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 container stowage problem concerns with suitable placements of containers on a container-ship in a multi-port voyage. Many scholars have conducted research with the objective of minimum rehandles. This paper investigates a variant which accommodates given containers with the minimum stacks subject to K rehandles; we call it stowage stack minimization problem with K-rehandle (SSM -KR). The variant facilitates forwarders to evaluate the value of received individual shipping orders. In this paper, we analyze the structure of SSM - KR and its relationship with the circle graph. A heuristic approach is proposed to construct near-optimal solutions in a very short computational time. Experimental results show the effect of instance parameters on solutions.

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

1. Introduction

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. Over 60% of general cargoes are transported by containership at present. This mode is deeply accepted by shipper and carrier, and is becoming the main trend of freightage in the world. It shows powerful vitality and promising prospects. 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 contains for containershipZhang et al. (2008). 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. These problems have troubled and aroused the interests of both scholars and commercial shipping organization in many countries since 1970s. With the development of container-shipping, more and more containers are transported by sail. Up to now, no satisfying solution has development of container transportation trade. Thus, it makes higher demand of the problem.

The previous research ignores the importance of the capacity of ship and give much emphasis on the balance of ship, the number of rehandle and so on. Sometimes we need think about the importance of capacity

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

as well. For example, if we can use each stack sufficiently, the required number of ships will be reduced and thus the cost decreases. The problem mentioned in this paper give a priority on how to reduce the use of stack with K rehandle as much as possible. Through the research on this problem, another aspect of container stowage planning is exhibited and a lot of shipping cost can be saved by using what we propose in this paper. On the other hand, it represents a novel thinking perspective to make a research about this kind of problem and it will promote the development of academic shipping optimal problem.

We put forward the stowage stack minimization problem with K rehandle constraint problem (SSMP-KR) in this paper. Literally speaking, it is a container stowage planning problem whose objection is to find out minimization stack with K rehandle constraint on a given condition. The heuristic algorithm in this paper uses the greedy rule to finish the loading and unloading operation for every container to be loaded and unloaded in each port along the voyage. Therefore, the solution we get may be the approximately optimal rather than the global optimal.

In the remainder of this paper we provide a brief description of previous research and the current research trend (section 2) and the problem description is in section 3. In section 4 we give our heuristic algorithm for the problem and give specific explanation about the algorithm. Experiments and analysis as well as the comparison are shown in section 5. Concluding remarks will be discussed in section 6.

2. Literature Review

The container stowage problem can be divided into two classes: one is management and programming at the container terminal, the other is stowage plan for containership. This paper mainly focuses on the latter. And sorted by research method, this problem will be divided into six categories: simulation based on probability, heuristic procedures, mathematical programming approach, decision support system, expert system and evolutionary computation. The following is the approximate development process of containership stowage plan.

Since 1970s, the stowage plan problem has been studied by shipping organization for commercial demand. They tried to find a good solution which would satisfy fewer shifting and higher efficiency when loading and discharging containers. Dyke and WebsterWebster & Van Dyke (1970) first tried to solve the problem by using computer. They put forward a heuristic algorithm that could create a random stowage plan. ShieldsShields (1984) introduced a computer preplanning system(CAPS) which had been used by American President Lines. The CAPS produced stowage plan by using Monte Carlo theory. Scott and ChenScott & Chen (1978) constructed a model, in which containers were sorted into ten classes based on some characteristics and then three heuristic rules were used to assign gradually each type of containers to slots of the ship by four steps, while the shifting problem had not been considered in their model. Avriel and PennAvriel et al.

