European Journal of Operational Research 246 (2015) 242–249

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Innovative Applications of O.R.

Stowage planning for container ships: A heuristic algorithm to reduce the number of shifts

Ding Ding[a,*], Mabel C. Chou[b]

[a] School of International Trade and Economics, University of International Business and Economics, Beijing 100029, PR China
[b] Department of Decision Sciences, NUS Business School, National University of Singapore, Singapore 119245, Singapore

ARTICLE INFO

Article history:
Received 10 July 2013
Accepted 30 March 2015
Available online 7 April 2015

Keywords:
Stowage planning
Container movement
Shifts
Heuristic algorithm

ABSTRACT

We consider the stowage planning problem of a container ship, where the ship visits a series of ports sequentially and containers can only be accessed from the top of the stacks. At some ports, certain containers will be unloaded temporarily and will be loaded back later for various purposes. Such unproductive movements of containers are called shifts, which are both time and money consuming. Literature shows that binary linear programming formulation for such problems is impracticable for real life problems due to the large number of binary variables and constraints. Therefore, we develop a heuristic algorithm which can generate stowage plans with a reasonable number of shifts for such problems. The algorithm, verified by extensive computational experimentations, performs better than the Suspensory Heuristic Procedure (SH algorithm) proposed in Avriel et al. (1998), which, to the best of our knowledge, is one of the leading heuristic algorithms for such stowage planning problem.

1. Introduction

Containerization has revolutionized cargo shipping. According to World Shipping Council, around 52 percent of the value of world international seaborne trade today is being moved in containers. The world container ports' throughput grew by 5.58 percent to reach 651 million TEUs (abbreviation for twenty-foot equivalent units) in 2013 after a 4.97 percent growth in 2012 and 8.63 percent in 2011. Together the top 20 world container ports handled 299 million TEUs in 2013, which made up 46 percent of the world total. In relation to containerized trade, the global containership fleet expanded at an average annual growth rate of 7.9 percent over the period 2006–2013 and the total fleet amounted to about 19.9 million TEUs in 2013. The average carrying capacity of container ships grew from 2259 TEUs in January 2004 to 3161 TEUs in January 2009, and to 4449 TEUs in January 2014 (all data are taken from UNCTAD, 2014). Today, the largest container ships in the world are able to carry as many as 19224 TEUs (Wikipedia, 2015).

Container ship commonly calls at a series of ports according to its planned route, and at each port containers destined for that port are unloaded and containers destined for subsequent ports are loaded. Note that containers are only accessible when they are on the top of other containers. Unloading and loading movements of same containers in the same port are called shifts. Shifts arise either when we want to unload containers destined for current port which however are beneath those destined for subsequent ports, or when we want to reorder the sequence of containers to prevent more shifts in the future. Usually it is called necessary shifts in the former case and voluntary shifts in the latter.

Shifts are time-consuming and money-consuming activities, hence the arrangement of containers on board is crucial to achieve operations efficiency by reducing the number of shifts. The task of determining the arrangement of containers is called stowage planning or master bay planning (Ambrosino et al., 2004; GDV, 2008). In addition to the accessibility constraints mentioned previously, many other constraints have impacts on the stowage planning, such as horizontal and cross stability constraints of the container ship, weight constraints of containers when they are stacked on each other and on the decks, mixture constraints of containers when different sizes of containers are stacked together, electricity supply constraints of temperature controlled containers, space constraints of containers for hazardous cargo, and so on.

Aslidis (1989) solved one stack over-stowage problem by a dynamic programming algorithm with the kernel of a rearrangement policy which is widely adopted in later works. Avriel and Penn (1993) presented a binary linear programming formulation to find optimal solution for stowage in a single rectangular bay with only accessibility constraint. This method is limited due to the huge

* Corresponding author. Tel.: +86-10-64493325.
E-mail addresses: dingd.cn@gmail.com (D. Ding), mabelchou@nus.edu.sg (M.C. Chou).

http://dx.doi.org/10.1016/j.ejor.2015.03.044
0377-2217/

number of binary variables and constraints needed. Binary linear programming formulation for stowage planning problems with stability constraints, weight constraints, mixture constraints, etc., in addition to the accessibility constraints, can be found in Botter and Brinati (1992), Ambrosino et al. (2004), Ambrosino et al. (2006), Sciomachen and Tanfani (2007), Delgado et al. (2012) and Ding and Chou (2015). However, Botter and Brinati (1992) and Avriel et al. (2000) showed that the stowage planning problems are NP-complete, and currently available versions of formulations all require extensive time and resources to be solved even with the world's leading optimization softwares.

Avriel et al. (1998) proposed the Suspensory Heuristic Procedure (SH algorithm hereafter) which is a dynamic slot-assignment scheme that terminated with a stowage plan, which, to the best of our knowledge, is one of the leading heuristic algorithms for such stowage planning problem. Various optimization search algorithms such as genetic algorithm (Dubrovsky et al., 2002; Yang & Kim, 2006), branch and bound search (Sciomachen & Tanfani, 2003; Wilson & Roach, 2000), tabu search (Ambrosino et al., 2009; Wilson & Roach, 1999, 2000; Wilson et al., 2001), simulated annealing (Dubrovsky et al., 2002), greedy algorithm (Kang & Kim, 2002), tree search algorithm(Kang & Kim, 2002; Zhang et al., 2005) etc., have been applied in similar problems. However, the efficiencies and complexities of these algorithms still need to be verified. Readers are referred to Zhang et al. (2008), Sciomachen and Tanfani (2007) and Delgado et al. (2012) for extensive literature reviews.

In this paper, we propose a heuristic algorithm, which is shown to be more efficient than the SH algorithm in Avriel et al. (1998) especially when the number of ports visited or the volume capacity of the ship is large. Extensive computational experimentations show that our algorithm generally performs better than the SH algorithm. Furthermore, SH algorithm is only applicable when loading and unloading information for all ports are obtained, while our algorithm only requires such information for the current port to make a stowage plan.

The rest of paper is organized as follows. Section 2 provides a clear definition of the stowage planning problem of a containership. Section 3 introduces the main heuristic algorithm together with an example illustrating how the algorithm works. Section 4 analyzes the performance of algorithms based on extensive computational experimentations and Section 5 provides concluding remarks.

