# The Stowage Stack Minimization Problem with K-Rehandle Constraint

Ning Wang[a], Shanhui Ke[a], Zizhen Zhang[*,b]

[a] *Department of Information Management, School of Management, Shanghai University, Shanghai, China*
[b] *School of Data and Computer Science, Sun Yat-Sen University, China*

## Abstract

The Stowage Stack Minimization Problem (SSMP) investigates a stowage planning problem when carriers have the obligation to ship all the given containers in different ports, with the objective to utilize the fewest number of stacks on the ship. This problem has realistic usefulness since it can help shipping forwarders estimate shipping space of each shippers. To our best knowledge, we propose SSMP for the first time in the academic community.

In this paper, we talk about SSMP with $K$-rehandle constraint. A heuristic algorithm is put forward to construct solutions. We discuss the theoretical performance guarantee of the algorithm using mathematical inequations and induction. To evaluate the actual performance of our algorithm, we conduct experiments on a set of instances with practical size and compare the results when different values of $K$ are selected. What's more, instances with different values of ports, containers and heights have been tested to make sure the universality of our algorithm.

*Key words:* Containership stowage planning, stack minimization, K-rehandles, constructive heuristic

## 1. Introduction

International trade plays an important role in promoting the development of world economy. Maritime transport is the backbone of globalization and lies at the heart of cross-border transport networks that support supply chains and enable international trade. Nowadays, over 80% of world trade is carried by the maritime freight industry, which operates the container transportation business (Zhang & Lee, 2016). In 2015, world seaborne trade volumes surpassed 10 billion tons shown in UNCTAD (2016).

Container-shipping is a kind of modern transportation tool which has a lot of advantages in speed, security, and quality comparing with the break-bulk cargo ship. Containerization is an important driver of the global economy, and the container has become a mainstay of worldwide trade. According to World Shipping Council, around 52% of the value of world international seaborne trade today is being moved in containers. This mode is deeply accepted by shippers and carriers, and is becoming the main trend of freightage in the

*Corresponding author. Fax: +852 ********, Phone: +852 ********
*Email addresses:* ningwang@shu.edu.cn (Ning Wang), zhangzizhen@gmail.com (Zizhen Zhang)

world. It has resulted in a tremendous growth of the maritime freight industry and it shows powerful vitality and promising prospects.

With the development of container-shipping, more and more containers are transported by sail. Over the past decades there has been a continuous increase in demand for cost efficient containerized transportation. Thus, it makes higher research demand. The container stowage plan is a pivotal link to containers transportation, and the objective of which is to draw a plan of loading and discharging sequence of containers for containership in Zhang et al. (2008). The most frequently used addressing notation for storage locations in a container ship is the bay-row-tier system. The cargo space of a container ship is split into several 20-ft-long areas. These areas are termed as 20-ft bays. Figure 1 reproduced from Delgado et al. (2012) shows a diagram of containership.

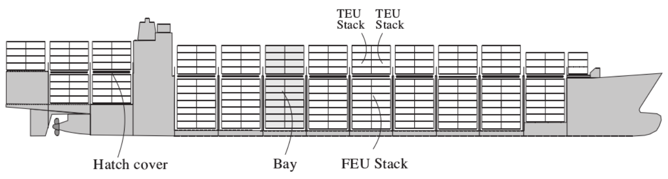

Figure 1: The arrangement of cargo space in a container vessel.

Each bay consists of container stacks placed along the width of ship. Stacks in a bay are indexed by a row number. The position for one 20-ft container in the ship is uniquely indexed by (bay, row, tier), called a slot. In addition, for some ships, two contiguous 20-ft bays form a 40-ft bay which can be used to accommodate 40-ft containers.

A ship usually calls a sequence of ports and containers are loaded and unloaded at each port by cranes in a last-in-last-out (LIFO) manner. Rehandles 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 rehandles in the future. Rehandles are expensive, since on one hand terminals directly charge the carriers for such operations, on the other hand, rehandles make ships dwell at terminals longer, inducing more berthing cost. Actual examples show that containership stowage plans not only influence the income of shipping company from transportation but also have direct relation to the safety of ships and freights. The problems have troubled and aroused the interests of both scholars and commercial shipping organization in many countries since 1970s (Webster & Van Dyke, 1970). Most of previous research on stowage planning focus on how to minimize the number of rehandles (Ding & Chou, 2015; Zhang & Lee, 2016; Zehendner et al., 2017).

In this paper, we investigate the problem from a new perspective. We deal with Stowage Stack Minimization Problem (SSMP) which minimizes the number of used stacks. The problem has realistic meaning.

In the real world, a carrier/forwarder may collect a few containers in a time interval and want to evaluate needed space (stacks), in order to estimate available space left. Or on the contrary, a carrier/forwarder wants to receive provisional shipment orders as many as possible while reserve enough space for regular shipment orders. Our algorithm provides them an indicator to achieve such goals.

The contribution of our paper lies in three aspects. First, to the best of our knowledge, our paper considers number of stacks used for the first time, and this is a very practical consideration. As talked before, previous research usually focus on how to reduce the rehandles or shifts. Second, we analyze SSMP and give upper and lower bounds of SSMP. The bounds pave a way for future study. Third, we propose a heuristic algorithm to solve SSMP and the algorithm has a performance guarantee.

The remainder of this paper is organized as follows: we review extant research works in Section 2. The formal definition of SSMP and its properties are provided in Section 3. In Section 4, we present our heuristic algorithm and the performance analysis. Experiments are illustrated in Section 5. Concluding remarks close the paper in Section 6.

## 2. Literature review

The container handling problems can be divided into two streams based on the container handling locations occurred; one is container management at terminals, and the other is containership stowage planning problems (CSP). The problem discussed in this paper belongs to the latter. Research methods of CSP can be divided into five categories: simulation methods based on probability, expert systems, decision support systems, mathematical programming approach and heuristic algorithm. Research about the last two methods are more widely used now and we focus on them in the following review of relevant works.

Webster & Van Dyke (1970) first tried to solve the problem by using simulation methods with the help of computer. They put forward a heuristic algorithm that could create a random stowage plan. Shields (1984) introduced a computer preplanning system (CAPS) which had been used by American President Lines. The CAPS produced stowage plans by using Monte Carlo theory.

Dillingham & Perakis (1986) introduced their researches on expert system of stowage based on rules. Each of the suggested container movements was displayed graphically as the decisions were made.

Wilson et al. (2001) outlined a computer system that generated good sub-optimal solutions to the s-towage pre-planning problem. The methodology progressively refined the arrangement of containers within the cargo-space of a container ship until each container is specifically allocated to a stowage location.

Botter & Brinati (1991) created a mathematical model for the stowage plan problem, which used binary decision variables to determine the containers unloading and loading sequence for each port. A 0-1 binary linear programming formulation was presented in order to minimize the number of shifts in Avriel et al.

(1998) and Avriel et al. (2000). Haghani & Kaisar (2001) proposed a MIP model for developing loading plans in order to minimize the time that a vessel spent at port, and the container handling cost which shifts caused by was highly influenced by the number of unproductive but necessary an unsatisfactory arrangement of containers. In Ambrosino et al. (2004), the problem of stowing containers into a containership had been faced by evaluating an exact 0-1 Linear Programming model, which was not practically useful for large cases. Ambrosino et al. (2015) proposed a Mixed Integer Programming (MIP) heuristic aimed at determining stowage plans in circular routes for container ships so as to give support for the ship coordinator and the terminal planner. In Parreno et al. (2016), they focused on the slot planning phase and presented a Constraint Programming and Integer Programming model for stowing a set of containers in a single bay section.