(1998) presented a 0-1 binary linear programming formulation based on minimizing the number shifting in a single bay of a ship calling at a given number of ports. Further, they showed that finding the minimum number of columns for which there was a zero shifts stowage plan is equivalent to finding the coloring number of circle graphs. However, the formulation of many binary variables and constraints made the working-out of model quite different, so the Suspensory Heuristic procedure is developed to solve the model. Botter and BrinatiBotter & Brinati (1991) created a model called complete mathematical model for the stowage plan problem, which used binary decision variables to determine the containers unloading and loading sequence for each port. The objective function of this model for the stowage plan involved two important factors: the number of re-stowing along the route and the longitudinal crane movement along the quay during the container loading and unloading operation. The number of overstowages along port the ship calling at, weight limiting in one vertical stack, the stability, structural stress and other feasible conditions were considered as constraints. The advantages were that the model considered every port the ship calling at as a different stage of the whole system, which made the model much more correspond to the actual cases. But too many factors were considered in the model, the dimension of formulization was too large to solve because of large number of binary variables and constraints arising in the model. Taboo search was employed to obtain an optimal solution by WilsonWilson et al. (2001) and his model was feasible for it can change configuration step by step until each container was assigned to specific slot. Expert system is a most active research and practical field of Artificial Intelligent (AI). Dillingham and PerakisDillingham & 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. Database in the system could not only accomplish the stowage plan, but also track the location of the container. It was highly interactive, the user could interrupt the system at any time when the planning proceeded and overrode or revoked suggestions that the system had made. Algorithm of evolutionary computation came from Darwins biologic theory of evolution. There are two or three branches at present, among which genetic algorithm is used widely. Dubrovsky O. and Levitin G.Dubrovsky et al. (2002) put forward a compact encoding technique, whose objective was to overcome disadvantages of complete encoding. The compact encoding method just saved the mutative part after the operation of loading and unloading completed along the route of ship calling, rather than holding the complete layout. Since the ship layout has a relatively small number of changes at each port, the size of solution encoding vectors would become much smaller than complete code. So this method would significantly reduce space of solution and advance efficiency of algorithm. R. KammartiKammarti et al. (2013) proposed an efficient genetic algorithm which consists on selecting two chromosomes(parent) from an initially constructed population using a roulette wheel technique. Then, the two parents are combined using a one point crossover operator. Finally, a mutation operator is performed. TierneyTierney et al. (2014) et al. investigated the complexity of stowage planning problem and showed that the capacitated k-shift problem is solvable in polynomial time for any choice of stacks and stack capacities. And some scholars paid special attention to the process of loading and unloading, MalucelliMalucelli et al. (2008) investigated stack reordering strategies aiming at minimizing the number of pop and push operations and focused on three versions of the problem in which reordering can take place in different phases: when unloading the stack, when loading it or in both phases. AmbrosinoAmbrosino et al. (2015) et al. 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. To the best of our knowledge, apart from the SSMP-ZR, only AvrielAvriel et al. (2000) et al. and JensenJensen (2010) discussed the stowage stack minimization and its connection with the chromatic number of circle graphs when the stack height is unlimited.

At present, the research tendency of the stowage planning problem is intelligent management system, which is a new stowage research direction. Intelligent containership stowage plan system is a technological innovation in contrast with traditional stowage method. It can improve the ability of decision-making management system of container shipping by means of artificial intelligence, information technology, communications and computer technology.

3. Problem Description

The basic description of problem is not of significant difference from the stowage stack minimization problem with zero rehandle constraint and the specific context is in the followings.

A ship starts its journey at port 1 and sequentially visits port 2, 3, ..., P (port P could be the same as port 1 in a circular route). There are N standard containers to be shipped along the journey. At each port, the containers destined to the current port are discharged, and the containers stowed at the port are loaded to the ship. Each container i is characterized by its shipping leg $O(i) \rightarrow D(i)$, indicating that the container is loaded at port $O(i)$ and discharged at port $D(i)$. All the loading and discharging ports of N containers are known in advance.

We assume that the ship is empty initially and it has enough capacity to accommodate all the containers along the journey. The ship spaces are divided into several stacks. Each stack can hold at most H vertically piled containers, i.e., the height limit of every stack is H. When H is bounded, we call the SSMP-KR as the capacitated SSMP-KR (CSSMP-KR); when H is sufficiently large (e.g., $H \geq N$) or unbounded, we refer to it as the uncapacitated SSMP-KR (USSMP-KR). For the CSSMP-KR, 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. The objective of the SSMP-KR is to transport all the containers using the fewest number of stacks with K rehandles allowed. The USSMP-KR is far from the real situation in the stowage planning, since stack heights of ships are restricted by operation cranes. Hence, this paper give emphasis on the CSSMP-KR and for the convenience of description, we use SSMP-KR for short in this paper.

A feasible solution to the SSMP-KR is represented by a operation set, from which we can clearly know the slot changing of each container, even when there is an overstowage. The detailed introduction will be given in the below.

We also denote the dynamic *Layout* as the layout of the ship after one operation for one container. We can output the layout after the operation of one container is done or the layout after all the operations at one specific port.

4. Methodology

In this section, we talk about the heuristic algorithms to the SSMP-KR, as well as the basic performance guarantee of the algorithms.