2. Problem description

The most frequently used addressing notation for storage locations in a container ship is the bay-row-tier system, please refer to Ambrosino et al. (2004) for an excellent detailed description. In this paper, we adopt similar notation and definitions as in Avriel et al. (1998) since we mainly compare our results with theirs and similar notation systems will make the comparison more readable. Please note that the two notation systems can be translated into each other easily.

Consider a container ship consisting of a single rectangular bay with horizontal rows labeled $r = 1,\ldots,R$ from top to bottom and vertical columns labeled $c = 1,\ldots,C$ from left to right. Slot in row r and column c is labeled (r, c). Assume that all the containers are of the same size, (e.g. 20 TEUs) and each slot can be fully occupied by one container. The volume capacity of the container ship is thus $P = R \cdot C$. The assumption of single rectangular bay is just for sake of simplicity. For multiple bays, we can simply label all the columns sequentially as the formulation does not take into account where the columns are located. For irregular bays, they can be transformed into rectangular ones with some additional imaginary slots, which by additional constraints are forced to be occupied by imaginary containers during the whole voyage.

Assume that a container ship plans to visit $N(\geq 2)$ ports, $1, 2,\ldots, N$, sequentially. At port 1, containers destined for port $2,\ldots, N$ are loaded. At each port $i = 2,\ldots, N-1$, all containers destined for port i are unloaded and those destined for ports $i+1,\ldots, N$ are loaded. Shifts may occur in these ports, either necessary or voluntary or both. At port N, all containers destined for port N are unloaded. The containers remain unmoved when the ship voyages among the ports.

A container is called a j-container if it is destined for port j. Obviously all j-containers are loaded before the ship visits port j and should be unloaded at port j. A j-container is called a i, j-container if it is originated from port i. A j-container in slot (r, c) is said to be blocked if there exists a j'-container in slot (r', c) with $j' > j$ and $r' < r$, and the j'-container is called a blocking container with respect to the j-container. Occurrence of container blocking at any time means that shifts are no longer avoidable.

Let $\mathbf{T} = [T_{ij}]$ be the $N \times N$ transportation matrix, where T_{ij} is the number of i, j-containers to be shipped, where $i,j \in \{1,\ldots, N\}$. Note that $T_{ij} \geq 0$ for all i,j and $T_{ij} = 0$ for all $i \geq j$. A transportation matrix is called feasible if in each port the total capacity required to stow the containers destined for subsequence ports is no more than the capacity of the ship. Technically, the non-negative upper triangular $\mathbf{T}$ is feasible if and only if

$$\sum_{k=1}^{i}\sum_{j=i+1}^{N} T_{kj} \leq P, \quad \forall\, i \in \{1,\ldots, N-1\} \tag{1}$$

If $\mathbf{T}$ is not feasible, decisions on which containers are to be loaded on board have to be made, which is beyond the scope of our consideration. Therefore, only feasible transportation matrices are considered throughout this paper.

Let $K = \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} T_{ij}$ be the total number of containers to be shipped, and L and U be the total number of loading and unloading movements during the whole voyage, respectively. Note that $L = U$ if there is no containers loaded on board before visiting the first port. Since each container requires at least one loading and one unloading movement, the total number of shifts $Z = L + U - 2K \geq 0$. Moreover, define $Z/(2K)$ as the shifting ratio, which is a relative performance measure for stowage plans.

Remark. We have

(1) If $N \leq 3$, for any given $\mathbf{T}$, R and C, there always exist stowage plans that lead to no shifts;
(2) If $R = 1$, for any given $\mathbf{T}$, N and C, all stowage plans lead to no shifts;
(3) If $N \geq 4$ and $R \geq 2$, for any given C, there always exists $\mathbf{T}$ such that shifts are inevitable.

Please refer to the online supplement (Ding & Chou, 2015) for proof.

This remark indicates that we only need to focus on stowage planning problems with $N \geq 4$ and $R \geq 2$. Also, we will emphasize cases with $C \gg 1$ for real life problems, noticing that $C = 1$ has been well studied by Aslidis (1989).

3. The heuristic algorithm

3.1. Preliminary

A column is called eligible if it is neither empty nor full. In a non-empty column, adjacent j-containers are said to form a j-layer. The topmost layer in a column is called the 1^{st} layer, and the adjacent layer beneath the k^{th} layer is called the $(k+1)^{th}$ layer. If the k^{th} layer is at the bottom of the column, we regard that the $(k+1)^{th}$ layer is a N-layer and there will be no $(k+2)^{th}$ layer. A non-empty column is called l-top if the 1^{st} layer is a l-layer, and it is called l-min if the container with the nearest destination in this column is a l-container.

A column or a stack of containers is said to be in order if there is no blocked container in it, otherwise it is said to be out of order. The blocking number of a column is the number of necessary shifts that must be carried out at future ports assuming that no containers will ever be loaded into this column. A column is said (re)ordered up to m if there is no blocked l-container with $l \le m$ in it. For a column being ordered up to m, all l-containers with $l > m$ (if exist) are stowed under those with $l \le m$ (if exist), and the latter itself is in order. Avriel et al. (1998) has similar definition of "reordered up to" which is also called "rearranged up to" in Aslidis (1989).

An intuitive conclusion, which has also been proven in Aslidis (1989), is that when several containers are to be loaded into one column, it is always better to load them in order, which means containers with further destinations should be loaded before those with closer destinations. Sometimes, we may want to unload a certain number of disordered containers before the loading movements on the purpose of reducing the number of shifts in the future. In our algorithm, such voluntary unloading movements are determined by the following Myopic Voluntary Shifting Determination (MVSD) procedure.

Consider the situation that we are trying to load a set of containers, called the trying set, into an eligible column c at port i. Assume that i-containers and those blocking them have already been unloaded. Note here the number of containers in trying set can exceed the number of empty slots in column c as we are just trying. Assume that the nearest destination for the containers in the trying set is port k, and the furthest destination for the containers in trying set and in column c is port u.

Before any containers in trying set is loaded into column c, we may first unload some from column c to reorder the column up to m in order to reduce the number of shifts in the future. After that the unloaded containers destined for port $j \ge k$ form a new set, called the unloaded set, while those unloaded containers destined for port $j < k$ are ignored (as our main algorithm deals with containers in a decreasing order of their destinations, they will be handled later). Finally, the containers in the trying set and the unloaded set will be loaded together into the column c in order and as many as possible, for containers with the same destinations, those in the unloaded set will be loaded before those in the trying set.