Three heuristic rules were used to assign gradually each type of containers to slots of the ship by four steps in Scott & Chen (1978), while the shifting problem had not been considered in their model. Tabu search was employed to obtain an optimal solution in Wilson et al. (2001), Bortfeldt et al. (2003) and Monaco et al. (2014). Ding & Chou (2015) developed a heuristic algorithm which can generate stowage plans with a reasonable number of shifts for such problems. The algorithm, verified by extensive computational experimentations, performed better than the Suspensory Heuristic Procedure (SH algorithm) in Avriel et al. (1998), which was a dynamic slot-assignment scheme. The presented hybrid GA for loading a single container in Bortfeldt & Gehring (2001) was particularly suitable for strongly heterogeneous containers stowage problems. Dubrovsky et al. (2002) used a GA for solving the stowage planning problem of minimizing the number of container movements. Search space was significantly reduced by a compact and efficient encoding scheme. Kammarti et al. (2009) proposed an efficient genetic algorithm which consists on selecting two chromosomes (parent) from an initially constructed population using a roulette wheel technique. The variant tackled in Cohen et al. (2017) involved several constraints, inspired by real-life problems and application found in the literature.

We have also read literatures about container stowage problem with rehandles constraint. Malucelli et al. (2008) investigated stack reordering strategies aiming at minimizing the number of loading and unloading operations. The complexity of stowage planning problem is investigated in Tierney et al. (2014) and it showed that the capacitated k-shift problem is solvable in polynomial time for any choice of stacks and stacks capacities. A multi-objective ship stowage planning problem was researched by Zhang & Lee (2016) and it aimed to optimize the ship stability and the number of rehandles simultaneously.

To the best of our knowledge, only Avriel et al. (2000) and Jensen (2010) discussed the stowage stack minimization and its connection with the chromatic number of circle graphs when the stack height is unlimited apart from the SSMP-ZR researched by Wang et al. (2014).

## 3. Problem description and properties

### *3.1. Problem description and notation*

We put forward SSMP in this subsection. Literally speaking, SSMP is a variant of CSP. A ship starts its journey at port 1 and sequentially visits port 2, 3, ..., $P$. We assume the ship stops at port $P$. $N$ containers are shipped along the journey. At each port, the containers destined to the current port are discharged, and the containers await on yard are loaded to the ship. We use $O(c)$ and $D(c)$ to denote the original and destination ports of container $c$, respectively. Accordingly, the itinerary is represented by a tuple $\{O(c), D(c)\}$. Given $N$ containers to be shipped, the objective of SSMP is to figure out a stowage planning so as to minimize needed stacks.

If two containers $c1$ and $c2$ satisfy $O(c1) < O(c2) < D(c1) < D(c2)$, and $c2$ is placed above $c1$, then one *rehandle* occurs, as we have to move $c2$ to the yard temporally for retrieving $c1$. Besides, containers above $c2$ also have to be moved, and therefore also extra rehandles occur. In this paper, we extend SSMP with zero rehandle constraint in Wang et al. (2014), assuming that $K$ rehandles are allowed.

Other assumptions assumed in this paper:

- Containers loaded are uniform standard 20-ft containers;
- The information of $N$ containers are known in advance;
- The containership is large enough to accommodate all the given containers;
- Each stack can hold at most $H$ vertically piled containers, i.e., the height limit of every stack is $H$;
- The balance of the containership can be resolved by assigning container stacks and water-ballast; hence, the balance is not considered when minimizing container stacks.

If the number of containers in a non-empty stack is less than $H$, the stack is termed as a *partial* stack; otherwise it is a *full* stack. If rehandle occurs, containers above are called *blocking containers* and the to-be-retrieved container is called *target container*. When moving blocking containers back to the ship, there is no guarantee that they will be returned to their original slots.

A feasible solution to the SSMP is represented by a sequence of operations. Each operation indicates the number of container and the accommodating stack. Especially, the yard is term as stack 0. Moving a container $c$ to stack 0 means that $c$ is a blocking container and moved temporally on the yard.

## 3.2. Integer model

For the special case with zero rehandle, we only need to consider the loading process. For the unloading process, containers can be discharged in order without rehandles. The following IP model decides the accommodating stack of each container.

$$(IP) \quad \min \sum_{s=1}^{S} y_s$$

$$\text{s.t.} \quad \sum_{s=1}^{S} x_{is} = 1, \ \forall 1 \leq i \leq N \tag{1}$$

$$\sum_{i=1}^{N} x_{is} \leq M y_s, \ \forall 1 \leq s \leq S \tag{2}$$

$$\sum_{i:O(i) \leq p \leq D(i)-1} x_{is} \leq H, \ \forall 1 \leq s \leq S, 1 \leq p \leq P \tag{3}$$

$$x_{is} + x_{js} \leq 1, \ \forall 1 \leq s \leq S, 1 \leq i, j \leq N, O(j) < O(i) < D(j) < D(i) \tag{4}$$

$$x_{is}, y_s \in \{0, 1\}, \ \forall 1 \leq s \leq S, 1 \leq i, j \leq N \tag{5}$$

In the above model, $M$ is a sufficiently large number and $S$ is the number of stacks on the ship. $x_{is}$ is a binary decision variable which is equal to 1 if container $i$ is loaded to stack $s$, and 0 otherwise. $y_s$ is a binary decision variable that is equal to 1 if stack $s$ is used for stowage, and 0 otherwise. $O(i)$ and $D(i)$ indicate the shipping leg of container $i$. The objective minimizes the number of stacks used. Constraints (1) ensure that each container $i$ is loaded to only one stack. Constraints (2) guarantee that containers can only be loaded to used stacks. Constraints (3) assure that at each port $p$, the number of containers in stack $s$ does not exceed $H$. Constraints (4) avoid rehandle requiring that container $i$ is prohibited to block container $j$ in the same stack when $O(j) < O(i) < D(j) < D(i)$.

The IP solution can be converted into a complete loading plan: start from port $p = 1$ to port $p = P$. At port $p$, containers are sorted in a descending order of destination ports. Each container $i$ is loaded to the top of its corresponding stack $s^*$ which satisfies $x_{is^*} = 1$.

For the IP model of general SSMP, it is hard to build a model. Since there are rehandles, we have to move blocking containers on the yard and back to the ship. The stacks of blocking containers are not necessarily the original slots. If we want to model the slots of blocking containers, we have to set variable $x_{ctl}$ that indicates the slot $t$ of a container $c$ at a layout $l$. We do not know which containers are blocking containers in advance, therefore the domain of $c$ is the whole set of $N$ containers. Especially, whenever we load or discharge a container, it forms a layout. In addition, we do not know the number of layouts in advance. All in all, the number of slots, containers, and layouts are numerous. Apart from other variables and constraints, the number of variable $x_{ctl}$ alone is too large to make the model meaningful and solvable.

## 4. Methodology

In this section, we will talk about the heuristic algorithm to solve the SSMP as well as the performance guarantee of the algorithm.

### *4.1. Heuristic algorithm for the SSMP*

Algorithm 1 solves instances of SSMP. The main two steps in our heuristic algorithm are unloading and loading procedures and the key point is how to choose the optimal stack for each container to make sure we can minimize the needed stacks..

Given an instance I, for each port $p$ along the voyage, containers with $D(c) == p$ are discharged one by one firstly. Especially, if there exist blocking containers, they are moved on the yard temporally and regarded as containers to be loaded at current port. After containers with $D(c) == p$ are discharged, containers on yard (including containers with $O(c) == p$ and blocking containers) are sorted in order and loaded.