4.1. Heuristic Algorithms for the SSMP-KR

In our algorithm, two classes are introduced in order to describe the containers' characteristic and record the operation for containers. They are Container class and OPeration class, respectively. For each container, we use arraylist "ac" to store its sequence number, origin port and destination port. No matter what the operation is , we add all of operations into the arraylist "Solution" to record every container's slot changing from the shore or previous stack to the chosen stack or shore. Other important global variables are Count_rehandle and Numofstack. The former is no greater than the given K, and the latter no longer change until the end of the shipping. What methods aren't exhibited in the below algorithm are read() and bound(). The method read() is used to read the parameters and the detailed containership information from each instance. As for the method bound(), we use it to give a upper bound for our algorithm and the rationality proved. Another important method "GetstackNum", which is used in both loading_rehandle_lessK and loading_rehandle_equalK method, is used to get the sequence number of chosen stack.

If there is an overstowage, we use arraylist S_1 to store partial stacks without rehandle happening, arraylist S_2 to store partial stacks with rehandle happening, arraylist S_3 to store empty stacks whether there is a rehandle or not. The priority order of them is S_1, S_2, S_3. If there isn't an overstowage, we use S_0 to store the feasible stacks. For ease of illustration, we need to define a one dimensional array called nearport to store the nearest port for every stack. For example, three containers are stored in the same stack, and their destination port are 4,5,6, respectively, then the value of nearport of this stack is 4. If a stack is empty, its value will be set to $P + 1$, P is the total number of the ports along the voyage. We firstly talk about the simple condition that the allowed rehandles is full. When traversing all stacks to choose the optimal according to our rules for each container, we add non-full stacks whose nearest port is greater than or equal to the current loading container into arraylist S_0. For the stacks in S_0, we choose the one with the nearest port. Next we talk about the complex condition that a rehandle is allowed and it is a little different from the simple condition. Under this condition, S_1 have the highest priority. If S_1 is null, we choose the stack from arraylist S_2 which is used to store partial stacks whose nearest port is smaller than the current loading container. Both in S_1 and S_2, we choose the stack whose nearest port is smallest. If S_2 is null, we choose the first stack we traverse in the S_3 which is used to store empty stacks. So it is the loading rules above and the following is the discharging rules.

When discharging containers, we just need to unload containers from top to bottom one by one when the container's destination port is the current port if there isn't an overstowage in the stack. when an overstowage happens, we have to unload the blocking container(s), then we can unload those right ones. As for blocking containers, we reset their origin ports as current port and their destination ports remain unchanged. That's to say, the outbound containers at the current port increases.

We give out some parts of algorithms as followings:

For all the ports along the voyage, we adopt the heuristic method to dispose containers for the given input. And we can transfer the input into the output by applying the heuristic method. So the below is the main process of our algorithm.

The following pseudo gives a simplistic explanation of heuristic method. The variables ”height”, ”Layout” and ”Numofstack” are one dimensional array, two-dimensional array and the number of accumulative used stack in the pseudo, respectively. $Unloading\_at\_port(p)$ and $Loading\_at\_port(p)$ methods provide the way how to unload the right containers and load the right containers at each port. Given a specific instance, we can get the correspondent value of ”Numofstack”.

The following unloading_at_port() method is used to decide which container should be discharged at port p. Then we need substitute the related information of chosen container into the Unloading_for_container(s,t,p) method.

Considering the length of pseudo code, some details are omitted. We will give a explanatory about the following pseudo code. We first define a one-dimensional array nearport which consists of "UB+1" integer number and it is used to store the nearest destination among all containers located in each stack. If the stack is empty, then the value will be set as "P+1". The ac is an arraylist to store all containers to store all the containers before the shipping. If one container is to be unloaded and there isn't overstowage, then it will be removed from the arraylist. For each blocking container, its origin port will be reset as current port after it is removed from its location, then it will be added into ac again.

As a result of the existence of rehandle, we need sort ac before we start loading containers at port p. The sorting rules are shown as followings. We compare origin port of containers in the first place, the number smaller, the location ahead. If the origin port is identical, compare the destination port, the number larger, the location ahead. If ac is not empty, we just need to choose the first element *cn* to load. We use loading_for_container method to choose the optimal stack. After loading it into the stack *st*. *st*'s height and nearport will be updated. At last, record this operation and add the operation into the arraylist Solution. Hence, we can eventually get all the operation information.

The following Loading_for_container() method provides us the method how to load containers and two different sub-methods are given to return the stack where the loading container belongs. The specific procedure of two sub-methods is described in the above content.

The below pseudo code of $unloading\_for\_container()$ method shows us how to unload containers one by one whether there is overstowage or not. Before the unloading operation,we need input the current port, the stack and the tier where the unloading container is. If there is overstowage, we invoke "$Relocate(s,t)$" method. After processing blocking container(s), we set out to unload the right container. We firstly record the unloading operation and add it into the arraylist Solution. They remove the container from its previous location and reduce the height of stack where it is located.