When $m \le k-1$ we indeed unload nothing, since a column containing only l-containers with $l \ge k > m$ is already ordered up to m. On the other hand, the process will be same for all $m \ge u$. Hence we only need to consider the reordering level $m \in \{k-1, \ldots, u\}$. For each reordering level m, the minimum number of unloading movements required to achieve that level is assumed to be $G(m)$, obviously $G(m+1) \ge G(m)$. Moreover, define $H(m)$ as the maximum number of containers in the trying set that can be loaded into column c after the unloading movement. Let $B(0)$ be the blocking number of column c before any movement, and $B(m)$ be the blocking number of column c after all these movements. Therefore, we can settle down $H(m)$ containers in the trying set at myopic cost of $B(m) + G(m) - B(0)$. The MVSD procedure selects the reordering level m with the largest $H(m)$ among all considered m, or the m with the smallest $B(m) + G(m)$ among those m with the same largest $H(m)$, or the smallest m with the smallest $B(m)$ among those m with the same largest $H(m)$ and the same smallest $B(m) + G(m)$. Please refer to the online supplement (Ding & Chou, 2015) for an example of the MVSD procedure.

3.2. The main algorithm

Given N, R, C and a feasible $\mathbf{T}$, our main algorithm generates a series of matrices, $\{\mathbf{S}^i\}$, $i \in \{1, \ldots, N-1\}$, where $\mathbf{S}^i$ is a $R \times C$ matrix corresponding to the stowage plan, that is, the status of the bay, upon leaving port i. Let $\mathbf{S}^0$ be the initial stowage plan before the voyage which will be a $R \times C$ matrix of 0 for an empty bay, and let the loading vector for port i be $\boldsymbol{w}^i = [\boldsymbol{w}_{ij}]$, $i \in \{1, \ldots, N-1\}$, $j \in \{i+1, \ldots, N\}$, where $\boldsymbol{w}_{ij}$ is the number of j-containers to be loaded on board at port i. At each port i the initial values of $\boldsymbol{w}^i$ are $\boldsymbol{w}_{ij} = T_{ij}$ before any container movement starts. Note that $\boldsymbol{w}_{ij}$ will be increased or decreased by 1 for each unloading or loading movement for j-container respectively, and it should always be ended up with 0 upon leaving port i. Our algorithm generates $\mathbf{S}^i$ based on $\mathbf{S}^{i-1}$ and $\boldsymbol{w}^i$ from $i = 1$ to $N-1$ sequentially. Note that $\mathbf{S}^i$, $\boldsymbol{w}^i$ and the status of the columns are updated live after each container movement, and hence will not be mentioned repeatedly in the algorithm.

During the execution of the algorithm, we sometimes have several qualified columns that can be used to stow the containers that we are dealing with and we have to specify in which order these columns are filled. When we say "the columns are filled based on the priority sequence, $p_1, p_2, \ldots, p_k$", we mean that: (1) among all the qualified columns, the column that satisfies p_1 will be filled first; (2) among all the columns satisfying $p_1, \ldots, p_{i-1}$, the column that satisfies p_i will be filled first; and (3) among all the columns satisfying $p_1, \ldots, p_k$, any column can be filled first.

Main Algorithm.

Input. $\mathbf{S}^0$, $\mathbf{T}$, N;

Step 1. Let $i = 1$;

Step 2. Let $\mathbf{S}^i = \mathbf{S}^{i-1}$ and $\boldsymbol{w}^i = [\boldsymbol{w}_{ij}]$ where $\boldsymbol{w}_{ij} = T_{ij}, j \in \{i+1, \ldots, N\}$;

Step 3. Unload all i-containers and those blocking them;

Step 4. Load containers on board in a descending order of their destinations as follows:

Step 4.1. Let $j = N$;

Step 4.2. Follow the undermentioned rules sequentially whenever applicable;

Rule 1. If $\boldsymbol{w}_{ij} = 0$. Let $j = j-1$ and go to **Step 4.3**;

Rule 2. If there exists only one empty column in the bay. Load all unassigned containers into that column in order. Let $j = i$ and go to **Step 4.3**;

Rule 3. If there exists only one eligible column in the bay. Apply the MVSD procedure to determine $G(m)$, the number of containers to be unloaded. Unload $G(m)$ containers from that column, then load all unassigned containers into that column in order. Let $j = i$, and go to **Step 4.3**;

Rule 4. If $j = N$, and the set of empty columns and eligible N-top in order columns, denoted by $E(4)$, is not empty. Load one N-container into the column with the most number of empty slots in set $E(4)$ repeatedly until no such loading movement can be carried out. If $\boldsymbol{w}_{ij} = 0$, let $j = j-1$ and go to **Step 4.3**;

Rule 5. If $j < N$, and the set of eligible j-top in order columns, denoted by $E(5)$, is not empty. Load as many j-containers as possible into columns in set $E(5)$ based on the following priority sequence: with the least number of empty slots, with the furthest destination of the 2^{nd} layer containers, with the most number of the 1^{st} layer containers, ..., (repeating the last two items by increasing the index by 1 at each time), until no such loading movements can be carried out. If $\boldsymbol{w}_{ij} = 0$, let $j = j-1$ and go to **Step 4.3**;

Rule 6. If $j < N$, and the set of empty columns, denoted by $E(6)$, is not empty. Load as many j-containers as possible into the column with the most number

of empty slots in set $E(6)$ until no such loading movements can be carried out. If $\boldsymbol{w}_{ij} = 0$, let $j = j - 1$ and go to **Step 4.3**;

Rule 7. If $j < N$, and the set of all eligible l-top ($l > j$) in order columns, denoted by $E(7)$, is not empty. Load as many j-containers as possible into columns in set $E(7)$ based on the following priority sequence: with the furthest destination of the 1^{st} layer containers, with the most number of empty slots, with the furthest destination of the 2^{nd} layer containers, with the most number of the 1^{st} layer containers, ..., (repeating the last two items by increasing the index by 1 at each time), until no such loading movements can be carried out. If $\boldsymbol{w}_{ij} = 0$, let $j = j - 1$ and go to **Step 4.3**;