When loading containers at each port, two different methods are adopted to choose stack for each target container according to whether $k$ is equal to $K$ or not. They are $loading\_equalK(c, nearport)$ and $loading\_lessK(c, nea$ respectively. $k$ means the number of real rehandles and it ranges from zero to $K$. $K$ means the number of allowed rehandles given in our problem. *nearport* is used to store the nearest port in each stack. For example, three containers are stored in the same stack, and their destination ports are 4,5,6, respectively, then the value of *nearport* of this stack is 4.

The value of $NumofStack$ will be updated when all the containers are processed at port $p$. $NumofStack$ here are referred as the largest number of used stacks after the containership departs from port $p$. The algorithm finally outputs the value of $NumofStack$ at port $P$ when all containers in an instance are processed.

This paper will give a detailed introduction of the two methods. We firstly talk about the simple condition that the allowed rehandles run out ($k == K$) and the procedure of method $loading\_equalK(c, nearport)$ is given in Algorithm 2. We add non-full stacks whose nearest port is no smaller than the current loading container into *arraylist* $S_0$ when traversing all the stacks to choose the feasible stacks for each container. For all the stacks in $S_0$, we choose the one with the smallest value of *nearport*. If there isn't only one stack with the smallest value of *nearport*, we choose the first one we traverse.

- $S_0$=currently non-full stacks whose *nearport* is greater than or equal to $D(c)$.
- **if** $S_0 \neq \emptyset$, then $s_{min} = \arg\min_{s \in S_0} nearport[s]$.
- Clear the *arraylist* $S_0$
- Load container $c$ to stack $s_{min}$

**Algorithm 1** A heuristic procedure for the SSMP

```
HEURISTIC(I)
1  for each port $p = 1, 2, \ldots, P$
2      Unload containers with $D(c) == p$.
3      Sort containers with $O(c) == p$ by the decreasing order of their destinations.
4      for each container $c$ with $O(c) == p$ at port $p$
5          if $k == K$
6              execute method $loading\_equalK(c, nearport)$.
7          else If $k < K$
8              execute method $loading\_lessK(c, nearport)$.
9      Update the value of $NumofStack$ at port $p$.
10 Output the value of $NumofStack$ in instance I.
```

**Algorithm 2** The procedure for the method $loading\_equalK(c, nearport)$

```
$loading\_equalK(c, nearport)$
1  $S_0$=currently non-full stacks whose $nearport$ is greater than or equal to $D(c)$.
2  if $S_0 \neq \emptyset$
3      $s_{min} = \arg\min_{s \in S_0} nearport[s]$.
4      Clear the $arraylist$ $S_0$.
5      Load container $c$ to stack $s_{min}$.
6  Update the value of $nearport$ of stack $s_{min}$.
7  Update the height of stack $s_{min}$.
```

- Update the value of *nearport* of stack $s_{min}$.
- Update the height of stack $s_{min}$.

step 1 $S_0$=currently non-full stacks whose *nearport* is greater than or equal to $D(c)$.

step 2 If $S_0 \neq \emptyset$, then $s_{min} = \arg\min_{s \in S_0} nearport[s]$.

step 3 Clear the *arraylist* $S_0$.

step 4 Load container $c$ to stack $s_{min}$.

step 5 Update the value of *nearport* of stack $s_{min}$.

step 6 Update the height of stack $s_{min}$.

Afterwards, we talk about the complex condition that the rehandles are allowed ($k < K$) and the procedure of method $loading\_lessK(c, nearport)$ is given in Algorithm 3. Undoubtedly, its procedure to load containers is not as easy as the simple condition. Under this condition, we use *arraylist* $S_1$ to store partial stacks without rehandles occurring, *arraylist* $S_2$ to store partial stacks with rehandles occurring and *arraylist* $S_3$ to store empty stacks. The priority order of them is $S_1$, $S_2$, $S_3$, which represents the chosen sequence of different types of stacks. Like the above Algorithm 2, we choose the first one with the smallest value of *nearport* when we traverse each set of feasible stacks. It's worth mentioning that $s_{min}$ in Algorithm 2 and 3 represents the first stack with the smallest value of *nearport* according to the traversing sequence.

- $S_1$=currently partial stacks whose *nearport* is greater than or equal to $D(c)$.
- $S_2$=currently partial stacks whose *nearport* is smaller than $D(c)$.
- $S_3$=currently empty stacks.
- If $S_1 \neq \emptyset$
- Then $s_{min} = \arg\min_{s \in S_1} nearport[s]$
- Clear the *arraylist* $S_1$
- Load container $c$ to stack $s_{min}$.
- Else If $S_2 \neq \emptyset$
- Then $s_{min} = \arg\min_{s \in S_2} nearport[s]$

**Algorithm 3** The procedure for the method *loading_lessK*$(c, nearport)$

```
loading_lessK(c, nearport)
1   $S_1$=currently partial stacks whose nearport is greater than or equal to $D(c)$.
2   $S_2$=currently partial stacks whose nearport is smaller than $D(c)$.
3   $S_3$=currently empty stacks.
4   if $S_1 \neq \emptyset$
5       $s_{min} = \arg\min_{s \in S_1} nearport[s]$.
6       Clear the arraylist $S_1$.
7       Load container $c$ to stack $s_{min}$.
8   else If $S_2 \neq \emptyset$
9           $s_{min} = \arg\min_{s \in S_2} nearport[s]$.
10          Clear the arraylist $S_2$.
11          Update the value of $k$.
12          Load container $c$ to stack $s_{min}$.
13      else
14          Clear the arraylist $S_3$.
15          Load container $c$ to the first empty stack in $S_3$, which is also indexed by $s_{min}$.
16  Update the value of nearport of stack $s_{min}$.
17  Update the height of stack $s_{min}$.
```

```
    - Clear the *arraylist* $S_2$
    - Update the value of $k$
    - Load container $c$ to stack $s_{min}$.
    - Else Clear the *arraylist* $S_3$
    - Load container $c$ to the first empty stack in $S_3$, which is also indexed by $s_{min}$.
    - Update the value of *nearport* of stack $s_{min}$.
    - Update the height of stack $s_{min}$.
```

```
step 1  $S_1$=currently partial stacks whose nearport is greater than or equal to $D(c)$.
step 2  $S_2$=currently partial stacks whose nearport is smaller than $D(c)$.
step 3  $S_3$=currently empty stacks.
step 4  If $S_1 \neq \emptyset$, then $s_{min} = \arg\min_{s \in S_1} nearport[s]$.
step 5  Clear the arraylist $S_2$.
step 6  Update the value of $k$.
step 7  Load container $c$ to stack $s_{min}$.
step 8  Else Clear the arraylist $S_3$.
step 9  Load container $c$ to the first empty stack in $S_3$, which is also indexed by $s_{min}$.
step 10  Update the value of nearport of stack $s_{min}$.
step 11  Update the height of stack $s_{min}$.
```

From Algorithm 2 and 3, we can see that we will clear the *arraylist* once we have chosen the stack for the target container. Of course, we will update the value of $k$ if there is a rehandle when loading containers. After loading the target container into the chosen stack, the value of *nearport* and height of the chosen stack need to be updated.

In a word, a lot of variables and methods of the specific unloading and loading strategies have been involved, we don't give the complete pseudo here for the sake of context length. For example, the method to record the information of loading and unloading operations isn't given in this paper. As is explained in the heading of this section, we divide the loading strategy into two methods on the basis of the relationship between $k$ and $K$. What is identical for the two methods is that they choose the optimal stack for each container according to the greedy rules.

It's worth mentioning that our algorithm can figure out the detailed location changes of each container and output the layout status in the containership at arbitrary time. In other words, we can predict the real-time container stowage process before the voyage as long as we know some information in advance. Therefore, our algorithm makes a little difference and makes some sense in the reality application.