In a word, a lot of variables and methods of the concrete Unloading and Loading plan have been involved, we don't give the correspondent pseudo here for the sake of context length. What is identical for the two procedures is that they find the optimal strategy for each container during the unloading and loading process according to the greedy rule made by our algorithm. As is explained in the heading of this section, we divide the loading method into two parts on the basis of the number of reached rehandles. And similarly, the unloading method are divided into two conditions whether there is overstowage. Hence, specific steps are a little different from each other between the two circumstances.

Our algorithm can give the detailed location changes of each container and output the container status at arbitrary time. That's to say, we can grasp the container stowage process before the voyage as long as we know some information in advance. Therefore, our algorithm makes a little difference and make 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 if the value of K is zero and we regard the port number P as a fixed number. That's to say, the SSMP-KR becomes SSMP-ZR under this condition. For ease of exposition, we first give some related notations.

- $\mathcal{U}^*$ & C^*: the optimal solution to the USSMP-ZR and the SSMP-ZR instances, respectively.
- $\mathcal{U}$ & C: the solution to the USSMP-ZR and the SSMP-ZR instances by the algorithm, respectively.
- $\mathcal{U}_p$ & C_p: the number of stacks used at port p 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. 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. *For the USSMP-ZR instance, the heuristic approach generates a solution in which at most v_p stacks are used before the ship departs from port p, i.e. $\mathcal{U}_p \leq v_p, \forall p = 1, \ldots, P$.*

Proof. We use an inductive proof.

Basis: For $p = 1$, without loss of generality, we assume that port 1 is a loading port. The heuristic piles up containers in one stack in a way that the containers with later destinations are placed in lower tiers. We have $\mathcal{U}_1 = v_1 = 1$, and therefore $\mathcal{U}_1 \leq v_1$ holds.

Inductive step: Suppose that before the ship departs from port p, there are at most v_p stacks used, i.e., $\mathcal{U}_p \leq v_p$. Upon the ship arrives at port $p + 1$, it first unloads the containers from the ship. This process does not increase the number of stacks utilized. If there are containers to be loaded at port $p+1$, then $v_{p+1} = v_p + 1$, otherwise $v_{p+1} = v_p$.

(1) If there exist some containers to be loaded, according to our heuristic, they are placed on either the extant stacks or a new blank stack (with the containers of later destinations placed lower). Hence, $\mathcal{U}_{p+1} \leq \mathcal{U}_p + 1 \leq v_p + 1 = v_{p+1}$.

(2) If no container is loaded, $\mathcal{U}_{p+1} \leq \mathcal{U}_p \leq v_p = v_{p+1}$.

Based on the above two steps, we have $\mathcal{U}_p \leq v_p, \forall p = 1, \ldots, P$. Furthermore, we have

$$\mathcal{U} = \max_{p=1,\ldots,P} \mathcal{U}_p \leq \max_{p=1,\ldots,P} v_p = v_P \leq P$$

□

Proposition 2. *The CSSMP-ZR 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. $\square$

Proposition 3. *For the heuristic solution to the CSSMP-ZR 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, $C = \max_p C_p$. Remind that C_p is the number of stacks in the layout L^C_{p,T_p} resulted from our algorithm. The number of partial stacks in the layout L^C_{p,T_p} is no greater than the number of stacks in the associated layout L^U_{p,T_p} (c.f., Definition **??**). The number of stacks of L^U_{p,T_p}, from Proposition 1, is at most v_p, so the number of partial stacks in L^C_{p,T_p} is no greater than v_p. In addition, the number of full stacks in L^C_{p,T_p} is no greater than $\lfloor\frac{N_p}{H}\rfloor$. Therefore, $C_p \leq \lfloor\frac{N_p}{H}\rfloor + v_p$ and $C \leq \max_p(\lfloor\frac{N_p}{H}\rfloor + v_p)$ hold.

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 2. Thus, $\lfloor\frac{N_p}{H}\rfloor \leq C^*_p$. It then holds $\max_p(\lfloor\frac{N_p}{H}\rfloor + v_p) \leq \max_p(C^*_p + v_p) \leq \max_p(C^*_p + v_P) = C^* + v_P$. $\square$

The above proposition indicates that our approximation algorithm has a constant performance guarantee if we regard the number of ports as fixed under the condition that the number of allowed rehandle is zero. In practice, the number of ports is generally much smaller than the number of stacks used along the journey. Hence, the gap between the optimal solution and our heuristic solution is relatively small, which demonstrates that our heuristic can generate promising solutions.

In view of the number of allowed rehandle we give isn't big and it's influence on the result isn't salient. So we use the *UB* to test the effectiveness of SSMP-KR.