Rule 8. If $j < N$, and the set of all eligible l-min ($l \geq j$) out of order columns, denoted by $E(8)$, is not empty. Load as many j-containers as possible into columns in set $E(8)$ based on the following priority sequence: with the furthest destination of the 1^{st} layer containers, with the most number of empty slots, with the furthest destination of the 2^{nd} layer containers, with the most number of the 1^{st} layer containers,..., (repeating the last two items by increasing the index by 1 at each time), until no such loading movements can be carried out. All tagged containers in those filled columns will be untagged. If $\boldsymbol{w}_{ij} = 0$, let $j = j - 1$ and go to **Step 4.3**;

Rule 9. Let $E(9)$ denote the set of all eligible columns. For each column, the trying set contains all unassigned j-containers and all the tagged containers in that column, and those tagged containers are regarded as being not in that column. Apply the MVSD procedure to determine the reordering level for each column. Sort the columns in set $E(9)$ based on the following priority sequence: with the largest $H(m)$, with the smallest $B(m) + G(m) - B(0)$, with the largest $G(m)$. Only the first column is filled in the way being described in the MVSD procedure. If for the column filled $G(m) = 0$, the containers loaded into that column from the trying set will be all tagged. If $G(m) > 0$, all tagged containers in that column will be untagged. Repeat the previous process until $\boldsymbol{w}_{ij} = 0$. Let $j = j - 1$ and go to **Step 4.3**;

Rule 10. If $j = i + 1$, load all unassigned j-containers into the remaining empty slots by any feasible way, this always works as $\mathbf{T}$ is feasible. Let $j = i$ and go to **Step 4.3**;

Step 4.3. If $j \geq i + 1$, go to **Step 4.2**;

Step 5. Record $\mathbf{S}^i$, which is the stowage plan upon leaving port i;

Step 6. If $i < N - 1$, let $i = i + 1$ and go to **Step 2**;

Output. $\mathbf{S}^i$, $i \in \{1, \ldots, N-1\}$.

Please note that in the main algorithm a tagged container is a container with a tag (or a mark) reminding us that it requires further consideration. A container may be tagged (marked) in Rule 9 and a tagged container may be untagged in Rules 8 and 9.

Here are some remarks on the rules in the core of the main algorithm. **Rule 1** states that if there is no j-container to be loaded at all then the algorithm will in turn deal with $(j-1)$-containers. **Rule 2** and **Rule 3** are speed-up rules for the cases in which all unassigned containers are to be loaded into the only remaining non-full column. **Rule 4** assigns N-containers into empty columns and the columns that contain only N-containers as much as possible (N-containers at the bottom of columns will never be unloaded before port N, thus it will be more efficient to do so). j-containers with $j < N$ will be loaded first into j-top in order columns based on **Rule 5**, then into empty columns based on **Rule 6**, then into l-top in order columns with $l > j$ based on **Rule 7**, and then into l-min out of order columns with $l \geq j$ based on **Rule 8**. All these loading movements do not generate additional blockings. However if there are still j-containers with $j \leq N$ to be loaded, they have to be loaded into columns containing l-containers with $l < j$, hence more blockings or shifts will occur and the MVSD procedure will be applied in **Rule 9** to determine which columns to be filled and how they are filled. As ij-containers have no impact on the performance when $j = i + 1$, **Rule 10** indicates that they can be loaded on board by any feasible way.

The main algorithm is directly applicable for stowage planning problems with non-rectangular bay, multiple bays, and initial containers (that is, containers loaded on board before visiting the first port).

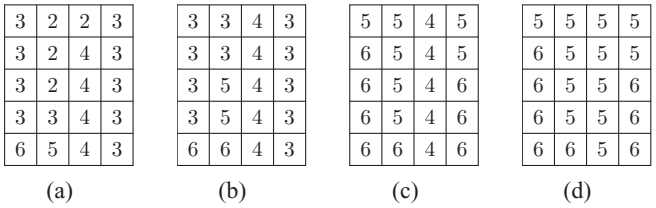


Fig. 1. Algorithm generated stowage plans for the example problem upon leaving (a) port 1; (b) port 2; (c) port 3; and (d) port 4. Note that a square labeled with j stands for a slot occupied by a j-container.

Example. Assume $N = 6$, $R = 5$, $C = 4$, and

$$\mathbf{T} = \begin{bmatrix} 0 & 4 & 10 & 4 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 5 & 6 \\ 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 12 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The algorithm-generated stowage plans upon leaving each port are shown in Fig. 1. For this algorithm generated stowage plan, the number of shifts is 4 and the shifting ratio is 3.85 percent. Please refer to the online supplement (Ding & Chou, 2015) for more details.

4. Performance analysis

We analyze and compare the performance of our main algorithm and HS algorithm with large scale computational experimentations. Both algorithms are implemented with C++ and compiled with Microsoft Visual Studio 2010. The computational experimentations are carried out on PCs with Intel Core 2 Duo CPU and 4GB memories.

In order to evaluate both algorithms, we have to generate transportation matrices automatically. Obviously, the generation method of the matrices has great impact on the performance of the algorithms.

4.1. Transportation matrices

A stowage planning problem with parameters N, R, C and $\mathbf{T}$ is called a full loading problem if the bay is fully occupied upon leaving each port, or technically,

$$\sum_{k=1}^{i} \sum_{j=i+1}^{N} T_{kj} = P, \quad \forall\, i \in \{1, \ldots, N-1\}$$

recalling that P is the volume capacity of the container ship. Correspondingly, $\mathbf{T}$ is called a full loading transportation matrix for capacity P. Notice that the numbers of $i(i+1)$-containers, $i \in \{1,\ldots,N-1\}$, have no impact on the performance of rational stowage plans as $i(i+1)$-containers should never be stowed beneath those containers that have further destinations. We can always transform a non-full loading $\mathbf{T}$ for capacity P into a full-loading $\mathbf{T}'$ for the same capacity without changing the performance of rational stowage plans where

$$T'_{ij} = \begin{cases} T_{ij}, & \text{if } j \neq i+1 \\ T_{ij} + P - \sum_{k=1}^{i}\sum_{j=i+1}^{N} T_{kj}, & \text{if } j = i+1 \end{cases}$$

We say that $\mathbf{T}$ has rank n if the containers to be shipped at most visit n ports (including their destined ports but excluding their original ports), that is,

$$n = \max_{i,j}\left\{ j - i \,\middle|\, T_{ij} > 0 \right\} \leq N-1.$$

We say that $\mathbf{T}$ has norm n if the containers from same original ports have at most n different destinations, or

$$n = \max_{i}\left\{ \sum_{j=i+1}^{N} \operatorname{sgn}(T_{ij}) \right\} \leq N-1$$

where

$$\operatorname{sgn}(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x = 0 \\ -1, & \text{if } x < 0 \end{cases}$$

is the sign function. Obviously, for any $\mathbf{T}$, the rank is no less than the norm.

