, the container above it in slot (4, 1) must have destination 3 or it will produce a new reshuffle as is the case. Constraints (5) also identify unproductive moves required to unload a container at its destination port. Consider now the container located in slot (4, 3) at port 2. There is a 3-container at port 2 in slot (3, 3), so the container on top of it has to be unloaded unproductively at port 3 because its destination is not port 3.

4.2. New Valid Inequalities. The integer linear formulation described above produces valid solutions for the multiport stowage problem, but its linear relaxation is quite loose, producing very low lower bounds. In most cases, the lower bound at the root node is precisely zero. In order to enhance the integer formulation, we studied the solutions provided by its linear relaxation and identified some valid constraints satisfied by all integer solutions but not by the linear relaxation solutions. Adding these inequalities to the initial model would produce a tighter linear relaxation, improving the quality of the bounds and therefore reducing the search tree required to obtain an optimal solution.

We consider the upper row R of the bay at port $i - 1$ and the number of containers going to the next port, i . If the number of slots in the upper row occupied by i -containers at port $i - 1$ is less than the minimum number required to ensure direct access to these containers, there will be at least as many reshuffles as necessary at port i to reach that number. It is satisfied by adding constraints (9).

$$\sum_{c=1}^C x_{i-1iRc} + \sum_{j=i+1}^N \sum_{c=1}^C y_{ijRc} \geq \left\lceil \frac{\sum_{p=1}^{i-1} T_{pi}}{R} \right\rceil \quad (9)$$

$$\forall i \in \mathcal{N} : 1 < i < N;$$

Constraints (10) extend (9) to consider not only row R , but all rows from r' to R .

$$\begin{aligned} & \sum_{s=r'}^R \sum_{c=1}^C x_{i-1isc} + \sum_{j=i+1}^N \sum_{s=r'}^R \sum_{c=1}^C y_{ijsc} \\ & \geq (R - r' + 1) \cdot \left\lceil \frac{\sum_{p=1}^{i-1} T_{pi}}{R} \right\rceil \\ & + \min \left\{ \sum_{p=1}^{i-1} T_{pi} \pmod{R}, (R - r' + 1) \right\} \end{aligned} \quad (10)$$

$$\forall i \in \mathcal{N} : 1 < i < N; \quad \forall r' \in \mathcal{R} : r' > 1;$$

We now consider the bottom row. If there are containers on board going to the last port, N , either they are placed on top of other N -containers or in the bottom row. Otherwise, they will block other containers and that will produce some reshuffles at subsequent ports.

$$R \cdot \sum_{c=1}^C x_{iN1c} + \sum_{i'=i+1}^{N-1} \sum_{j=i'+1}^N \sum_{r=2}^R \sum_{c=1}^C y_{i'jrc} \geq \sum_{p=1}^i T_{pN} \quad (11)$$

$$\forall i \in \mathcal{N} : i \leq N - 2;$$

Instead of considering only the containers going to the last port, N , we can consider the containers going to a port k and subsequent ports. The argument is similar to the previous constraints, but we now have constraints (12).

$$R \cdot \sum_{j=k}^N \sum_{c=1}^C x_{ij1c} + \sum_{i'=i+1}^{N-1} \sum_{j=i'+1}^N \sum_{r=2}^R \sum_{c=1}^C y_{i'jrc} \geq \sum_{p=1}^i \sum_{j=k}^N T_{pj} \quad (12)$$

$$\forall i, k \in \mathcal{N} : i \leq N - 2, i < k;$$

If after loading operations at port $i - 1$ there is a container in position (r, c) whose destination is port k or a port subsequent to k on top of a container whose destination is previous to k , the first container will be relocated at some port prior to k . Then, constraints (13) hold.

$$\sum_{j=k}^N x_{i-1jrc} \leq \sum_{j=k}^N x_{i-1jrc} + \sum_{p=i}^{k-1} \sum_{j=p+1}^N y_{pjr+1c} \quad (13)$$

$$\forall i, k \in \mathcal{N} : 1 < i \leq N - 2, i < k; \quad \forall r \in \mathcal{R} : r < R; \quad \forall c \in \mathcal{C}$$

FIGURE 2: Example of configuration at port 2 given by the LP-relaxation at the root node.

When we introduce these constraints, the linear programming relaxation provides a solution which lies in general not too far away from the integer optimum. More importantly, the lower bound is increased.

We illustrate their effect by an example, in which we consider a container ship with 6 rows and 4 columns and whose route is composed of 8 ports. Although its optimal value is 3 reshuffles, the objective value of the LP-relaxation (without valid constraints), provided by CPLEX at the root node of its internal Branch & Bound process, is zero, which is indeed a loose lower bound. Figure 2 shows the configuration provided by the LP-relaxation at port 2. On the left-hand side, the bay is represented by the large square, divided into smaller squares that represent each slot of the bay, and each one of these squares is split up (using dotted lines) into smaller squares in order to assign the value taken by the variable that represents each destination port.

