creative commons

C O M M O N S D E E D

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

공학석사 학위논문

Container Packing Problem with Guillotine Cutting and Complete-Shipment Constraints

길로틴 절단 및 완전 선적 제약을 고려한
컨테이너 패킹 문제

2016 년 2 월

서울대학교 대학원
산업공학과
정 민 철

Abstract

Container Packing Problem with Guillotine Cutting and Complete-Shipment Constraints

Mincheol Jeong
Industrial Engineering
The Graduate School
Seoul National University

This paper presents a tree search algorithm for the three-dimensional container packing problem (3D-CPP). There are many practical requirements for the 3D-CPP, and this paper considers the orientation, guillotine cutting, and complete-shipment constraints. A wall-building approach and the tabu search algorithm are used to maximize the volume utilization of the container. The famous Bischoff and Ratcliff test data from 1995 are used for testing the algorithm. This algorithm can quickly find an appropriate container packing plan with high volume utilization. Furthermore, it can offer a packing pattern that satisfies the complete-shipment constraint. It is easy and intuitive for staff to understand and can be quickly implemented.

Keywords : Container packing problem, Guillotine cutting pattern, Complete-shipment, Wall-building approach, Tree search, Tabu search

Student Number : 2014-20631

Contents

List of Tables

List of Figures

Chapter 1. Introduction

As transportation and communication technologies have become more numerous and the world trade becomes increasingly globalized, logistics has become a key factor for companies seeking to maximize their profits. There are many transportation modes, and a container is one of the most effective, convenient, and relatively economical modes for use in maritime trade. Because of its importance, many researchers are carrying out studies on containers, such as the development of a foldable container and an operation system for empty containers.

Many types of containers are used for oceanic and land transportation. The dry container is the standard for loading environment-insensitive cargos such as raw materials and clothes. The reefer container is used for temperature-sensitive cargos. Temperature can be controlled inside the reefer container, so it is suitable for loading food. In addition to these containers, there are many varieties, such as the open top and the flat rack container for specific cargo. Typical containers are 20ft- or 40ft-long. Most have the same 8ft width and 8.6ft height.

The three-dimensional container packing problem (3D-CPP), also known as a *container loading problem*, is the subject of important practical and academic research. In the real world, managers must determine the best way to load cargo into a specified number of containers. Most of managers do not depend on a scientific decision maker but rather on their experience and luck, which causes a waste of money for their companies.

Research on the container packing problem has led to the development of various container packing algorithms and systems. These algorithms and systems can offer efficient and effective container packing plans that companies can use to increase their profits.

The 3D-CPP is also an important subject in the literature of operations research (OR). It can be considered a three-dimensional extension of the cutting stock problem. The objective of the cutting stock problem is to minimize loss by partitioning stock adequately. Similarly, the objective of the 3D-CPP is to minimize loss (i.e. to maximize the volume utilization). In the case of the multiple container packing problem, the objective is to minimize the cost or the number of used containers. Because of the complexity of real-world situations and many practical constraints, container packing problems as well as cutting stock problems are valuable research areas in the OR literature.

Variations of the 3D-CPP can be categorized in accordance with specific characteristics. Wäscher et al. (2007) proposed a typology and an SLOPP (single large-object placement problem); the typology is considered in this thesis. There are two sets of elements of a 3D-CPP: a set of containers (input) and a set of boxes (output). All of the 3D-CPPs can be categorized as either an input minimization problem or output maximization problem, or they can be considered a 3D bin packing problem or 3D knapsack problem, respectively. The SLOPP belongs to the output maximization category.

According to the typology of Wäscher et al. (2007), the difference between the SKP (single knapsack problem) and the SLOPP is the degree of heterogeneity of box types. A box type is defined by its three dimensions; that is, the size of a box determines the box type. When one type of box is considered, it is described as *homogeneous*. If there are only a few types with many boxes per type, then the boxes constitute a weakly heterogeneous box set. The SLOPP assumes a set of weakly heterogeneous boxes; however a variety of box types with a relatively small number of boxes per type is a strongly heterogeneous box set. The SKP assumes a set of strongly heterogeneous boxes. This paper is based on a weakly heterogeneous box set: an SLOPP.