5. Experiments and Analysis

This paper uses the test data from the existing literature and we add an extra parameters K into the primary instance 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£$K = 0$ represents the benchmark and comparison object. There are 80 sets of instances in total, which are categorized by three parameters: 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 result isn't shown in the table when $N = 5000$; the height limit H selected from {4,7, 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 "Numofstack" when $K = 0$, $K = 10$, $K = 20$, $K = 50$, $K = 100$, respectively. And this is applicable to the heuristic algorithm as well as the deuterogenic *P&H* algorithm, the simple introduced will be introduced in the next content.

In this section, we will showcase the performance of our heuristic algorithms on a number of test instances. The heuristic was implemented in Java 8 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.

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

P	N	H	UB	K=0	K=10	K=20	K=50	K=100
5	50	4	9.6	8	8	8	8	8
5	50	7	6.8	5	5	5	5	5
5	50	8	6	4.6	4	4	4	4
5	50	12	5	3.2	3	3	3	3
5	100	4	16.8	15	15	15	15	15
5	100	7	10.4	9	8.8	8.8	8.8	8.8
5	100	8	9.6	7.6	7.6	7.6	7.6	7.6
5	100	12	7.6	5.4	5.6	5.4	5.4	5.4
5	200	4	32.2	30.6	30.6	30.6	30.6	30.6
5	200	7	19.4	17.6	17.6	17.6	17.6	17.6
5	200	8	17.4	15.6	15.6	15.6	15.6	15.6
5	200	12	12.4	10.4	10.4	10.4	10.4	10.4
5	500	4	78	76.6	76.6	76.6	76.6	76.6
5	500	7	45.6	44.2	44.2	44.2	44.2	44.2
5	500	8	40	38.6	38.6	38.6	38.6	38.6
5	500	12	40	27.4	26	26	26	26
5	1000	4	153.8	152.2	152.2	152.2	152.2	152.2
5	1000	7	89	87.2	87.2	87.2	87.2	87.2
5	1000	8	78.2	76.6	76.6	76.6	76.6	76.6
5	1000	12	52.8	51	51	51	51	51
10	50	4	12.6	9	8	7.8	7.8	7.8
10	50	7	10	6.4	6.2	5.6	5.6	5.6
10	50	8	10	6	6	5	5	5
10	50	12	9.4	5.6	4.6	4.4	4.4	4.4
10	100	4	19.6	15.8	15.6	15	14.8	14.8
10	100	7	13.6	10.2	10.4	10.2	8.8	8.6
10	100	8	12.6	9.2	9.8	9.6	7.8	7.6
10	100	12	10.4	7	7	7.2	6.4	5.2
10	200	4	33.2	29.6	29.6	28.6	28.6	28.6
10	200	7	21.2	17.6	17.8	18	16.4	16.4
10	200	8	19.2	16.4	16.6	16.2	14.8	14.4
10	200	12	15	12	12	11.8	11.4	9.8
10	500	4	74.2	69.8	69.8	69.8	69.8	69.8
10	500	7	44.4	40.6	40.4	40.2	40	40
10	500	8	39.4	35.6	35	35.2	35	35
10	500	12	28	25.2	24.8	24.8	24	23.6
10	1000	4	141.8	137.6	137.6	137.6	137.6	137.6
10	1000	7	83.2	79	79	79	79	79
10	1000	8	73.2	69	69	69	69	69
10	1000	12	50.4	46.8	46.8	46.6	46.2	46.6