Looking at the two enlarged slots that represent slots (6,2) and (5,2) and considering port $k = 8$ as a reference, the variable whose destination is port 8 takes a positive value in the upper slot. Nevertheless, in the lower slot the variable whose destination is port 8 takes value zero, so constraints (13) are not satisfied. Moreover, according to the transportation matrix, there is one container going to the next port, port 3, so in order to avoid reshuffles, it should be stowed in the top row, row 6. However, looking at the top row, at the top right corner of Figure 2, the sum of the values of variables whose destination is port 3 is less than one. This solution could be removed by including constraints (9).

If the valid constraints are included in the initial model and the LP-relaxation of this extended model is solved, its objective value takes value two, so the lower bound is increased and the solution obtained is very near to an integer solution.

5. A GRASP Algorithm

GRASP (Greedy Randomized Adaptive Search Procedure) is a multistart metaheuristic in which each iteration consists basically of two phases: a randomized construction phase and an improvement phase. The first phase builds a feasible solution, and the improvement phase seeks to improve the outcome. Once the two phases have been executed, the solution is stored and the method starts another iteration, keeping the best solution found so far. An extended description of the algorithm can be found in Resende and Ribeiro [20]. GRASP algorithms have been successfully developed for many hard combinatorial problems, in container loading [21], location [22], and routing [23], among other areas.

Concerning stowage problems closely related to that studied here, Parreño et al. [12] also used GRASP for the slot planning problem, but in recent papers other approaches have been successfully applied: particle swarm optimization [24], genetic algorithms [25–27], simulated annealing [27], or hill climbing [27].

The following subsections present the specific implementation of the metaheuristic GRASP that we have developed for this stowage problem: the randomizing strategies, and the local search procedure.

5.1. Randomizing the Constructive Algorithms. To the best of our knowledge, there are two constructive algorithms in the scientific literature that deal with the problem studied in this paper: that of Avriel et al. [3] and that of Ding and Chou [4]. The suspensory algorithm proposed by Avriel et al. [3] is basically composed of a hanging step and a filling step. In the hanging step the containers are assigned to columns by increasing order of destination, occupying the top positions (*hanging*), and in the filling step the columns are completed

and containers are put in the usual positions one on top of the other. On the other hand, the algorithm by Ding and Chou [4] is composed of rules. Containers by decreasing order of destination are assigned one by one following the corresponding rule in each case. The computational results provided by Ding and Chou [4] show that their algorithm outperforms that of Avriel et al. [3] and obtains good solutions for the large-size instances tested.

Unlike other implementations of GRASP, the randomization procedure used here does not randomize the selection of the next element to be included in a partial solution, but the constructive process itself. The two constructive algorithms can be randomized, taking into account the way in which they build a feasible solution.

In Avriel et al. [3] the positions for the containers are assigned column by column. We randomize the next column to be assigned at each step and the order in which containers are loaded at port 1. Since the algorithm places the containers by increasing destination, the randomization consists of using a geometric distribution of parameter $1/2$ to choose the sequence. We also randomize the number of containers to be loaded in the Hanging Stage (H), and the containers not assigned in this phase are considered leftover containers to be loaded in the Filling Stage (F).

In Ding and Chou [4] the assignment is made according to the destination of the containers, group by group, following a set of rules. We randomize the algorithm using two strategies. On the one hand, we modify Rule 4. In the original algorithm, if the destination of the container to be loaded is the last port ($j = N$), and the set of empty columns and columns partially filled only with N -containers, denoted by $E(4)$, is not empty, Rule 4 assigns N -containers to the column with the largest number of empty slots in set $E(4)$ until no such loading moves can be carried out. We modify this rule, assigning the N -containers to columns selected at random from $E(4)$. On the other hand, we randomize the order of some rules followed in Step 4. If $j < N$, there are three rules, Rules 5, 6, and 7, that assign containers to columns in such a way that there will not be any blocking containers. These rules are used sequentially. In the randomized algorithm the rules are taken in random order.

5.2. Improvement Phase. In the improvement phase, the solutions of the constructive phase are modified to find better solutions. When defining a move, we have to take into account the specific structure of the multiport problem, in which decisions at one port directly determine the layout at later ports. Looking for a move that was simple to compute and, at the same time, able to modify the solution substantially, we realized that loading operations at port 1 play an important role in the final configuration of the stowage plan. The initial configuration largely determines the position of the empty slots after the unloading operations at the subsequent ports. Therefore, we have developed a local search algorithm based on this idea in which only the configuration at port 1 is subject to changes. At this port, we select two columns c_1 and c_2 , with $c_2 > c_1$. Containers belonging to c_1 are grouped according to their destination port and we define G_j as the set formed by all the j -containers in column c_1 ,

FIGURE 3: Example of neighboring solutions.