The 3D-CPP is difficult to solve because of its complexity conferred by the three-dimensional characteristics. Three basic geometric conditions must be satisfied for some box arrangements to be feasible:

- All boxes must be loaded within a container.

- No boxes can overlap each other.

- All boxes must be parallel with the face walls of a container.

In addition to these constraints, there are many practical constraints. Among them, three constraints are imposed in this paper:

Orientation constraint. If three dimensions of a box can be placed in a vertical orientation, then a box can lie six ways, as in Figure 1. In reality, however, some vertical orientations are prohibited. We prohibit up to two vertical orientations for each box type.

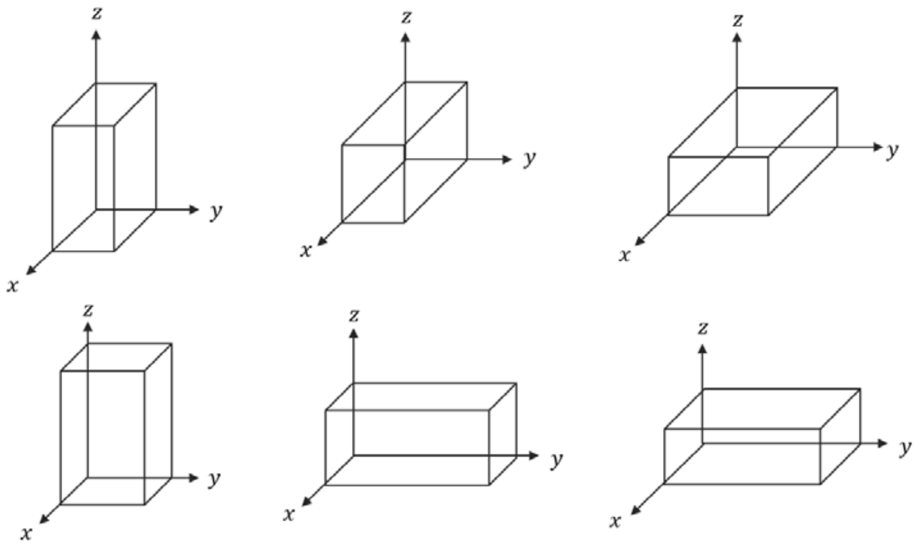

Figure 1 Six possible orientations of a box

Guillotine cutting constraint. In multi-dimensional cutting stock problems, staff has trouble implementing complex cutting patterns. These patterns are not practical. A guillotine cutting pattern can be described intuitively and used to pack items easily. It is derived from the 2D cutting stock problem. A pattern is said to be *guillotineable* if it can be obtained by a series of simple guillotine cuts in parallel to the edge of the stock. In this thesis, if the top view of a 3D loading pattern is guillotineable, the pattern is deemed consistent with a guillotine cutting pattern. Figure 2 shows some examples of guillotine patterns and non-guillotine patterns.

Complete-shipment constraint. In the output maximization problem, items may not be loaded within a single container and some items may be inevitably left behind. However, if any part of a subset is packed, then all of the other boxes of the subset also should be packed within the same container. For example, if a set of kitchen furniture consists of a sink, a cook stove, a range