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

P	N	H	UB	K=0	K=10	K=20	K=50	K=100
20	50	4	19.6	9	9	8	7,2	7.2
20	50	7	19.2	7.8	7	6.6	5.2	4.4
20	50	8	19	7.8	7	6.6	5.4	3.8
20	50	12	19	7.8	6.6	5.6	4	2.8
20	100	4	25	16	16	16	14.2	14.2
20	100	7	20.4	11.6	11.4	11.2	10.6	8.4
20	100	8	19.6	10.8	11	10.8	10.2	7.6
20	100	12	19.2	10.2	9.8	9	9	7.6
20	200	4	37.6	29.8	30.2	29.8	28.8	27.6
20	200	7	26.8	19.2	20	19.4	18.8	18.2
20	200	8	25.2	17.6	18	18	17.4	17.4
20	200	12	21.6	13.8	13	13.8	13.2	13.4
20	500	4	77.2	70.2	70.4	70	70.4	67.6
20	500	7	48.6	41.8	42.4	42.2	42.2	41.8
20	500	8	43.6	38	37.6	37.8	38	38
20	500	12	33.2	27	27.2	27.2	27.2	27.2
20	1000	4	143.6	135.8	135.8	136.2	136	134.4
20	1000	7	86.4	80.4	80.6	80.6	80.6	80.4
20	1000	8	76.8	70.6	71	70.4	70.4	70.6
20	1000	12	54.4	48.8	49.8	49.2	49.2	48.8
30	50	4	29.8	10	9.8	9.4	8	8
30	50	7	29	8.4	7.4	7.8	5.8	4.8
30	50	8	29	8.2	7.8	7.6	5.2	4.2
30	50	12	29	8	7.4	6.8	5.2	3.4
30	100	4	33.4	16.8	16.6	16.2	15.6	13.8
30	100	7	29.8	12.2	12	11.6	11.6	9.4
30	100	8	29.6	12	11.4	11.8	10.8	9.8
30	100	12	29	10.8	10.4	10.2	9.8	8.2
30	200	4	43.8	30.4	31	31.6	31.2	28
30	200	7	33.8	20	19.8	19.4	20.2	20
30	200	8	32.2	18.6	18.8	19	19.4	19.8
30	200	12	29.8	17	16.6	16.2	16.2	16.2
30	500	4	80.4	69.6	69.6	69.6	69.4	69.2
30	500	7	53.4	42.4	42	42.8	42.6	42.8
30	500	8	49	38.4	38.6	38.6	38.2	37.8
30	500	12	39.2	29	28.6	28.6	28.6	28.8
30	1000	4	144	133.6	133.6	133.6	133.8	133.6
30	1000	7	89.6	79.8	79.8	79.8	80	79.4
30	1000	8	80.4	71.4	70.6	71	70.6	70.6
30	1000	12	59.4	50.8	50.6	50.8	50.8	50.8

Table 1 summarizes the results of instance with small number of ports, i.e. $P \in \{5, 10\}$. For each given P, N, H, we work the value UB, the average $Numofstack$ when K selects different values. Similarly, the UB is used to test the rationality and the result is used to compare when $K = 0$. Most of instances provide the evidence that it works well when K isn't equal to zero comparing the condition where K is equal to zero.

Table 2 summarizes that the results of the instances with large P, 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.

Table 3: average results of the instances with heuristic algorithm

K	0	10	20	50	100
$Numofstack$	97.6854	97.61042	97.4875	97.15	96.77708

Table 3 gives the average results when the K has different values. It isn't difficult to conclude that the value of $Numofstack$ becomes smaller if several rehandle is allowed and the more rehandle, the less stacks used. That's to say, our problem based on zero rehandle is meaningful and our algorithm is practical.

In fact, we have give another related algorithm named *P&H* algorithm to get the stowage planning. We give some restrictions on the height and nearport of alternative stacks when choosing one stack for each container. The concrete content of "P&H" method isn't described in this paper but the computational results will be listed in the following tables. The parameters in *P&H* values between 0 and 1. The former parameter represents the dispersion degree between the loading container's destination port and alternative stack's nearport and their values are equal if the value of parameter is 1. The latter parameter means the ratio of stack's height and the limited height H.

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

P&H	0	10	20	50	100
0.75&0.75	97.725	97.6375	97.49583	97.12708	96.72083
0.75&0.80	97.725	97.6375	97.49583	97.12708	96.72083
0.80&0.75	97.69583	97.61667	97.47083	97.11667	96.74583
0.80&0.80	97.69583	97.61667	97.47083	97.11667	96.74583

Table 4 shows the result with "P&H" method, which is firstly designed to improve the original algorithm and eventually it is used to test our original algorithm as reference. Both the two parameters of the method are selected form {0.75,0.8}. We can draw a conclusion that the first parameter has a significant influence on the result while the second parameter makes no difference. I suppose that the 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* method works better when the value of K is large. So maybe we can use the better one after working out the result when K is certain.

We also use some pictures to make our result clear and the pictures are made by Excel.

As what is shown in Table 3, Figure 1 indicates visually the relationship between used stacks and allowed rehandle. It's not hard to draw the conclusion that the more allowed rehandle, the less used stacks. So it intuitively reflect the meaning of this research topic and it show the feasibility of our algorithm. Comparing with zero rehandle, some rehandle are allowed to reduce the partial stacks and improve the utilization of space.

Figure 2 shows us the relationship between containers and used stacks. We can conclude that the more containers are, the more stacks will be used, which is undoubted in practice.