$j \in \{2, \dots, N\}$. Let O and D be the sets of containers placed in columns c_1 and c_2 respectively.

A neighboring solution can be obtained by exchanging a homogeneous subset of containers belonging to set O with another subset of the same cardinality, but not necessarily homogenous, from set D . A subset is said to be homogeneous if it is composed of containers with the same destination. As containers with the same destination tend to appear together in the stowage plans, it seems natural to select homogeneous subsets from column c_1 to be moved to other column c_2 . However, it would not be easy to find homogeneous subsets of the same cardinality in c_2 and therefore we allow any subset in this column to be included in the move. The exchange is carried out according to the following steps:

- (1) Take n containers from any group G_j in column c_1 with $j : |G_j| > 0, n \in \{1, \dots, |G_j|\}$. This set is denoted as $S_{c_1}^1$.
- (2) Select another set of n containers in column c_2 with destinations different from j . This set does not have to be homogeneous and it is denoted as $S_{c_2}^1$.
- (3) Let $O' = (O \setminus S_{c_1}^1) \cup S_{c_2}^1$ be the new set of containers to be loaded in column c_1 .
- (4) Let $D' = (D \setminus S_{c_2}^1) \cup S_{c_1}^1$ be the new set of containers to be loaded in column c_2 .
- (5) Place containers of O' in column c_1 and of D' in column c_2 in decreasing order of destination. Note that the remaining columns $c \in \{1, \dots, C\} \setminus \{c_1, c_2\}$ maintain their initial configuration.
- (6) After the exchange between the two columns, complete the solution with one of the constructive algorithms described above.

Figure 3 shows on the left-hand side the configuration at port 1 of one solution for an instance in which $R = 6$, $C = 4$, and $N = 10$. Each square represents a container and the number indicates the port of destination. The figure also shows the possible exchanges to be considered if $c_1 = 2$, $c_2 = 3$, and two containers from the group of 9-containers of c_1 (G_9) are chosen.

The local search algorithm starts from a candidate solution x and then iteratively moves to a neighboring solution $x' \in N(x)$ following a first-improving strategy. The search starts with $c_1 = 1$ and all possible swaps with $c_2 \in \{2, \dots, C\}$ are checked until an improved neighboring solution is found. If an improvement is not obtained, it proceeds with $c_1 = 2$

```

(1) function EXCHANGES( $x, c_1, c_2$ )
(2)   for  $j = 2$  to  $N$  do
(3)     if  $|G_j| > 0$  then
(4)       for  $n = 1$  to  $|G_j|$  do
(5)          $S_{c_1}^1 = \bigcup_{i=1}^n \{j\}$ ;
(6)         COMB( $n, j, c_2$ );
(7)         for all  $S_{c_2}^1$  in COMB( $n, j, c_2$ ) do
(8)            $x' \leftarrow \text{SWAP}(S_{c_1}^1, c_1, S_{c_2}^1, c_2)$ ;
(9)           if  $f(x') < f(x)$  then
(10)            Return  $x'$ ;
(11)   Return  $x$ 

```

ALGORITHM 1: Exchanging containers between c_1 and c_2 .

and $c_2 \in \{3, \dots, C\}$, and continues until all pairs of columns have been considered.

Algorithm 1 shows the pseudo-code of the procedure for exchanging containers between columns c_1 and c_2 of a solution x . The objective function $f(x)$ is the total number of reshuffles. If Algorithm 1 returns x , it has not been able to obtain a better solution. If it finds a better solution x' , the search starts again with the new solution as the initial solution and with columns $c_1 = 1$ and $c_2 = 2$. In line 6 COMB(n, j, c_2) determines all the ways of selecting n containers belonging to column c_2 with destinations different from j . For each selection, $S_{c_2}^1$, a new neighboring solution is built.

6. Computational Experiments

To test the performance of the mathematical model and the heuristic and metaheuristic algorithms, exact solutions were obtained by CPLEX 12.7 considering a time limit of 3600 seconds and heuristic algorithms were run with a time limit of 300 seconds or 1000 iterations. We conducted all computational experiments on virtual machines with 4 virtual processors and 16 GBytes of RAM running Windows 10 Enterprise 64 bits. Virtual machines are run in an OpenStack virtualization platform supported by 12 blades, each with four 12-core AMD Opteron Abu Dhabi 6344 processors running at 2.6 GHz and 256 GB of RAM, for a total of 576 cores and 3 TBytes of RAM.