As a preparation, we consider a random integer partition problem in which a non-negative integer v is randomly partitioned into b non-negative integers $x_1,\ldots,x_b$. This problem can be done efficiently by the following Random Integer Partition (RIP) algorithm, which derived from the famous Fisher–Yates shuffle algorithms (see Fisher & Yates 1948, Durstenfeld 1964; Knuth 1998). And please refer to the online supplement (Ding & Chou, 2015) for a demonstration for the RIP algorithm.

Random Integer Partition Algorithm.

Input. v, b;

Step 1. If $b = 1$, then $x_1 = v$ and go to **Output**;

Step 2. Let $y_n = n$ for $n = 1,\ldots,v+b-1$. Let $i = 1$;

Step 3. Pick a uniformly distributed random integer j between i and $v+b-1$ inclusive. If $j \neq i$, swap the values of y_i and y_j. Let $i = i+1$;

Step 4. If $i < b$, go to **Step 3**;

Step 5. Sort $y_1,\ldots,y_{b-1}$ in ascending order as $y'_1,\ldots,y'_{b-1}$. Let $x_1 = y'_1 - 1$, $x_i = y'_i - y'_{i-1} - 1$ $(i \in \{2,\ldots,b-1\})$ and $x_b = v + b - 1 - y'_{b-1}$. It is easy to verify that $\sum_{i=1}^{b} x_i = v$;

Output. x_i, $i \in \{1,\ldots,b\}$.

We generate the **authentic matrices** randomly for full loading stowage planning problems with parameters N (the number of ports) and P (the volume capacity) by the following Authentic Matrices Generation (AMG) Algorithm. Please refer to the online supplement (Ding & Chou, 2015) for demonstration.

Authentic Matrices Generation Algorithm.

Input. P, N;

Step 1. Initiate $\mathbf{T}$ to be a $N \times N$ matrix of 0;

Step 2. Let $i = 1$;

Step 3. Let $v = P - \sum_{k=1}^{i-1}\sum_{j=i+1}^{N} T_{kj}$ (the number of additional containers required to fill up the bay) and $b = N - i$ (the number of ports where those containers can be destined);

Step 4. Apply the RIP algorithm to obtain b non-negative integers $x_1,\ldots,x_b$ such that $\sum_{j=1}^{b} x_j = v$. Let $T_{i(i+j)} = x_j$ for $j = 1$ to b;

Step 5. If $i < N-1$, let $i = i+1$ and go to **Step 3**;

Output. $\mathbf{T}$.

Avriel et al. (1998) considered three types of auto-generated matrices: **mixed matrices**, **long distance matrices** and **short distance matrices**. Unlike the authentic matrices, stowage planning problems with these three types of matrices generally are not full loading problems. Besides the feasible constraints (1), mixed matrices and long distance matrices are generated by assigning random integers to unassigned elements in random sequence with the constraints $T_{ij} \leq 0.2RC$ and $T_{ij} \leq ((j-i)/(N-1))^2 RC$ respectively, while short distance matrices are generated by assigning random integers to the unassigned elements in lexicographic sequence.

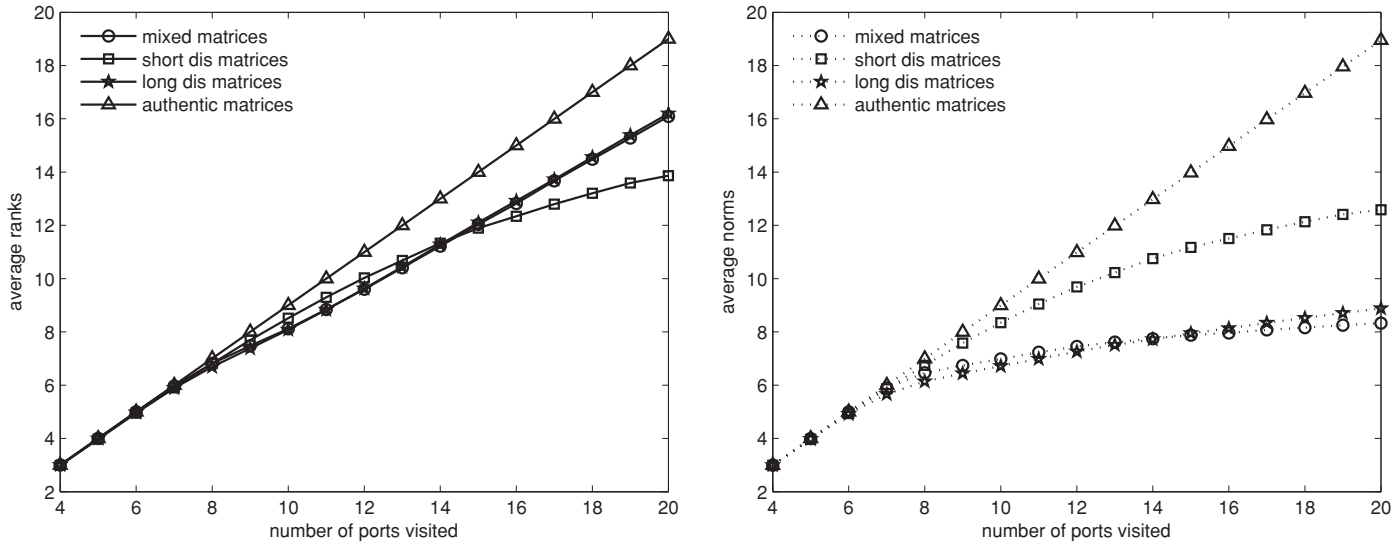
Fig. 2 compares the average ranks and norms of these four types of matrices of 10,000 auto-generated samples, each with $X = 5000$. Authentic matrices have the largest average numbers of ranks and norms, which are both close to the upper bounds. Hence, most elements in the upper triangular of the matrices are non-zeros. As the elements of the same rows are generated at the same time, they have equal possibilities to be large or to be small. Problems with authentic matrices generally lead to the most number of shifts among the four types of matrices which will be shown later.