## *4.2. Performance Guarantee of the Algorithms*

In order to simplify the problem, we show that the heuristic algorithms have a performance guarantee and we regard the port number $P$ as a fixed number in our voyage. For ease of exposition, we first give some related notations.

- $C^*$: the optimal solution to the SSMP instances.
- $C$: the solution to the SSMP instances by the algorithm.
- $N_p$: the number of containers on the ship before its departure from port $p$.
- $V_p$: the number of loading ports that the ship has visited before it departs from port $p$.

It should be noted that a port is called a loading port if there exists at least one container to be loaded. Clearly, $V_p \leq p, \forall p = 1, \ldots, P$.

**Proposition 1.** *The SSMP instance has a lower bound:*

$$\max_{p=1,\ldots,P} (\lceil \frac{N_p}{H} \rceil) \leq C^*$$

*Proof.* $\lceil \frac{N_p}{H} \rceil$ is the least number of stacks to accommodate all the containers at port $p$, and thus, $\max_p(\lceil \frac{N_p}{H} \rceil)$ is the least number of stacks needed throughout the journey. $\square$

The lower bound can be a reference to check the optimality of solutions obtained by our heuristic. If the solution is close to the lower bound, we can conclude that the solution is close to optimal.

We divide the stack used in the port into two parts: full stacks and partial stacks. For the full stacks, the number of full stacks in port $p$ is no greater than $\lfloor \frac{N_p}{H} \rfloor$. For the partial stacks, we define the number of partial stacks in port $p$ as $P(p)$, and we have Proposition 2 for $P(p)$.

**Proposition 2.** *The number of partial stacks in port p is no greater than $V_p$:*

$$P(p) \leq V_p$$

*Proof.* We use an inductive proof.

*Basis*: For $p = 1$, without loss of generality, we assume that port 1 is a loading port, and therefore $V(p = 1)$ =1. In addition, P(1)=0 or 1, i.e. $P(1) \leq 1$. Hence, $P(p) \leq V_p$ holds.

*Inductive step*: Suppose that the partial stacks in port $i$ is no greater than $V_i$, i.e. $P(i) \leq V_i$. Upon the ship arrives at port $i + 1$, it first unloads the containers from the containership. If there are blocking containers to be unloaded, they are moved on the yard temporally and regarded as containers to be loaded at current port. It's not difficult to know new partial stack transforming from empty stack won't be produced. Obviously, new partial stack transforming from empty stack won't be produced if there are no blocking containers. That's to say, the unloading process does not increase the number of partial stacks transforming from empty stack won't be produced. We don't consider the partial stack transforming from full stack since it won't change the total used stacks.

(1)If there are containers to be loaded at port $i + 1$, they will be loaded into the previous partial stacks or a empty stack, then $V_{i+1} = V_i + 1$, $P(\mathrm{i+1}) \leq P(i) + 1$.

(2)If there is no containers to be loaded at port $i + 1$, $V_{i+1} = V_p$, $P(\mathrm{i+1}) \leq P(i)$. Hence, $P(\mathrm{i+1}) \leq P(i) \leq V_i = V_{i+1}$.

Based on the above two steps, we have $P(p) \leq V_p, \forall p = 1, \ldots, P$. □

**Proposition 3.** *For the heuristic solution to the SSMP instance, it holds*

$$C^* \leq C \leq \max_{p=1,\ldots,P} (\lfloor \frac{N_p}{H} \rfloor + V_p) \leq C^* + V_P$$

*Proof.* The first inequality obviously holds.

For the second inequality, $C = \max_p C_p$. Noted that $C_p$ is the number of used stacks in port $p$ obtained from our algorithm and it includes the number of full stacks and the number of partial stacks. As the number of full stacks in port $p$ is no greater than $\lfloor \frac{N_p}{H} \rfloor$ and the number of partial stacks in port $p$ is no greater than $V_p$. Therefore, the second inequality holds. For the third inequality, $\lfloor \frac{N_p}{H} \rfloor \leq \lceil \frac{N_p}{H} \rceil$, and $\lceil \frac{N_p}{H} \rceil$ is the lower bound of the number of stacks used at port $p$ by Proposition 1. Thus, $\lfloor \frac{N_p}{H} \rfloor \leq C^*_p$. It then holds $\max_p(\lfloor \frac{N_p}{H} \rfloor + V_p) \leq \max_p(C^* + V_p) = C^* + V_P$. □

The above propositions show that our heuristic algorithm has a constant performance guarantee. In the real shipping transportation, the number of ports is generally much smaller than the number of used stacks. Therefore, the difference between the optimal solution and our heuristic solution is relatively small, which indicates that our heuristic algorithms can generate promising solutions.

## 5. Experiments and Analysis

This paper uses the test data from the existing literature (Wang et al. (2014)) and we add an extra parameters $K$ into the primary instances in order to meet the demands of research. $K$ means the number of allowed rehandles and it is selected from {0, 10, 20, 50, 100}. Particular, instances with $K = 0$ represent the benchmark instances comparing with the same instances with different values of $K$. In fact, the problem becomes SSMP-ZR when $K = 0$. There are 360 sets of instances, or 1800 instances in total, which are categorized by four parameters: the number of allowed rehandles, which is selected from {0, 10, 20, 50, 100}; the number of ports $P$, which is selected from {5, 10, 20, 30}; the number of containers $N$, which is selected from {50, 100, 200, 500, 1000, 5000}; and the height limit $H$ is selected from {4, 8, 12}. Each set consists of 5 instances generated by different random seeds. In each instance, the origin $O(i)$ and the destination $D(i)$ of a container $i$ are generated from a uniform distribution on the integers $1, 2, \ldots, P$ satisfying that $O(i) < D(i)$. For the convenience of comparison, we only show the average value of ”Numofstack” when $K = 0$, $K = 10$, $K = 20$, $K = 50$, $K = 100$, respectively.

In this section, we will showcase the performance of our heuristic algorithms on a number of test instances. The heuristic was implemented in Java and the experiments were conducted on a computer with Intel Core processor clocked at 2.30 GHz and 8 GB RAM. The operating system of the computer is Windows 10.

The experimental results are shown in Table 1 and Table 2. Considering the size of table, the results of instances with 5000 containers are not listed.

Table 1 summarizes the results of those easy instances with small number of ports, i.e. $P \in \{5, 10\}$. For each given $P$, $N$ and $H$, we work out the values of $LB$, $UB$ of our heuristic algorithm. For both our heuristic algorithm and random algorithm, we work out the average numbers of $Numofstack$ in different instances when $K$ selects different values. The $LB$ and $UB$ are used to test the rationality of heuristic algorithm and the average $Numofstack$ of instances with different values of $K$ by using heuristic and random algorithms are used to compare with each other.

Table 2 summarizes that the results of those difficult instances with large number of ports, i.e. $P \in \{20, 30\}$. The columns in this table are similarly defined as those of Table 1 and so is the conclusion we can get from Table 1.