6.1. Test Instances. We used two sets of instances in which full transportation matrices were randomly generated using the Authentic Matrices Generation Algorithm proposed by Ding and Chou [4]. For each set of parameters, N (ports), R (rows), and C (columns), five different instances were generated. On the one hand, as the instances generated by Ding and Chou [4] were not available, we generated Set I looking for large-size instances. On the other hand, Set II was generated looking for smaller but more computationally difficult instances. Studying Set I we observed that when the number of columns was increased, the problems became easier, with an optimal solution of zero reshuffles in most cases, making it difficult to assess the relative efficiency

of the algorithms. Therefore, we decided to generate Set II with fewer columns but more rows and ports so as to create instances in which the optimal solutions have many reshuffles, in order to obtain more information about the performance of the algorithms.

Set I: 625 instances.

$$\begin{aligned}
 N &\in \{4, 6, \dots, 22\}; \\
 R &= 10; \\
 C &\in \{200, 300, \dots, 2000\};
 \end{aligned} \tag{14}$$

Set II: 840 instances.

$$\begin{aligned}
 N &\in \{4, 6, \dots, 16\}; \\
 R &\in \{6, 8, \dots, 12\}; \\
 C &\in \{2, 4, \dots, 12\}
 \end{aligned} \tag{15}$$

6.2. Performance of the Integer Linear Programming Model. All combinations of valid inequalities were tested in order to decide the best configuration of the integer linear model proposed. In the light of the results, we decided to use the basic model (expressions (3)-(8)), as well as valid inequalities (10) with $r' = 5$.

We compared the performance of our mathematical formulation with the two existing models described in Avriel et al. [3] and Ding and Chou [4] by implementing and running all of them with the default CPLEX, with a time limit of 3600 CPU seconds and using 4 threads. Table 1 shows the results obtained by the three models on the second set of instances, Set II, whose main characteristic is that instances are smaller and have a greater number of reshuffles. The instances in Set I are too large for mathematical formulations. Although all models optimally solve instances of four ports, with zero reshuffles, they are not able to obtain an optimal solution when the number of ports increases.

In Table 1 all instances of 4 and 6 ports in Set II are optimally solved by the three models with averages of 0.1 and 0.6 reshuffles. For 8 ports, Avriel's and Ding and Chou's models provide the same number of optimal solutions and

TABLE 1: Performance of the IP models from Avriel et al. [3] and from Ding and Chou [4] and of the new model on Set II grouped by number of ports (N). Column # contains the number of instances tested in each group and column #Opt contains the number of optimal solutions reached. The average number of reshuffles and average time in seconds are expressed in columns Res. and CPU.

N	#	#Opt	Avriel et al.		#Opt	Ding and Chou		#Opt	New model	
			Res.	CPU		Res.	CPU		Res.	CPU
4	120	120	0.1	0.2	120	0.1	0.2	120	0.1	0.1
6	120	120	0.6	5.7	120	0.6	22.7	120	0.6	1.1
8	120	115	1.2	241.8	115	1.2	311.8	118	1.2	116.7
10	120	90	2.5	1342.2	66	3.1	1927.6	99	2.5	1038.4
12	120	48	6.3	2516.1	35	26.8	2873.1	59	7.4	2172.6
14	120	28	64.1	3001.7	17	116.7	3178.4	36	19.1	2743.8
16	120	9	269.7	3359.8	7	276.5	3422.7	15	68.0	3266.6
Total	840	530	49.2	1495.3	480	60.7	1676.6	567	14.1	1334.2

our model reaches 3 more optimal solutions. For larger numbers of ports, the new model outperforms the other models in terms of both the number of optimal solutions and the average number of reshuffles, with differences increasing as the number of ports increases. However, although the average number of reshuffles in our model is reduced by 74.8% compared to that of Avriel's model, it is still too high, because when the optimal solution is not found, the average number of reshuffles of the best solution found is sometimes very high. Over the 473 instances optimally solved by the three models, our model achieves a reduction in CPU time of 50.42% compared to Avriel's model and of 75.9% compared to Ding's model.

6.3. Selecting the Best Configuration for the GRASP Algorithm.

We first test the constructive algorithms developed for the multiport container ship stowage problem and their randomized counterparts developed in this paper. The GRASP presented here applies the best randomized version in the construction phase and the best constructive approach to complete the solutions in the improvement phase. We then study the effect of the local search by itself on the sets studied and we finally establish the stopping criteria used.

6.3.1. Performance of the Constructive and Randomized Algorithms. We reimplemented the two constructive algorithms proposed by Avriel et al. [3], *SH*, and Ding and Chou [4], *DCA*, and their randomized versions, in C++. In the *SH* algorithm, there are two alternative column selection rules, the function rule and the necessary shift rule. The authors tested these two rules and established that the function rule is better than the necessary shift rule if $R \leq 10$ and $N \geq 10$, and is worse in other cases. Therefore, we apply these two alternatives following that criterion.