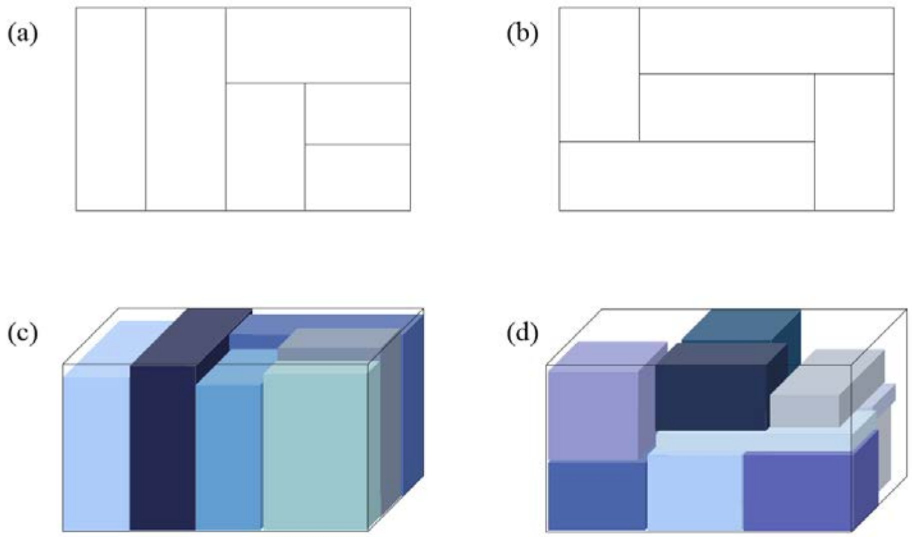

Figure 2 Example of cutting patterns: (a), (c) guillotineable; (b), (d) nonguillotineable

hood, several cabinets, a dining table, and four dining chairs, it is efficient to load all these together within the same container. This constraint can be applied to various situations:

- Specific types of box should be loaded together.
- The number of packed boxes of a specific type should be a multiple of a given lot size.

A group consisting of several box types (e.g. two boxes of type 1, a box of type 2, and a box of type 3) must be packed together. Whereas the orientation constraint is considered in most of the relevant papers, few researchers have consider the guillotine cutting and complete-shipment constraints in spite of their practical importance.

In this thesis, the orientation, guillotine cutting, and complete-shipment constraints are considered with heuristic methods. A wall-building approach and a tree search algorithm are used for satisfying the guillotine cutting constraint and a tabu search is used to increase the container volume utilized.

This paper is organized as follows. Chapter 2 provides a literature review on the 3D-CPP and relevant constraints and methods. Chapter 3 includes the specific problem definition and description as well as the explanation of used methods. In Chapter 4, a container packing algorithm named *HAGC* (heuristic algorithm with the guillotine cutting and complete-shipment constraints), based on a tree search and a tabu search, is presented. Chapter 5 is dedicated to computational experiments, and Chapter 6 summarizes this thesis and presents some perspectives for future research.

Chapter 2. Literature review

The 3D-CPP is a three-dimensional extension of the cutting stock problem, which is a well-known NP-hard problem in the OR literature. Since Gilmore and Gomory (1965) first addressed problems more complex than those involving 2D cutting stock cases, the 3D-CPP has been studied actively and many papers and algorithms have been presented for solving the problem.

Three papers give particularly valuable insights on research of the 3D-CPP. Bischoff and Ratcliff (1995) proposed some practical requirements for the 3D-CPP, such as load stability and shipment priorities constraints. This paper also offered 700 instances of test data for the SLOPP and these data are also adopted in the research presented in this thesis. Wäscher et al. (2007) presented up-to-date typology on cutting and packing problems. Bortfeldt and Wäscher (2013) presented a state-of-the-art review paper classifying the problems in accordance with the typology of Wäscher et al. (2007) and practical constraints. The authors analyzed and reviewed 163 papers published between 1980 and 2011.

According to Bortfeldt and Wäscher (2013), 96 papers (58.9%) dealt with the output maximization problem, and among the total reviewed, 37 papers (22.7%) addressed the SLOPP. Davies and Bischoff (1999) dealt with an SLOPP and an SKP by considering weight distribution. The authors developed a new container loading heuristic with post-processing approaches to distribute cargo weight evenly. Eley (2002) considered heterogeneous single and multiple container packing problems. The author presented a block-building approach in which a block consists of the same identically oriented items. A tree search was also used and some conditions, such as load stability, were considered. Ren et al. (2011) addressed an SLOPP with the shipment priority constraint. Their algorithm is also based on a block-building approach and a tree search. Moon and Nguyen (2014) presented an MIP (mixed integer programming) formulation and a hybrid genetic algorithm for solving an SLOPP. Their paper also considered weight limit and distribution constraints.