Most of instances provide the evidence that it works better when $K$ isn't equal to zero comparing the

Table 1: Results of the instances with $P \in \{5, 10\}$

| P | N | H | Heu | | | | | | | Ran | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | LB | UB | K=0 | K=10 | K=20 | K=50 | K=100 | K=0 | K=10 | K=20 | K=50 | K=100 |
| 5 | 50 | 4 | 8 | 9.6 | 8 | 8 | 8 | 8 | 8 | 12.2 | 9.6 | 10.4 | 9.6 | 9.6 |
| 5 | 50 | 8 | 4 | 6 | 4.6 | 4 | 4 | 4 | 4 | 10 | 8.4 | 8 | 9.6 | 7.8 |
| 5 | 50 | 12 | 3 | 5 | 3.2 | 3.2 | 3 | 3 | 3 | 9.2 | 9.2 | 7 | 8.6 | 7.6 |
| 5 | 100 | 4 | 15 | 16.8 | 15 | 15 | 15 | 15 | 15 | 18.4 | 16.2 | 16.2 | 16.6 | 16.6 |
| 5 | 100 | 8 | 7.6 | 9.6 | 7.6 | 7.6 | 7.6 | 7.6 | 7.6 | 14.2 | 14.8 | 12.8 | 12 | 13.2 |
| 5 | 100 | 12 | 5.4 | 7.6 | 5.4 | 5.6 | 5.4 | 5.4 | 5.4 | 14.2 | 14.4 | 12.6 | 13.2 | 12 |
| 5 | 200 | 4 | 30.6 | 32.2 | 30.6 | 30.6 | 30.6 | 30.6 | 30.6 | 32.8 | 32.4 | 32.4 | 32 | 33 |
| 5 | 200 | 8 | 15.6 | 17.4 | 15.6 | 15.6 | 15.6 | 15.6 | 15.6 | 23.6 | 24.6 | 22.4 | 19.4 | 20.4 |
| 5 | 200 | 12 | 10.4 | 12.4 | 10.4 | 10.4 | 10.4 | 10.4 | 10.4 | 21.8 | 22.8 | 20.4 | 19.2 | 16.2 |
| 5 | 500 | 4 | 76.6 | 78 | 76.6 | 76.6 | 76.6 | 76.6 | 76.6 | 78.6 | 78.6 | 78 | 77.8 | 79 |
| 5 | 500 | 8 | 38.6 | 40 | 38.6 | 38.6 | 38.6 | 38.6 | 38.6 | 46 | 44.4 | 45 | 42.8 | 42 |
| 5 | 500 | 12 | 26 | 27.4 | 26 | 26 | 26 | 26 | 26 | 38 | 36.2 | 39.8 | 37.6 | 33.8 |
| 5 | 1000 | 4 | 152.2 | 153.8 | 152.2 | 152.2 | 152.2 | 152.2 | 152.2 | 153.6 | 153.6 | 154.2 | 154.2 | 153.4 |
| 5 | 1000 | 8 | 76.6 | 78.2 | 76.6 | 76.6 | 76.6 | 76.6 | 76.6 | 82.2 | 81.8 | 81.4 | 80.2 | 79.6 |
| 5 | 1000 | 12 | 51 | 52.8 | 51 | 51 | 51 | 51 | 51 | 61.4 | 60.8 | 60.4 | 63 | 60.4 |
| 10 | 50 | 4 | 7.8 | 12.6 | 9 | 8 | 7.8 | 7.8 | 7.8 | 12.4 | 11 | 10.2 | 9.8 | 9.6 |
| 10 | 50 | 8 | 4 | 10 | 6 | 6 | 5 | 4 | 4 | 10.8 | 10.8 | 10.4 | 8.8 | 8.8 |
| 10 | 50 | 12 | 3 | 9.4 | 5.6 | 4.6 | 4.4 | 3.2 | 3 | 12 | 11.4 | 9 | 8.8 | 8.6 |
| 10 | 100 | 4 | 14.8 | 19.6 | 15.8 | 15.6 | 15 | 14.8 | 14.8 | 20.4 | 20.2 | 20.2 | 16.8 | 17.4 |
| 10 | 100 | 8 | 7.6 | 12.6 | 9.2 | 9.8 | 9.6 | 7.8 | 7.6 | 20 | 18.4 | 17 | 13.6 | 13.6 |
| 10 | 100 | 12 | 5.2 | 10.4 | 7 | 7 | 7.2 | 6.4 | 5.2 | 18.4 | 17.8 | 17.6 | 13.8 | 14.2 |
| 10 | 200 | 4 | 28.6 | 33.2 | 29.6 | 29.6 | 28.6 | 28.6 | 28.6 | 35.4 | 36 | 34.4 | 31.4 | 30.6 |
| 10 | 200 | 8 | 14.4 | 19.2 | 16.4 | 16.6 | 16.2 | 14.8 | 14.4 | 28.6 | 29.2 | 25.6 | 27.4 | 23 |
| 10 | 200 | 12 | 9.8 | 15 | 12 | 12 | 11.8 | 11.4 | 9.8 | 26.4 | 27.2 | 26.8 | 24.2 | 24 |
| 10 | 500 | 4 | 69.8 | 74.2 | 69.8 | 69.8 | 69.8 | 69.8 | 69.8 | 78 | 75.4 | 74.6 | 74.2 | 71.2 |
| 10 | 500 | 8 | 35 | 39.4 | 35.6 | 35 | 35.2 | 35 | 35 | 52.8 | 49.4 | 52.6 | 51.4 | 52.6 |
| 10 | 500 | 12 | 23.6 | 28 | 25.2 | 24.8 | 24.8 | 24 | 23.6 | 50.2 | 48.4 | 50 | 47.4 | 47.6 |
| 10 | 1000 | 4 | 137.6 | 141.8 | 137.6 | 137.6 | 137.6 | 137.6 | 137.6 | 143.2 | 143.8 | 143.4 | 141.8 | 140.2 |
| 10 | 1000 | 8 | 69 | 73.2 | 69.2 | 69 | 69.2 | 69 | 69 | 87.8 | 84.8 | 85.8 | 86 | 85.2 |
| 10 | 1000 | 12 | 46.2 | 50.4 | 46.6 | 46.8 | 46.8 | 46.6 | 46.2 | 77.2 | 78 | 74.4 | 76.4 | 74.8 |