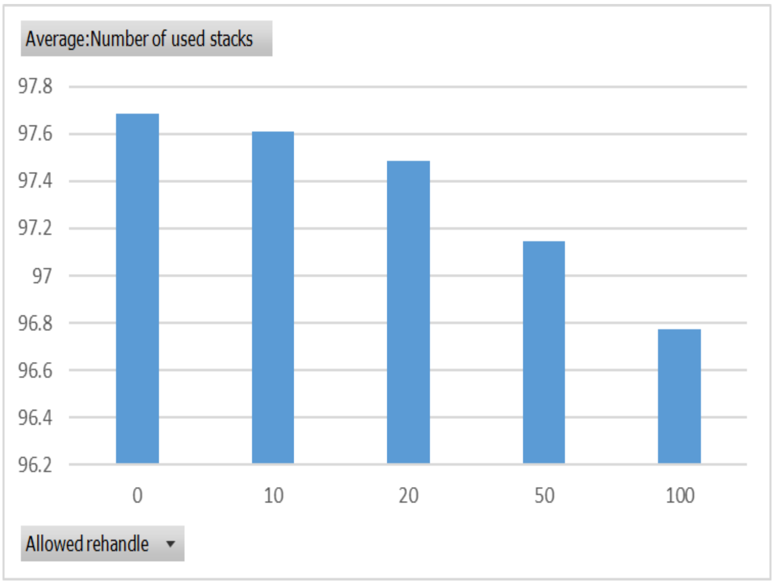


Figure 1: Relationship between allowed rehandle and used stacks

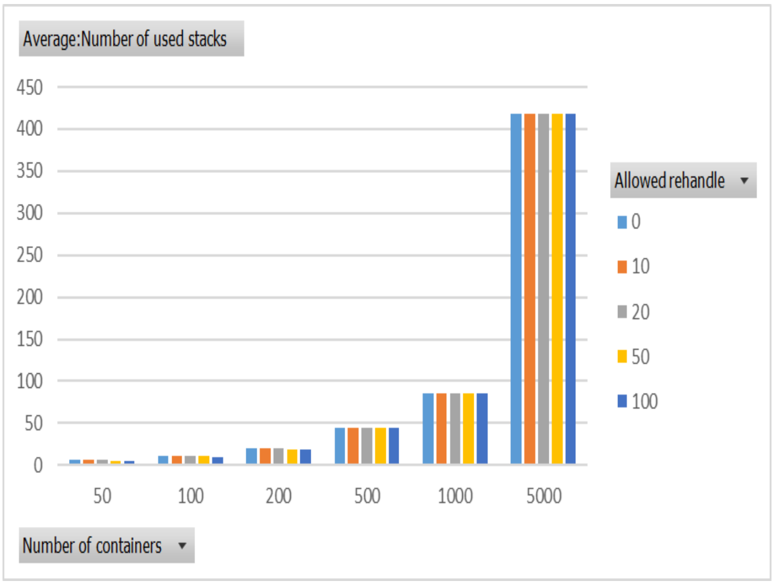


Figure 2: Relationship between containers and used stacks

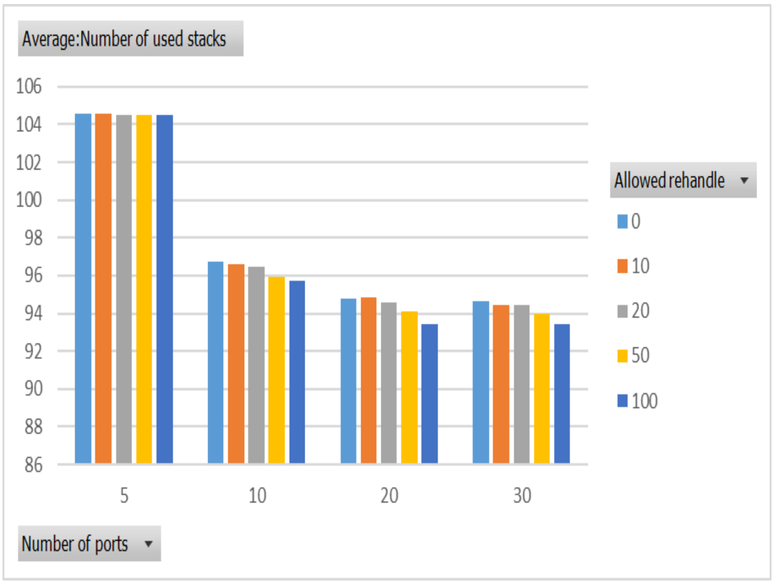


Figure 3: Relationship between ports and used stacks

Figure 3 provides us the relationship between ports and used stack. In general, the more ports are among the shipping, the more stacks will be used. Another related factor to be taken into consideration is the sparse distribution situation of containers in each port.