As can be seen in Table 2, the average numbers of reshuffles obtained by the deterministic algorithms are very small in all cases in Set I and increase in Set II. The greater the number of ports and the smaller the number of columns, the more difficult the problem. The results shown in the second part of Table 2 underline this statement. It can be seen that the average numbers of reshuffles for groups of 4 to 16 ports in Set II are higher than those for groups with a similar number

of ports in Set I. Although the *SH* algorithm finds more solutions with 0 reshuffles than the *DCA* algorithm, solving 465 instances compared to 264 in Set I and 177 compared to 146 in Set II, *DCA* reduces the number of reshuffles by 56.3% and 11.1% in Sets I and II, respectively. Therefore, the constructive algorithm by Ding and Chou [4] outperforms that by Avriel et al. [3] for both sets.

With respect to the randomization strategies, Table 2 also shows the values of the solutions obtained by each randomized algorithm for the instances in both sets. It can be observed that in all cases the average number of reshuffles decreases compared to their deterministic counterparts. The randomized version of the *SH* algorithm reduces the average number of reshuffles given by its deterministic version by 33.9% and 39.3% on Sets I and II, respectively. On the other hand, randomizing the order of the steps followed in the *DCA* algorithm reduces the average number of reshuffles by 65.3% on Set I and by 30.83% on Set II. According to these results, we select the algorithm by Ding and Chou [4] and its randomization to be part of the GRASP.

6.3.2. Performance of the Local Search. In order to study the effect of the local search described in Section 5.2 by itself, before including it in the GRASP algorithm, we applied it to the solutions obtained using the *DCA* algorithm. Once the new configuration for port 1 has been obtained, the solution is completed using the same algorithm. Table 3 shows the results obtained on both sets. Besides the original local search, denoted in the table as *LS I*, and in order to reduce its running time, we also include a modified version, *LS II*, in which, given a group $G_j : |G_j| > 0$ in column c_1 , instead of considering the exchanges of any number of containers $n \in \{1, \dots, |G_j|\}$ we only consider the exchange of $n = 1$ or $n = |G_j|$ j -containers.

Two points are noticeable in this table. First, the local search is able to find solutions for all the groups with a significantly lower average number of reshuffles, from 4.9 to 3.31 on Set I, and from 12 to 4.83 on Set II. Second, with the modified version of the local search, the average number of reshuffles is reduced by 24.8% on Set I and slightly increased by 2.1% on Set II. Nevertheless, the running times are reduced by 10.5% and 71.7% on Sets I and II, respectively. We therefore

TABLE 2: Algorithm performance of the constructive algorithms from Avriel et al. [3] and from Ding and Chou [4] and their randomized versions on both sets grouped by number of ports (N). Column # shows the number of instances tested in each group, and column # Z shows the total number of solutions found with 0 reshuffles. The average number of reshuffles and average time in seconds are expressed in columns Res. and CPU.

N	#	Avriel et al.						Ding and Chou					
		Deterministic, SH			Randomized			Deterministic			Randomized		
		# Z	Res.	CPU	# Z	Res.	CPU	# Z	Res.	CPU	# Z	Res.	CPU
Set I													
4	95	95	0.0	0.0	95	0.0	0.0	86	0.2	0.1	95	0.0	0.8
6	95	94	0.0	0.0	94	0.0	0.1	69	0.6	0.1	93	0.1	5.0
8	95	86	0.3	0.0	87	0.3	0.5	63	0.7	0.1	91	0.1	5.1
10	95	73	1.5	0.0	74	1.3	4.4	16	2.1	0.1	82	0.2	33.2
12	95	55	2.6	0.0	58	2.3	8.7	22	3.0	0.1	71	0.5	48.7
14	95	35	5.8	0.0	39	4.0	18.7	4	5.2	0.1	58	1.0	89.8
16	95	13	10.1	0.0	19	6.6	39.8	2	5.9	0.1	41	1.7	123.0
18	95	9	16.5	0.0	14	11.1	74.2	1	7.7	0.1	35	2.1	136.9
20	95	5	29.4	0.0	8	18.0	115.5	0	11.2	0.1	8	4.9	190.5
22	95	0	45.5	0.1	2	30.9	208.5	1	12.7	0.1	16	6.1	172.1
Total	950	465	11.2	0.0	490	7.4	47.0	264	4.9	0.1	590	1.7	80.5
Set II													
4	120	116	0.1	0.0	116	0.1	0.0	104	0.2	0.0	110	0.1	0.0
6	120	47	2.4	0.0	54	1.7	0.0	33	2.3	0.0	58	1.4	0.0
8	120	13	5.6	0.0	23	3.1	0.1	9	5.9	0.0	22	3.5	0.1
10	120	1	11.1	0.0	9	6.4	0.1	0	11.1	0.0	1	7.7	0.2
12	120	0	18.3	0.0	0	10.4	0.2	0	15.7	0.0	0	10.6	0.2
14	120	0	25.3	0.0	1	15.9	0.3	0	21.1	0.0	0	15.2	0.3
16	120	0	32.1	0.0	0	20.0	0.4	0	27.8	0.0	0	20.0	0.4
Total	840	177	13.5	0.0	203	8.2	0.2	146	12.0	0.0	191	8.3	0.2

select this *LS II* version for the final configuration of the GRASP algorithm.