Table 2: Results of the instances with $P \in \{20, 30\}$

| P | N | H | Heu | | | | | | | Ran | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | LB | UB | K=0 | K=10 | K=20 | K=50 | K=100 | K=0 | K=10 | K=20 | K=50 | K=100 |
| 20 | 50 | 4 | 7.2 | 19.6 | 9 | 9 | 8 | 7.2 | 7.2 | 13 | 12.2 | 10.8 | 10.2 | 9.6 |
| 20 | 50 | 8 | 3.8 | 19 | 7.8 | 7 | 6.6 | 5.4 | 3.8 | 12.8 | 13 | 10.8 | 8.8 | 8.8 |
| 20 | 50 | 12 | 2.8 | 19 | 7.8 | 6.6 | 5.6 | 4 | 2.8 | 13.6 | 12.4 | 11 | 8 | 8 |
| 20 | 100 | 4 | 14.2 | 25 | 16 | 16 | 16 | 14.2 | 14.2 | 21 | 21.8 | 22.2 | 17.8 | 16 |
| 20 | 100 | 8 | 7.4 | 19.6 | 10.8 | 11 | 10.8 | 10.2 | 7.6 | 20.6 | 19.4 | 19.4 | 17.4 | 13.4 |
| 20 | 100 | 12 | 5 | 19.2 | 10.2 | 9.8 | 9 | 9 | 7.6 | 21.2 | 21.4 | 20 | 18.4 | 13 |
| 20 | 200 | 4 | 27.6 | 37.6 | 29.8 | 30.2 | 29.8 | 28.8 | 27.6 | 39.6 | 37 | 37.2 | 36.4 | 31.8 |
| 20 | 200 | 8 | 14.2 | 25.2 | 17.6 | 18 | 18 | 17.4 | 17.4 | 33.4 | 35.4 | 32 | 31.8 | 28.6 |
| 20 | 200 | 12 | 9.4 | 21.6 | 13.8 | 13 | 13.8 | 13.2 | 13.4 | 33.8 | 31.6 | 31.8 | 30.2 | 30.8 |
| 20 | 500 | 4 | 67.6 | 77.2 | 70.2 | 70.4 | 70 | 70.4 | 67.6 | 80 | 81 | 80.8 | 80.4 | 77.8 |
| 20 | 500 | 8 | 34 | 43.6 | 38 | 37.6 | 37.8 | 38 | 38 | 64.2 | 64.6 | 62.6 | 61.4 | 61 |
| 20 | 500 | 12 | 22.8 | 33.2 | 27 | 27.2 | 27.2 | 27.2 | 27.2 | 60.8 | 61.6 | 62.4 | 59.8 | 58 |
| 20 | 1000 | 4 | 134.4 | 143.6 | 135.8 | 135.8 | 136.2 | 136 | 134.4 | 148 | 146 | 148.6 | 148.4 | 147.4 |
| 20 | 1000 | 8 | 67.6 | 76.8 | 70.6 | 71 | 70.4 | 70.4 | 70.6 | 102.2 | 101.6 | 101 | 100.2 | 100.8 |
| 20 | 1000 | 12 | 45.2 | 54.4 | 48.8 | 49.8 | 49.2 | 49.2 | 48.8 | 93.8 | 97.4 | 97.4 | 94.8 | 91.6 |
| 30 | 50 | 4 | 8 | 29.8 | 10 | 9.8 | 9.4 | 8 | 8 | 15.8 | 13.4 | 11 | 10.6 | 10.6 |
| 30 | 50 | 8 | 4.2 | 29 | 8.2 | 7.8 | 7.6 | 5.2 | 4.2 | 15.8 | 13.2 | 11.6 | 10 | 9.6 |
| 30 | 50 | 12 | 3 | 29 | 8 | 7.4 | 6.8 | 5.2 | 3.4 | 14.6 | 14.4 | 12.8 | 9 | 8.8 |
| 30 | 100 | 4 | 13.8 | 33.4 | 16.8 | 16.6 | 16.2 | 15.6 | 13.8 | 22.4 | 22.8 | 22.8 | 19.6 | 16.6 |
| 30 | 100 | 8 | 7.2 | 29.6 | 12 | 11.4 | 11.8 | 10.8 | 9.8 | 22 | 21.8 | 20.8 | 19 | 15 |
| 30 | 100 | 12 | 5 | 29 | 10.8 | 10.4 | 10.2 | 9.8 | 8.2 | 21.8 | 21.8 | 20.4 | 19.8 | 15.2 |
| 30 | 200 | 4 | 27.6 | 43.8 | 30.4 | 31 | 31.6 | 31.2 | 28 | 40.6 | 39.8 | 41.2 | 38.6 | 34 |
| 30 | 200 | 8 | 14 | 32.2 | 18.6 | 18.8 | 19 | 19.4 | 19.8 | 36.4 | 36.2 | 36.2 | 36.8 | 32.8 |
| 30 | 200 | 12 | 9.6 | 29.8 | 17 | 16.6 | 16.2 | 16.2 | 16.2 | 35.6 | 36 | 37 | 34.8 | 33.4 |
| 30 | 500 | 4 | 64.8 | 80.4 | 69.6 | 69.6 | 69.6 | 69.4 | 69.2 | 83.2 | 82.2 | 82.8 | 82 | 81.8 |
| 30 | 500 | 8 | 32.6 | 49 | 38.4 | 38.6 | 38.6 | 38.2 | 37.8 | 68.8 | 66 | 68.6 | 70.6 | 66.4 |
| 30 | 500 | 12 | 22.2 | 39.2 | 29 | 29 | 28.6 | 28.6 | 28.8 | 69.2 | 67.4 | 66.2 | 66.6 | 66 |
| 30 | 1000 | 4 | 128.8 | 144 | 133.6 | 133.6 | 133.6 | 133.8 | 133.6 | 149 | 150.8 | 150 | 147.8 | 150.6 |
| 30 | 1000 | 8 | 64.8 | 80.4 | 71.4 | 70.6 | 71 | 70.6 | 70.6 | 111.2 | 112.8 | 111.8 | 110.6 | 110 |
| 30 | 1000 | 12 | 43.2 | 59.4 | 50.8 | 50.6 | 50.8 | 50.8 | 50.8 | 107.6 | 108.8 | 108.6 | 107 | 101.2 |

condition where $K$ is equal to zero. However, the results between the easy instances and difficult instances have some differences. Comparing with the significant influence of different $K$ in the difficult instances, the influence of different $K$ in the easy instances isn't obvious. The same conclusion can be drawn if we focus on one specific parameter or factor while keeping other parameters unchanged.

As far as we are concerned, we think it is the density and sparseness of stacks that cause this consequence. In other words, the smaller value of each variable is, the same kind of containers that have the identical origin port and destination port have more chance to be placed in the same stack and less rehandles will arise. We call this kind of stack with high density, and the high sparseness in contrast.

For the instance with the same $P$, $N$ and $H$, we call it benchmark instance when $K$ is equal to zero. If the number of used stacks is larger than the benchmark instance when $K$ isn't equal to zero, we call it an exceptional instance. There are few exceptions when we analyze the result through a large amount of experiments. Comparing with the zero rehandles, the number of needed stack is even larger when the allowed rehandle $K$ isn't equal to zero, though the deviation is not great.

We then find out those exceptional instances and compare the stowage planning with the same instances under the zero rehandle constraint, and we think it's the distribution status of previous containers to be blamed at first. The allowed rehandles delay the production of new stacks but not to reduce them, that's why the problem happens. Actually, it's the unloading strategy for blocking containers that causes this exceptional condition. Because we reset the origin port of blocking containers, the loading containers in current port increases and meanwhile the allowed rehandles run out. In the end, the new stacks are needed under the coincidence. In order to prove it, we output the layout of the exceptional instances and the corresponding benchmark instances. By comparing the exceptional instances with the corresponding benchmark instances, we get the conclusion that both the distribution of containers and the algorithm are blamed for this kind of abnormal condition. In our heuristic algorithm, we reset the blocking containers' origin ports as the current port, and it enlarges the number of containers to be loaded. Meanwhile, the allowed rehandles run out and thus new stacks are needed to stow those new outbound containers.

However, it doesn't mean that our algorithm isn't universal because the probability of causing exceptional instances is very low and it hardly happens in our practical operations. Therefore, our heuristics algorithm is feasible for most of the instances and has a good performance as well. The following tables and figures can be used to illustrate the good performance of our algorithm.

We then analyze the average result of all instances to get some conclusions. Table 3 and Figure 2 give the average results of heuristic and random algorithm when the allowed rehandle $K$ has different values. It's not difficult to draw the conclusion that the number of used stacks gets smaller as the allowed rehandle becomes larger for both heuristic algorithm and random algorithm. Another conclusion is, comparing with random algorithm, our heuristic algorithm has a fairly good performance to work out the solution to the SSMP. It's noteworthy that the average results in Table 3 include the instances with $N = 5000$.