The respective average ranks of mixed matrices and long distance matrices are close to each other while the latter has relatively large norms due to its smaller upper bound constraints. As the elements in the matrices are generated sequentially, there are high possibilities for those generated first to be large and for those generated later to be zero. As a result, problems with long distance matrices lead to relatively more shifts than those with mixed matrices.

Short distance matrices have smaller ranks and larger norms than the mixed and long distance matrices. Also short distance matrices have large diagonal elements because the elements close to diagonal are generated before the others in each row. As mentioned previously, $T_{i(i+1)}$ have no impacts on the performance of common stowage plans, hence large $T_{i(i+1)}$ would make the problems with short distance matrices much easier than those with the other three types of matrices.

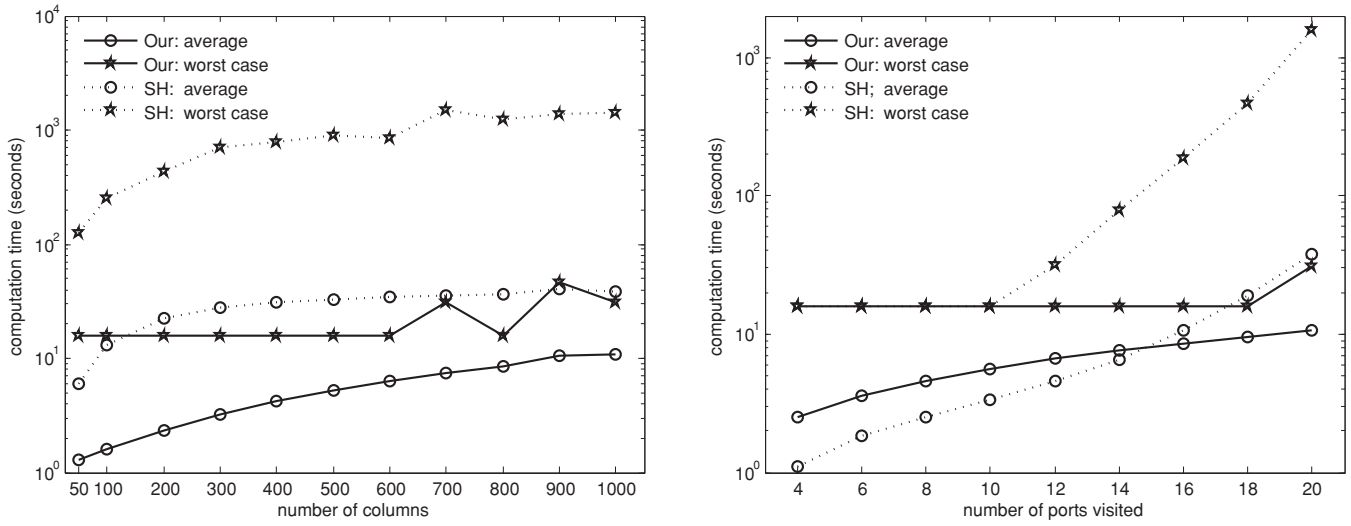
4.2. Performance of the algorithms

Performance of our algorithm and HS algorithm are compared based on large scale computational experimentations, in which R (the number of rows) varies from 6 to 10 and C (the number of columns) varies from 50 to 1000, therefore, $P = R \cdot C$ (the volume capacity of the container ship) varies from 300 to 10,000 TEUs. Moreover, N (the number of ports visited) varies from 4 to 20 and all the four types of matrices are tested. For each setting, 10,000 times of experimentations are carried out, the average number of shifts, the average shifting ratios as well as the computation times are recorded.



(a) average ranks (b) average norms

Fig. 2. Comparison for average ranks and norms of the four types of matrices. Note that the norm and rank of authentic matrices are very close to each other, and the rank of mixed matrices and long distance matrices are close to each other.



(a) the average and the worst case computation time when $N = 20$ and $R = 10$

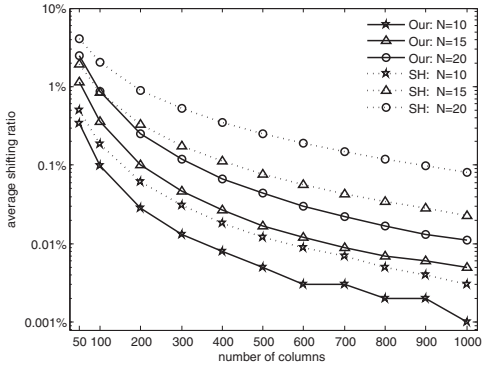
(b) the average and the worst case computation time when $C = 1000$ and $R = 10$

Fig. 3. Comparison of the average and the worst case computation time of the two algorithms when dealing with authentic matrices.

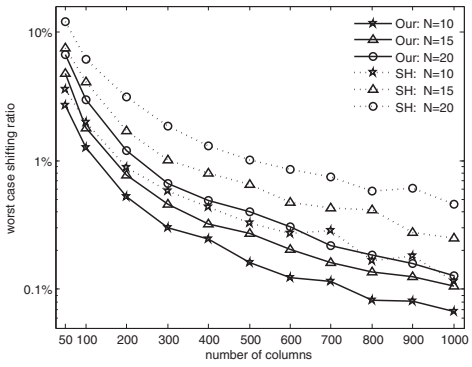
Fig. 3 shows the average and the worst case computation time for both algorithms when dealing with authentic matrices. Note that the time axes are shown in logarithmic scales. Apparently, our algorithm is quite time-efficient, which can generate stowage plans for container ships within a minute. The worst case computation times for our algorithm are shorter than SH algorithm in general, while the average computation times for our algorithm are shorter than SH algorithm when dealing with large size problems. We have very similar results for other three types of matrices and therefore are omitted here.