6.4. Results of the Complete GRASP Algorithm. Table 4 presents the aggregated results of GRASP for each group in Set I and Set II and summarizes the number of times that GRASP achieves a solution of zero reshuffles and the number of times that it achieves optimal solutions, either because they match an optimal solution provided by one or more of the IP models or because the solution has zero reshuffles. Differences between columns #Z and #Opt. only appear in Set II, on which the integer models have been tested.

With respect to Set I, GRASP is able to solve the majority of these instances, 603 out of 950, to proven optimality. For the remaining instances, the number of reshuffles in the solutions is greater than zero. Although their optimality cannot be proven, it is possible that some of them are in fact optimal, further increasing the total number of optimal solutions found by GRASP.

The instances from 4 to 14 ports are solved with an average number of reshuffles lower than or equal to 1 in an average time of 49.47 seconds. The instances with 16 and 18 ports are solved with an average lower than 2 and those of 20 and 22 ports with averages of 4.1 and 4.3 reshuffles. The average number of reshuffles in Set I obtained with GRASP is just 1.3, while that obtained with the *DCA* algorithm, the

best according to Section 6.3.1, is 4.9. The results obtained with GRASP are also better than that obtained with the randomized version of *DCA*, 1.3 compared to 1.7 reshuffles, and with the local search technique *LSII*, 1.3 compared to 2.49. As expected, the stopping criterion that prevails when the number of columns and ports is so high is the time limit of 300 seconds.

The situation with Set II is somewhat different, because this set is composed of instances with larger numbers of reshuffles. The average number of reshuffles is more than twice that of Set I and the percentage of solutions with zero reshuffles decreases from 63.47% to 40.35%. While the average number of reshuffles for this set with the *DCA* algorithm is 12, and 8.3 with its randomized version, the average with GRASP is 4.2 reshuffles. With up to 10 ports, the new model provides slightly better solutions than GRASP, but for instances with 12 ports, GRASP outperforms the model with an average of 5 reshuffles instead of 7.4. The difference increases as the number of ports increases; in instances with 16 ports GRASP obtains an average of 11.6 reshuffles while for the model it is 68. Although the time limit was set at 300 seconds, the average time for all groups of instances in Set II is much shorter. In fact, GRASP finishes after 300 seconds only in 6 instances of 8 ports, 45 of 10 ports, 51 of 12 ports, 55 of 14 ports, and 83 of 16 ports. In most cases, the algorithm stops after 1000 iterations.

TABLE 3: Comparing local search procedures on both sets grouped by number of ports. Column # shows the number of instances tested in each group, and column # Z shows the total number of solutions found with 0 reshuffles. The average number of reshuffles and average time in seconds are expressed in columns Res. and CPU.

LSI					LSII		
N	#	# Z	Resh.	CPU	# Z	Resh.	CPU
Set I							
4	95	94	0.01	8.68	91	0.04	13.68
6	95	83	0.20	51.25	86	0.13	37.82
8	95	89	0.11	44.60	93	0.03	17.85
10	95	55	0.76	169.43	74	0.40	93.59
12	95	40	1.61	193.09	50	1.20	161.20
14	95	23	3.18	252.07	28	2.32	235.39
16	95	11	4.15	274.81	20	2.97	252.85
18	95	4	5.73	290.97	11	4.14	279.48
20	95	1	8.19	299.35	2	6.28	296.89
22	95	1	9.21	298.85	3	7.39	295.80
Total	950	401	3.31	188.31	458	2.49	168.46
Set II							
4	120	115	0.07	0.00	115	0.07	0.00
6	120	78	0.81	0.08	81	0.79	0.01
8	120	53	1.69	0.60	54	1.79	0.12
10	120	16	3.81	3.10	14	4.01	0.79
12	120	6	6.00	5.90	7	6.02	1.66
14	120	4	8.76	7.77	4	8.74	2.65
16	120	0	12.71	15.22	1	13.08	4.00
Total	840	272	4.83	4.67	276	4.93	1.32

TABLE 4: Results obtained with the GRASP algorithm on both sets grouped by number of ports (N). Column # shows the number of instances tested in each group, column #Z shows the total number of solutions found with 0 reshuffles, and #Opt. shows the total number of optimal solutions. The average number of reshuffles and average time in seconds are expressed in columns Res. and CPU.