Table 3: average results of the instances with different K for two algorithms

| $K$ | 0 | 10 | 20 | 50 | 100 |
|---|---|---|---|---|---|
| $Numofstack$ of Heu | 99.3 | 99.21944 | 99.08333 | 98.73889 | 98.35 |
| $Numofstack$ of Ran | 114.62778 | 114.31667 | 114.21448 | 112.86111 | 111.9 |

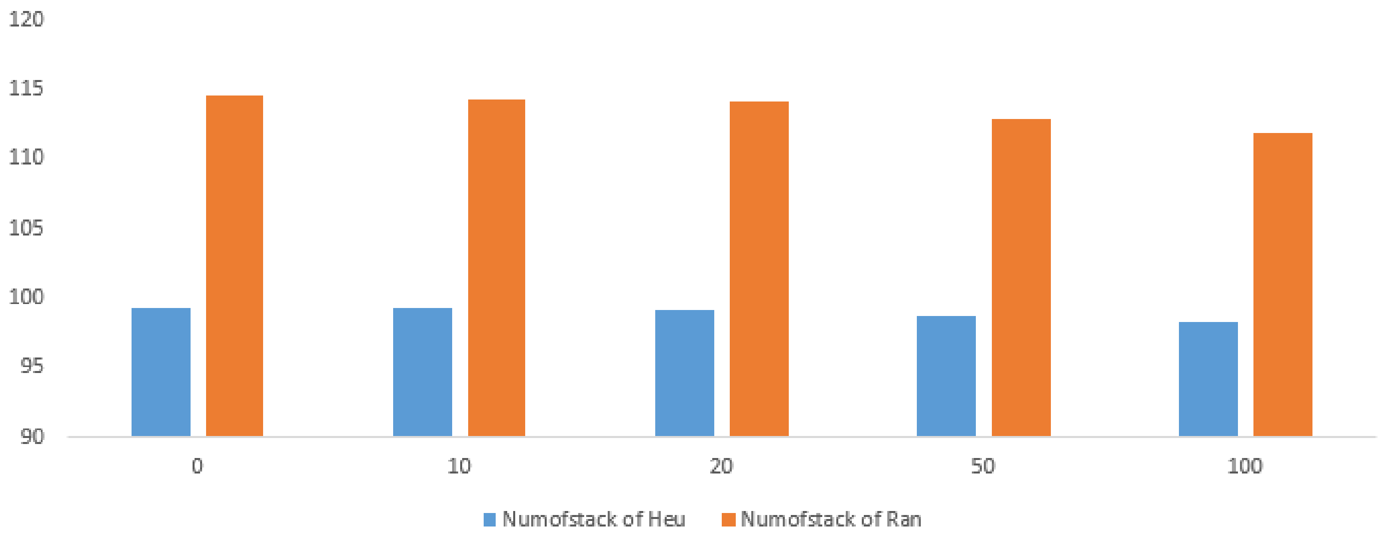

Figure 2: Numofstack of Heu and Ran with different K.

There is a simple example to elaborate why a certain number of allowed rehandles can improve the utilization of containership comparing with the zero rehandle. Assuming that there are thirteen containers to be transported along the voyage with six ports. We let C(i,j) denote index container whose origin port is i and destination port j. The following is the way to denote those containers: four containers are C(1,3), one is C(2,4), one is C(2,6), four are C(3,5), one is C(4,6) and two are C(5,6).

If they are loaded without any rehandles, the loading sequence and layout will be the following Figure 3.

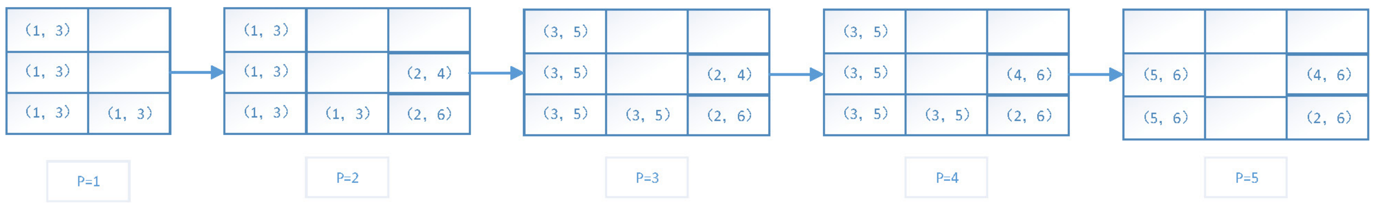

Figure 3: stowage plan without rehandle.

If they are loaded with rehandles, the loading sequence and layout will be the following Figure 4.

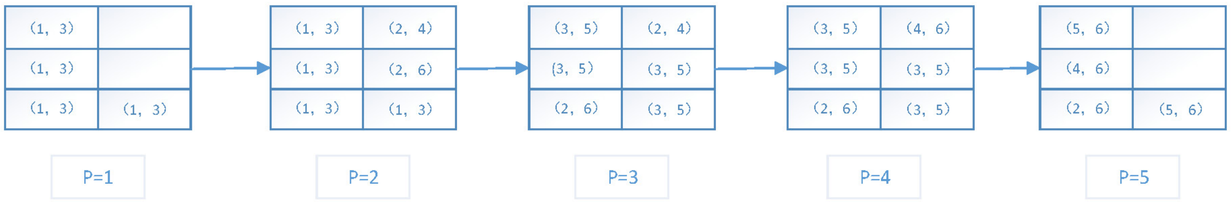

Figure 4: stowage plan with rehandle.

From Figure 3 and 4, we can clearly draw the conclusion that allowing a certain number of rehandles will reduce the number of stacks used to store containers in a vessel compared with zero rehandle. That's to say, the utilization of containership can be improved if some rehandles are allowed.

In fact, we have tried a related algorithm named *P&H* algorithm before adopting our current heuristic algorithm to get the stowage planning. This algorithm is actually a process of parameter tuning, our original intention is attempting to find out better solution to stow containers. Nevertheless, the result through this algorithm isn't as good as the heuristic algorithm and it doesn't work well as we expected if the value of $K$ is small. From another point of view, it just shows the good performance of our heuristic algorithm.

In our *P&H* algorithm, we give some restrictions on the height and the destination port of current loading container when choosing stack for each container. All the parameters in *P&H* value between 0 and 1, zero is not included. The former parameter represents the ratio of the loading container's destination port and the farthest port $P$. The value of this parameter is 1 if the loading container's destination port is $P$. The latter parameter represents the ratio of stack's height and the limited height $H$. The main idea of *P&H* algorithm is to take the destination port of loading container and the limit height of stack into consideration when choosing stack for the target container. The purpose of this algorithm is to load those containers with farther destination port into the bottom of stack and the computational result will be listed in the Table 4.

Table 4: average results of the instances with *P&H* algorithm

| *P&H* | 0 | 10 | 20 | 50 | 100 |
|---:|---:|---:|---:|---:|---:|
| 0.75&0.75 | 99.32222 | 99.24167 | 99.08056 | 98.725 | 98.29167 |
| 0.75&0.80 | 99.32222 | 99.24167 | 99.08056 | 98.725 | 98.29167 |
| 0.80&0.75 | 99.30833 | 99.225 | 99.05556 | 98.71111 | 98.31389 |
| 0.80&0.80 | 99.30833 | 99.225 | 99.05556 | 98.71111 | 98.31389 |

Table 4 shows the result with *P&H* algorithm, which is firstly designed to improve the original algorithm and eventually it is used to prove the good performance of our original algorithm. It's noteworthy that the average results in Table 4 include the instances with $N = 5000$. Both the values of two parameters of the algorithm are selected form {0.75, 0.8}. It's clear that the result shows the allowed rehandles will decrease the number of used stacks as the above results in Table 3. We can draw a conclusion that the first parameter has a significant influence on the result while the second parameter makes no difference if we compare the result in each line. I suppose that it happens as a result of the value set of $H$. Comparing with the results shown in Table 3, we could find that original algorithm we propose works better when the value of $K$ is small while the $P\&H$ algorithm gets the similar results when the value of $K$ is large.

To find the relationship between the related parameters and the result got from our heuristic, we use some data from our results obtained from heuristic algorithm to produce the Figure 5 to make it visual.