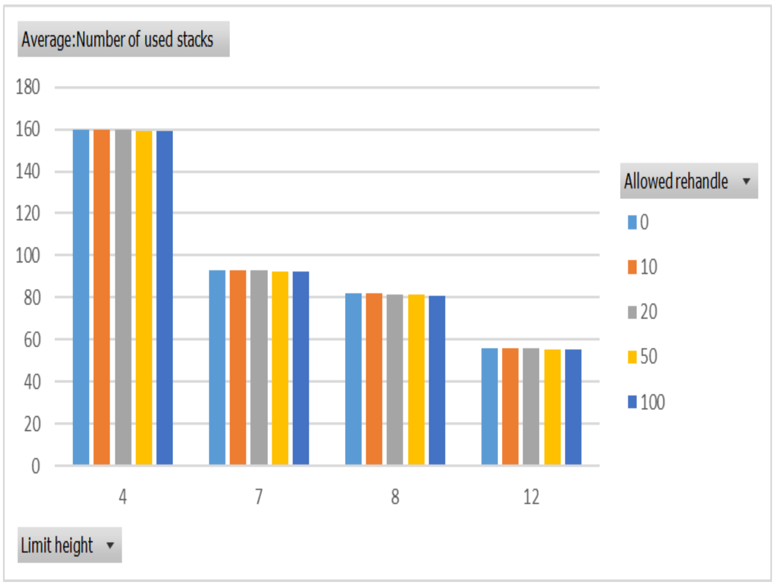
height.png 

Figure 4: Relationship between limit height and used stacks

Figure 4 shows us that the greater the number of limit height is, the less stacks will be used, which is definitely right.

Figure 5 shows us that the comprehensive impact on the result we work out for all the basic factors considered in this paper. And we can choose the factor we want to analyze and it is convenient for us to work out how different factor influences our final result. For the convenience of observation, we only give the result with filtered numbers of each factor.

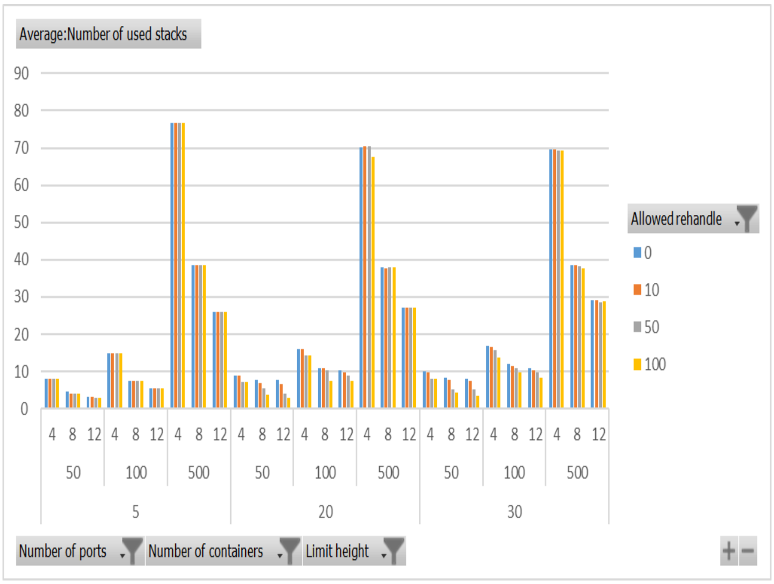


Figure 5: Relationship between comprehensive factors and used stacks

6. Conclusion

The stowage stack minimization problem with K-rehandle constraint (SSM-KR deals with accommodating give containers throughout a voyage by fewest stacks on a containership subject to K rehandles. In this paper, we analyze the structure of this combinatorial optimization problem and its relationship with the circle graph. A heuristic approach is proposed to construct near-optimal solutions in a very short computational time. The experimental results show that our heuristic approaches generate very promising solutions on a variety of instances. Comparing with the stowage stack minimum rehandle problem with zero constraint, our problem with K rehandle constraint acts as a buffer and offers great flexibility for the actual problem. The results shows the problem we put forward is of practical significance and it can improve the utilization ratio of vessels with certain flexibility. We use 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 the unnecessary spending in the case of meeting other constraints. The novelty and practicality show the importance of this paper and it can give a reference to the future study.

References

References

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

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.

Botter, R. & Brinati, M. (1991). Stowage container planning: A model for getting an optimal solution.

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

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.

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

Kammarti, R., Ayachi, I., Ksouri, M., & Borne, P. (2013). Evolutionary approach for the containers bin-packing problem. *arXiv preprint arXiv:1306.0442*.

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

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.

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

Webster, W. C. & 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.

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.