GRASP					
N	#	#Z	#Opt.	Res.	CPU
Set I					
4	95	95	95	0.0	1.1
6	95	93	93	0.1	10.7
8	95	93	93	0.0	8.5
10	95	84	84	0.1	48.0
12	95	72	72	0.4	87.7
14	95	54	54	1.0	140.8
16	95	41	41	1.6	183.1
18	95	39	39	1.8	194.8
20	95	12	12	4.1	271.9
22	95	20	20	4.3	253.7
Total	950	603	603	1.3	120.0
Set II					
4	120	116	117	0.1	0.0
6	120	88	89	0.7	4.5
8	120	70	74	1.3	32.3
10	120	31	33	3.1	115.7
12	120	14	14	5.0	168.1
14	120	14	15	7.6	193.1
16	120	6	9	11.6	217.1
Total	840	339	351	4.2	104.4

The results show the effectiveness of the GRASP we propose in this study. It obtains extremely good solutions in very short computing times, less than 5 minutes for the larger instances. These solutions are clearly better than those provided by other heuristic methods, which could be used to obtain acceptable solutions if extremely fast answers were needed. Integer models can only be used for small instances, and even in the cases in which the models can prove optimality, the solutions obtained by GRASP match or are very close to the optimal solutions. For medium and large-size instances, the integer models fail to obtain feasible solutions in many cases, and when they do obtain feasible solutions these are much worse than those provided by GRASP.

7. Conclusions and Future Work

Efficient stowage plans for increasingly larger container ships have a critical impact on the performance of operations at container terminals. The problem is being studied from various different perspectives, using a range of decomposition strategies and algorithmic approaches. In this article we have chosen to study the complete multiport problem, considering loading and unloading operations along the route, and have studied it from two complementary points of view. First, we have developed a new integer model and compared it with existing proposals in an extensive computational study. The conclusion is that, though our model performs consistently better than previous models, none of them is able to produce acceptable solutions for large-size instances. Therefore, in a second part we have developed a GRASP algorithm with new features, such as a randomizing strategy that randomizes the steps of the existing constructive algorithms. A new local search, tailored to the problem, improves the solutions of the randomized constructive phase and allows the GRASP to obtain many optimal solutions and extremely good solutions for instances for which optimality cannot be proven, clearly outperforming previous heuristic algorithms. Our conclusion is that the GRASP framework is very well suited to this problem because using a constructive algorithm is a good way of studying the sequence of operations in which the empty spaces left when unloading containers are then used in the loading operations. By adding randomization and a local search phase we achieve a very efficient algorithm for this complex problem.

The version of the problem considered in this study can be extended in several ways, progressively adding characteristics of the real problem. The two most important characteristics are the container type and weight. Considering the joint stowage of 20-foot and 40-foot containers introduces new constraints in the way they can be stacked. Reefer containers have only a limited set of available slots, those with electric plugs, high-cube containers reduce the number of tiers in a stack, and containers with dangerous goods bring new strict stowage conditions. In real problems these container types have to be considered and their characteristics taken into account by proposed solution procedures. Container weight is also of utmost importance, because it directly affects

stability conditions that have to be respected for safety reasons. Although these features can be included in the integer linear model, the results obtained clearly indicate that the extended model will not be able to solve real-world instances. Extending the heuristic and metaheuristic algorithms seems a more promising approach, because their flexibility and modest computing time requirements make them able to cope with more complex versions of the problem.

Data Availability

The datasets Set I and Set II used to support the findings of this study are available from the corresponding author (consuelo.parreno@uv.es) upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This study has been partially supported by the Spanish Ministry of Education, Culture, and Sport, FPU Grant A-2015-12849, by the Spanish Ministry of Economy and Competitiveness, DPI2014-53665-P, cofinanced by FEDER funds, and by the Generalitat Valenciana, PROMETEO/2013/049, Grant CPI-13-351.

References

- [1] UNCTAD, "Review of Maritime Transport," 2017.
- [2] B. D. Brouer, C. V. Karsten, and D. Pisinger, "Optimization in liner shipping," *4OR*, vol. 15, no. 1, pp. 1–35, 2017.
- [3] M. Avriel, M. Penn, N. Shpirer, and S. Witteboon, "Stowage planning for container ships to reduce the number of shifts," *Annals of Operations Research*, vol. 76, pp. 55–71, 1998.
- [4] D. Ding and M. C. Chou, "Stowage planning for container ships: A heuristic algorithm to reduce the number of shifts," *European Journal of Operational Research*, vol. 246, no. 1, pp. 242–249, 2015.
- [5] OOCL, <https://www.oocl.com>, 2017.
- [6] D. Ambrosino, D. Anghinolfi, M. Paolucci, and A. Sciomachen, "An experimental comparison of different heuristics for the master bay plan problem," in *Proceedings of the International Symposium on Experimental Algorithms*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pp. 314–325.
- [7] A. Sciomachen and E. Tanfani, "The master bay plan problem: a solution method based on its connection to the three-dimensional bin packing problem," *IMA Journal of Management Mathematics*, vol. 14, no. 3, pp. 251–269 (2004), 2003.
- [8] A. Sciomachen and E. Tanfani, "A 3D-BPP approach for optimising stowage plans and terminal productivity," *European Journal of Operational Research*, vol. 183, no. 3, pp. 1433–1446, 2007.
- [9] D. Ambrosino, M. Paolucci, and A. Sciomachen, "Experimental evaluation of mixed integer programming models for the multi-port master bay plan problem," *Flexible Services and Manufacturing Journal*, vol. 27, no. 2-3, pp. 263–284, 2015.