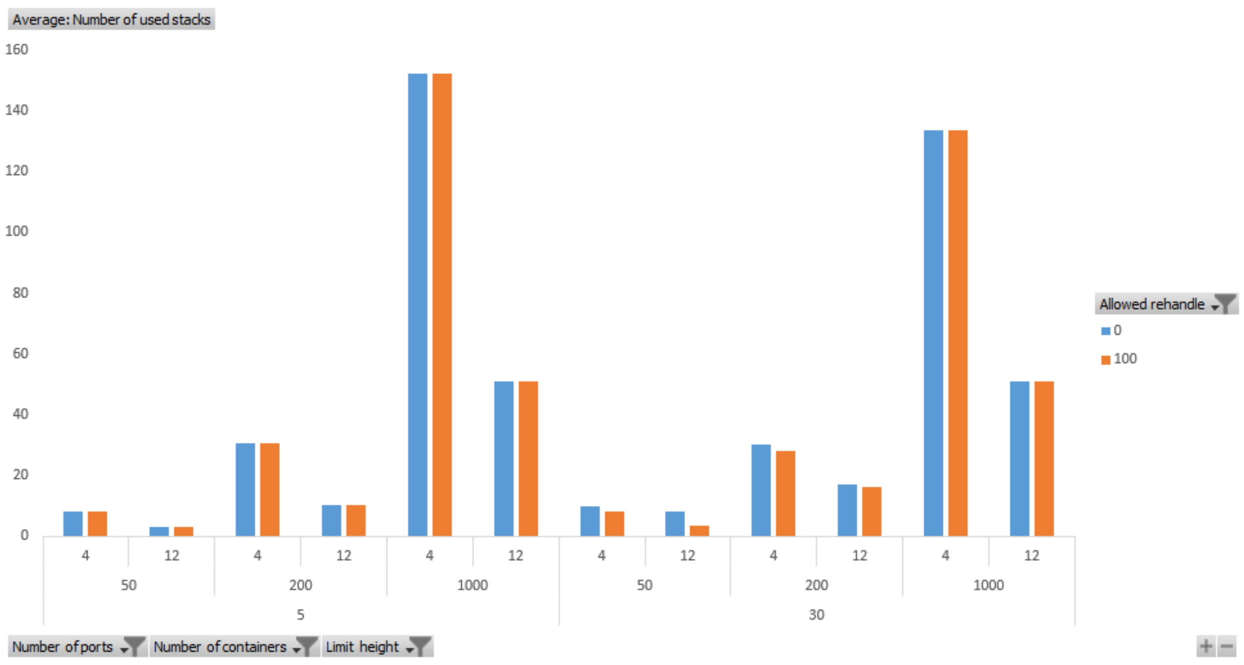

Figure 5: comprehensive analysis.

Seeing from Figure 5, we can get the following relationships. On one hand, the more ports, the more containers and the higher the limit height will result in the more needed stacks if we only consider the factor we want to analyze. On the other hand, the difference in the results between zero rehandle and certain allowed rehandles become obvious as the number of port gets larger, the limit height gets higher and the number of container gets smaller.

In short, the rationality and good performance of our heuristic have been confirmed through all the above tables and figures.

## 6. Conclusion

The stowage stack minimization problem with K-rehandle constraint (SSMP-KR) is aimed to find out a stowage plan that fewest stacks on a containership are required to accommodate given containers throughout a voyage by subject to K-rehandles.

In this paper, we analyze the structure of this containership stowage planning problem and its relationship with the SSMP-ZR problem proposed in Wang et al. (2014).

We can regard the stowage stack minimization problem with zero rehandle as an particular case when we choose the value of K as zero. Hence, the problem proposed in this paper is a generalized and extended problem of SSMP-ZR considering the existence of K-rehandle.

In this paper, we presented the integer programming model for the SSMP-ZR. However, it is hard to build an IP model for the general SSMP-KR.

A heuristic approach is proposed to construct near-optimal solutions to the SSMP-KR problem in a very short computational time. Since there are different status in the process of handling containers, different methods in our algorithm are thought to cope with the corresponding conditions.

The experimental results show that our heuristic approaches generate very promising solutions on a variety of instances. Comparing with the stowage stack minimum problem with zero constraint, our problem with K rehandle constraint can offer a certain flexibility for the actual problem. The results show the problem we put forward is of practical significance and it can improve the utilization of containership. Our algorithm uses greedy rules to find the best slot for each container, hence the results may be the near-optimal rather than global optimal. As we all know, it is difficult to get the global optimal for this kind of problem. Applying this algorithm into the real shipping management can enhance the hull space utilization and reduce some spending in the case of improving the utilization of containership. The novelty and practicality show the importance of this paper and it can give a reference to the future study.

Furthermore, the rehandles are unavoidable in the actual port operation, which indicates the problem we propose in this paper does make some sense and the solution we work out using our heuristic algorithm can provide a preliminary assessment for forwarders to arrange linear shipping company for transportation.

## References

Ambrosino, D., Paolucci, M., & Sciomachen, A. (2015). A mip heuristic for multi port stowage planning. *Transportation Research Procedia*, 10, 725–734.

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

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

Bortfeldt, A. & Gehring, H. (2001). A hybrid genetic algorithm for the container loading problem. *European Journal of Operational Research*, 131(1), 143–161.

Bortfeldt, A., Gehring, H., & Mack, D. (2003). A parallel tabu search algorithm for solving the container loading problem. *Parallel Computing*, 29(5), 641–662.

Botter, R. & Brinati, M. (1991). Stowage container planning: A model for getting an optimal solution. In *Proceedings of the IFIP TC5/WG5. 6 Seventh International Conference on Computer Applications in the Automation of Shipyard Operation and Ship Design, VII* (pp. 217–229).: North-Holland Publishing Co.

Cohen, M. W., Coelho, V. N., Dahan, A., & Kaspi, I. (2017). Container vessel stowage planning system using genetic algorithm. In *European Conference on the Applications of Evolutionary Computation* (pp. 557–572).: Springer.

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(1), 251–261.

Dillingham, J. & Perakis, A. (1986). Application of artificial intelligence in the marine industry. In *Fleet Management Technology Conference*: Boston.

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

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(6), 585–599.

Haghani, A. & Kaisar, E. (2001). A model for designing container loading plans for containerships. In *Annual Conference for Transportation Research Board* (pp. 55–62).

Jensen, R. M. (2010). *On the complexity of container stowage planning*. Technical report, Tech. rep.

Kammarti, R., Ayachi, I., Ksouri, M., & Borne, P. (2009). Evolutionary approach for the containers bin-packing problem. *Studies in Informatics and Control*, 18(4), 315–324.

Malucelli, F., Pallottino, S., & Pretolani, D. (2008). The stack loading and unloading problem. *Discrete Applied Mathematics*, 156(17), 3248–3266.

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

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

Scott, D. & Chen, D.-S. (1978). A loading model for a container ship. *Cambridge: University of Cambridge*.

Shields, J. J. (1984). *Containership stowage: A computer-aided preplanning system*. Marine Tech.

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

UNCTAD (2016). *Review of maritime transport*. Technical report, United Nations.

Wang, N., Zhang, Z., & Lim, A. (2014). The stowage stack minimization problem with zero rehandle constraint. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems* (pp. 456–465).: Springer.

Webster, W. & Van Dyke, P. (1970). Container loading: a container allocation model: I-introduction background, ii-strategy, conclusions. In *Proceedings of Computer-Aided Ship Design Engineering Summer Conference. University of Michigan*.

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(3), 137–145.

Zehendner, E., Feillet, D., & Jaillet, P. (2017). An algorithm with performance guarantee for the online container relocation problem. *European Journal of Operational Research*, 259(1), 48–62.

Zhang, W.-y., Lin, Y., Ji, Z.-s., & Zhang, G.-f. (2008). Review of containership stowage plans for full routes. *Journal of Marine*

*Science and Application*, 7, 278–285.

Zhang, Z. & Lee, C.-Y. (2016). Multiobjective approaches for the ship stowage planning problem considering ship stability and container rehandles. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 46(10), 1374–1389.