Fig. 4 shows the average and worst case shifting ratios for stowage plans generated by both algorithms. Note that the shifting ratio axes are shown in logarithmic scales. Obviously, our algorithm outperforms the SH algorithm in general as we have smaller average and worst case shifting ratios in most cases. Indeed, our algorithm generates better, equal and worse solutions than the SH algorithm in 52.59 percent, 41.12 percent and 6.29 percent of all our computational experimentations, respectively. With such facts, we can strategically apply the two algorithms together for same problems to obtain better solutions, see Fig. 5. The probability of achieving a better stowage plan by implementing both algorithms rather than only ours is around 6.29 percent.

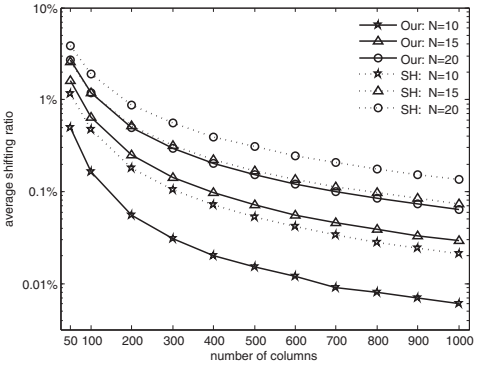
Our algorithm generates stowage plans for each port based only on transportation information of that port, while SH algorithm generates stowage plans also based on additional transportation information of further ports. In practice, it is often impossible to obtain the whole transportation matrix at the beginning of the voyage. Our algorithm has no difficulty in dealing with such situations at all. Moreover, our



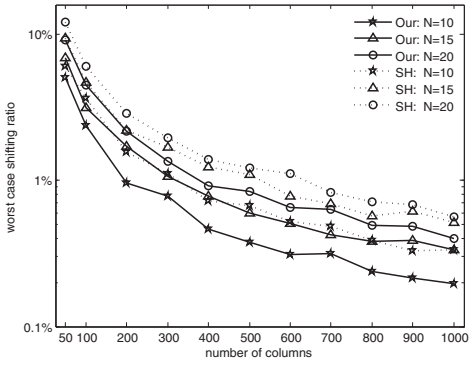
(a) dealing with authentic matrices



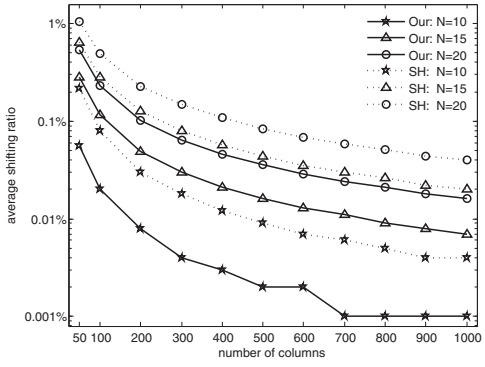
(b) dealing with authentic matrices



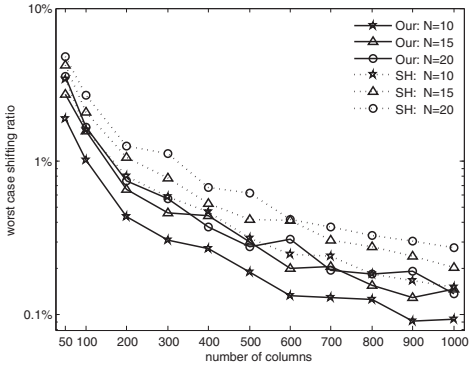
(c) dealing with long distance matrices



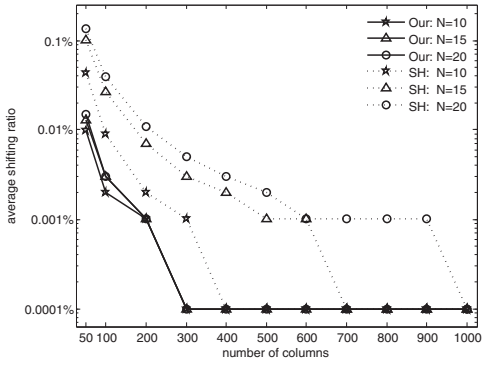
(d) dealing with long distance matrices



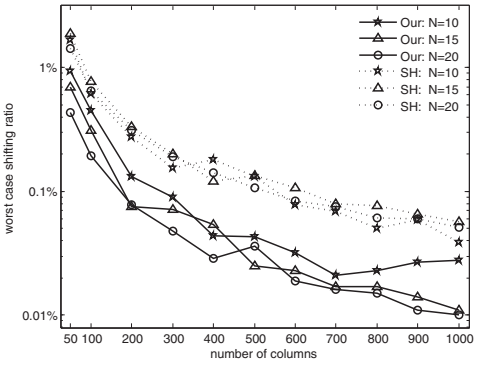
(e) dealing with mixed matrices



(f) dealing with mixed matrices



(g) dealing with short distance matrices



(h) dealing with short distance matrices

Fig. 4. Comparison of the average and worst case shifting ratios of stowage plans generated by both algorithms. Figures on left side: average shifting ratios. Figures on right side: worst case shifting ratios.

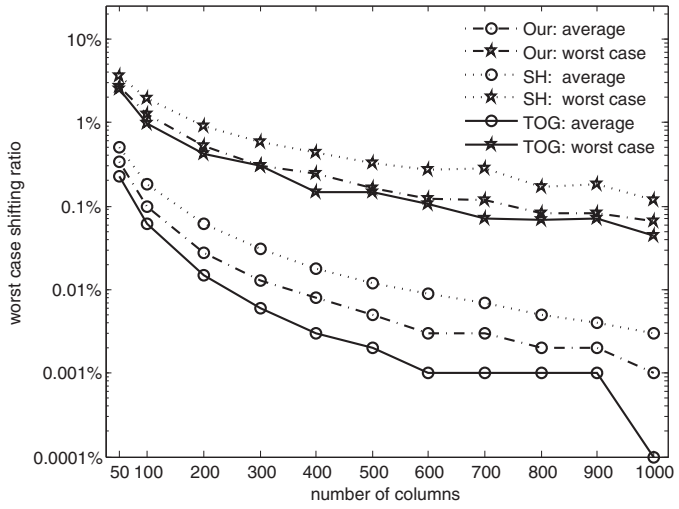


Fig. 5. The two algorithms are applied together to obtain better performance. The problems are with authentic matrices, $N = 10$ and $R = 10$.

algorithm can be easily applied to the cases of round trips, while the SH algorithm seems not.