- [10] D. Pacino, A. Delgado, R. M. Jensen, and T. Bebbington, *Fast Generation of Near-Optimal Plans for Eco-Efficient Stowage of Large Container Vessels*, vol. 6971 of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2011.
- [11] A. Delgado, R. M. Jensen, K. Janstrup, T. H. Rose, and K. H. Andersen, "A constraint programming model for fast optimal stowage of container vessel bays," *European Journal of Operational Research*, vol. 220, no. 1, pp. 251–261, 2012.
- [12] F. Parreño, D. Pacino, and R. Alvarez-Valdes, "A GRASP algorithm for the container stowage slot planning problem," *Transportation Research Part E: Logistics and Transportation Review*, vol. 94, pp. 141–157, 2016.
- [13] J. Lehnfeld and S. Knust, "Loading, unloading and premarshalling of stacks in storage areas: survey and classification," *European Journal of Operational Research*, vol. 239, no. 2, pp. 297–312, 2014.
- [14] I. D. Wilson and P. A. Roach, "Container stowage planning: A methodology for generating computerised solutions," *Journal of the Operational Research Society*, vol. 51, no. 11, pp. 1248–1255, 2000.
- [15] J.-G. Kang and Y.-D. Kim, "Stowage planning in maritime container transportation," *Journal of the Operational Research Society*, vol. 53, no. 4, pp. 415–426, 2002.
- [16] Z. Wei-ying, L. Yan, and J. Zhuo-shang, "Model and algorithm for container ship stowage planning based on bin-packing problem," *Journal of Marine Science and Application*, vol. 4, no. 3, pp. 30–36, 2005.
- [17] D. Ambrosino, M. Paolucci, and A. Sciomachen, "Computational evaluation of a MIP model for multi-port stowage planning problems," *Soft Computing*, vol. 21, no. 7, pp. 1753–1763, 2017.
- [18] D. Ambrosino, A. Sciomachen, and E. Tanfani, "Stowing a containership: The master bay plan problem," *Transportation Research Part A: Policy and Practice*, vol. 38, no. 2, pp. 81–99, 2004.
- [19] O. Dubrovsky, G. Levitin, and M. Penn, "A genetic algorithm with a compact solution encoding for the container ship stowage problem," *Journal of Heuristics*, vol. 8, no. 6, pp. 585–599, 2002.
- [20] M. G. Resende and C. C. Ribeiro, *Optimization by GRASP*, Springer, New York, NY, USA, 2016.
- [21] J. Francisco Correcher, M. Teresa Alonso, R. Alvarez-Valdes, and P. Francisco, "olving a large multicontainer loading problem in the car manufacturing industry," *Computers and Operations Research*, vol. 82, pp. 139–152, 2017.
- [22] J. A. Díaz, D. E. Luna, J.-F. Camacho-Vallejo, and M.-S. Casas-Ramírez, "GRASP and hybrid GRASP-Tabu heuristics to solve a maximal covering location problem with customer preference ordering," *Expert Systems with Applications*, vol. 82, pp. 67–76, 2017.
- [23] S. C. Ho and W. Y. Szeto, "GRASP with path relinking for the selective pickup and delivery problem," *Expert Systems with Applications*, vol. 51, pp. 14–25, 2016.
- [24] Matsaini and B. Santosa, "Solving the container stowage problem (CSP) using particle swarm optimization (PSO)," *IOP Conference Series: Materials Science and Engineering*, vol. 337, article no. 012002, 2018.
- [25] A. T. Azevedo, L. L. S. Neto, A. A. Chaves, and A. C. Moretti, "Solving the 3D stowage planning problem integrated with the quay crane scheduling problem by representation by rules and genetic algorithm," *Applied Soft Computing Journal*, vol. 65, pp. 495–516, 2018.
- [26] A. T. De Azevedo, C. M. Ribeiro, G. J. De Sena, A. A. Chaves, L. L. S. Neto, and A. C. Moretti, "Solving the 3D container ship loading planning problem by representation by rules and meta-heuristics," *International Journal of Data Analysis Techniques and Strategies*, vol. 6, no. 3, pp. 228–260, 2014.
- [27] M. A. Yurtseven, E. Boulougouris, and O. Turan, "Container ship stowage plan using steepest ascent hill climbing, genetic, and simulated annealing algorithms," *Marine Design XIII*, vol. 1, pp. 617–623, 2018.

Submit your manuscripts at
www.hindawi.com