A few papers dealt with guillotine cutting and complete-shipment constraints. Amossen and Pisinger (2010) presented a generalized constructive algorithm for a multi-dimensional bin packing problem with the guillotine cutting constraint. In the paper, they assumed that the boxes cannot be rotated and a constraint programming method was used. Liu et al. (2014) used a wall-building approach and a tree search algorithm to satisfy the guillotine cutting condition. The algorithm of Liu et al. (2014) is based on IP (integer programming) models of one-dimensional knapsack problems. The only paper considering the complete-shipment constraint was presented by Eley (2003), which dealt with multiple container packing problems. The Eley's paper presented a bottleneck assignment approach for minimizing the number of required containers. Furthermore, it considered two special practical constraints, the complete-shipment constraint and the separation constraint in which two boxes of differing type must not be stowed in the same container.

In many relevant papers, specific box arrangement approaches were used. A wall-building approach and a block-building approach are two representative arrangements. Only a few papers, such as George and Robinson (1980), Bortfeldt and Gehring (2001), Pisinger (2002), and Liu et al. (2014), used a wall-building approach in which a container is filled with walls made of boxes. However, many papers explain use of a block-building approach, including Eley (2002), Bortfeldt et al. (2003), Fanslau and Bortfeldt (2010), Ren et al.

(2011), and others. In a block-building case, a container is filled with cuboid blocks that consist of a single-type of box.

These two approaches have their particular advantages. In this thesis, a wall-building approach is used to find a simple and intuitive loading pattern.

Most authors of 3D-CPP papers proposed their own heuristic algorithms despite the importance of the cutting stock problem in the OR literature. One of the reasons is that the 3D-CPP has many realistic constraints that require use of complicated mathematical equations. Heuristic algorithms cannot guarantee the optimality of a solution, but many offer a good solution in a reasonable time. Heuristic methods for the 3D-CPP can be divided into the tree search method and other types. Some adopted a block-building approach, such as Eley (2002), Fanslau and Bortfeldt (2010), and Ren et al. (2011), which used a tree search algorithm. Liu et al. (2014) offered the only paper featuring a binary tree search algorithm.

Many researchers used metaheuristic algorithms entirely or partially to solve complex optimization problems or increase the performance of their whole algorithms; that is, they search neighborhood and escape a local optimum. A genetic algorithm and a tabu search algorithm are two of the most popular metaheuristic methods. Gonçalves and Resende (2012) presented a multi-population, biased, random-key genetic algorithm for the single container packing problem. Bortfeldt and Gehring (2001) and Feng et al. (2015) proposed some hybrid genetic algorithms. A tabu search is so simple that many researchers, such as Bortfeldt et al. (2003), Crainic et al. (2009), and Liu et al. (2011), adopted this method. In this thesis, a tabu search is more suitable than a genetic algorithm for handling and encoding a solution.

The problem considered in this thesis can be thought of as the 2D cutting stock problem because using a strip as a unit of a container packing prevents concern over the height orientation in the packing process where a strip is a box tower. Bortfeldt and Jungmann (2012) also approached a 3D-CPP as if it were a 2D cutting stock problem by using strip packing. Furthermore, many papers on true 2D cutting stock problems, such as Alvarez-Valdes et al. (2002), de Armas et al. (2012), Dolatabadi et al. (2012), Clautiaux et al. (2013), and Russo et al. (2013), offered some valuable and applicable ideas.

This study can contribute to the relevant literature. The complete-shipment constraint, which is quite practical and plausible but rarely addressed, is considered. This may motivate many researchers to do related studies and consider characteristics and correlation of boxes. Moreover, by using a tabu search method, computational time can be reduced effectively compared to other algorithms. But most importantly, the proposed algorithm can offer simple, easy, intuitive, and worker-friendly container loading plans. What is more, the algorithm can be made practical and realistic for loading patterns because the height limit of strips can be adjusted such that vertically stacking many boxes is restricted. The contributions of this study and all relevant references are summarized in Table 1.