5. Concluding remarks

In this paper, we consider the stowage planning problem of a container ship. We develop a heuristic algorithm which can generate stowage plans with a reasonable number of shifts for such problems. The algorithm, verified by extensive computational experimentations, performs better than the SH algorithm in Avriel et al. (1998).

Our algorithm can make a stowage plan based on the loading information of current port only, while the SH algorithm requires loading information of all ports, which prevent it from working with partial information. A future work direction is to develop a smart algorithm with certain "look ahead" mechanism, which can make use of all loading information available. For example, if a containership obtains loading information of next three ports when it arrives at some port, it will take all these loading information into account when making the stowage plan for this port, rather than only consider the loading information of this port as in our algorithm. Theoretically it can generate better stowage plans than our algorithm.

Our procedures may be extended to deal with more complex stowage planning problems, such as those with additional weight constraints (e.g. a heavier container cannot be stacked on a lighter one), mixture constraints (e.g. different sizes of containers are stacked together), balance constraints and so on. Indeed, we have extended our main algorithm to deal with stowage planning problems with two additional balance constraints: the cross balance, which requires the absolute difference between the weight on the right side of the ship and the weight on the left side of the ship to be less than a given tolerance, and the horizontal balance, which requires the absolute difference between the weight on the stern and the weight on the bow to be less than a given tolerance. In particular, we have developed procedures that can generate stowage plans which have a high possibility to satisfy the balance conditions and requires reasonable number of shifts. Due to page limit, we omit the details here. Interested readers are welcome to contact the authors for details.

Acknowledgments

The authors thank the editor and the anonymous reviewers for their insightful comments and suggestions, which have helped greatly improve the exposition of this paper. This research was partially supported by National Natural Science Foundation of China (71102082).

Supplementary materials

Supplementary data associated with this article can be found, in the online version, at 10.1016/j.ejor.2015.03.044.

References

Ambrosino, D., Anghinolfi, D., PaNolucci, M., & Sciomachena, A. (2009). A new three-step heuristic for the master bay plan problem. *Maritime Economics & Logistics, 11*, 98–120.
Ambrosino, D., Sciomachen, A., & Tanfani, E. (2004). Stowing a containership: The master bay plan problem. *Transportation Research Part A, 38*, 81–99.
Ambrosino, D., Sciomachen, A., & Tanfani, E. (2006). A decomposition heuristics for the container ship stowage problem. *Journal of Heuristics, 12*, 211–233.
Aslidis, A. H. (1989). *Combinatorial algorithms for stacking problems* (Ph.D. dissertation). MIT.
Avriel, M., & Penn, M. (1993). Exact and approximate solutions of the container ship stowage problem. *Computers and Industrial Engineering, 25*, 271–274.
Avriel, M., Penn, M., & Shpirer, N. (2000). Container ship stowage problem: complexity and connection to the coloring of circle graphs. *Discrete Applied Mathematics, 103*, 271–279.
Avriel, M., Penn, M., Shpirer, N., & Witteboon, S. (1998). Stowage planning for container ships to reduce the number of shifts. *Annals of Operations Research, 76*, 55–71.
Botter, R. C., & Brinati, M. A. (1992). Stowage container planning: A model for getting an optimal solution. In C. B. Vieira, P. Martins, & C. Kuo (Eds.), *Computer applications in the automation of shipyard operation and ship design, VII* (pp. 217–229). North-Holland.
Delgado, A., Jensen, R. M., Janstrup, K., Rose, T. H., & Andersen, K. H. (2012). A constraint programming model for fast optimal stowage of container vessel bays. *European Journal of Operational Research, 220*, 251–261.
Ding, D., & Chou, M. (2015). Online supplement to: Stowage planning for container ships: A heuristic algorithm to reduce the number of shifts.
Dubrovsky, O., Levitin, G., & Penn, M. (2002). A genetic algorithm with a compact solution encoding for the container ship stowage problem. *Journal of Heuristics, 8*, 585–599.
Durstenfeld, R. (1964). Algorithm 235: Random permutation. *Communications of the ACM, 7*, 420.
Fisher, R. A., & Yates, F. (1948). *Statistical tables for biological, agricultural and medical research* (3rd ed., pp. 26–27). Oliver & Boyd.
GDV (2008). Container handbook. http://www.containerhandbuch.de/
Kang, J. G., & Kim, Y. D. (2002). Stowage planning in maritime container transportation. *Journal of the Operational Research Society, 53*, 415–426.
Knuth, D. E. (1998). *The art of computer programming volume 2: Seminumerical algorithms* (3rd ed., pp. 145–146). Addison-Wesley.
Sciomachen, A., & Tanfani, E. (2003). The master bay plan problem: A solution method based on its connection to the three-dimensional bin packing problem. *IMA Journal of Management Mathematics, 14*, 251–269.
Sciomachen, A., & Tanfani, E. (2007). A 3D-BPP approach for optimising stowage plans and terminal productivity. *European Journal of Operational Research, 183*, 1433–1446.
UNCTAD (2014). Review of maritime transport.
Wikipedia (2015). http://en.wikipedia.org/wiki/List_of_largest_container_ships, Accessed April 18, 2015.
Wilson, I. D., & Roach, P. A. (1999). Principles of combinatorial optimization applied to container-ship stowage planning. *Journal of Heuristics, 5*, 403–418.
Wilson, I. D., & Roach, P. A. (2000). Container stowage planning: A methodology for generating computerised solutions. *Journal of the Operational Research Society, 51*, 1248–1255.
Wilson, I. D., Roach, P. A., & Ware, J. A. (2001). Container stowage pre-planning: Using search to generate solutions, a case study. *Knowledge-Based Systems, 14*, 137–145.
Yang, J. H., & Kim, K. H. (2006). A grouped storage method for minimizing relocations in block stacking systems. *Journal of Intelligent Manufacturing, 17*, 453–463.
Zhang, W., Lin, Y., & Ji, Z. (2005). Model and algorithm for container ship stowage planning based on bin-packing problem. *Journal of Marine Science and Application, 4*, 30–36.
Zhang, W., Lin, Y., Ji, Z., & Zhang, G. (2008). Review of containership stowage plans for full routes. *Journal of Marine Science and Application, 7*, 278–285.