Table 1 Summary of the contributions of relevant papers

Author(s)	Problem type	Constraints			Methods	
	Output maximization	Orientation	Guillotine cutting	Complete-shipment	Heuristic	Metaheuristic
Davies and Bischoff (1999)	√	√			√	
Bortfeldt and Gehring (2001)	√	√			√	√
Alvarez-ValdeHs et al. (2002)	√		√		√	√
Eley (2002)	√	√			√	
Pisinger (2002)	√		√		√	
Bortfeldt et al. (2003)	√	√				√
Eley (2003)				√	√	
Crainic et al. (2009)						√
Amossen and Pisinger (2010)			√			
Fanslau and Bortfeldt (2010)	√	√	√		√	
Liu et al. (2011)	√	√			√	√
Ren et al. (2011)	√	√			√	
Bortfeldt and Jungmann (2012)	√	√	√		√	
Goncalves and Resende (2012)	√	√				√
Liu et al. (2014)	√	√	√		√	
Moon and Nguyen (2014)	√	√			√	√
Feng et al. (2015)					√	√
This study	√	√	√	√	√	√

Chapter 3. Problem description

3.1. Basic assumption

The 3D coordinate system is used, and *x*, *y*, and *z* axes of the first octant of the 3D space represent the length, width, and height of a container as shown in Figure 3. As can be seen, the origin corresponds to the rear-left-bottom corner of a container. All loaded boxes are always laid somewhere in the first octant parallel to the axes.

The dimensions of a container are denoted by *L*, *W*, and *H* which represent the length, width, and height, respectively. The test data from Bischoff and Ratcliff (1995), which is used in this thesis, specify a 20ft container with $L = 587\text{cm}, W = 233\text{cm}, \text{and } H = 220\text{cm}$. In reality, a 40ft container is also

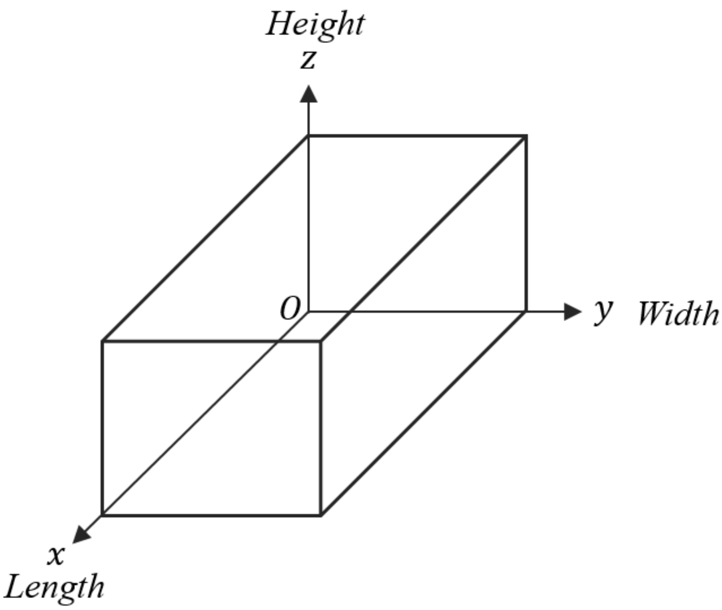

Figure 3 3D coordinate system

commonly used, and the width and height of the 40ft container are the same as the 20ft container, but the length of the container is about 1200cm.

Let $B = \{B_1, B_2, \ldots, B_n\}$ be the box set that contains n types of boxes. Box type i has following specifications:

$$(l_i, \alpha_i, w_i, \beta_i, h_i, \gamma_i, b_i)$$

for all i. l_i, w_i, and h_i are the length, width, and height of box type i. For convenience, in the Bischoff and Ratcliff (1995) test data, it was assumed that $l_i > w_i > h_i$. b_i is the number of available boxes of type i. α_i, β_i, and γ_i are the binary parameters with respect to the possibility of a vertical orientation:

- $\alpha_i = 1$ if the x-axis direction (length) of box type i can be in the vertical orientation, 0 if it cannot;
- $\beta_i = 1$ if the y-axis direction (width) of box type i can be in the vertical orientation, 0 if it cannot;
- $\gamma_i = 1$ if the z-axis direction (height) of box type i can be in the vertical orientation, 0 if it cannot.

A cuboid, e.g. a box or a container, is referred to as *oriented* if the vertical

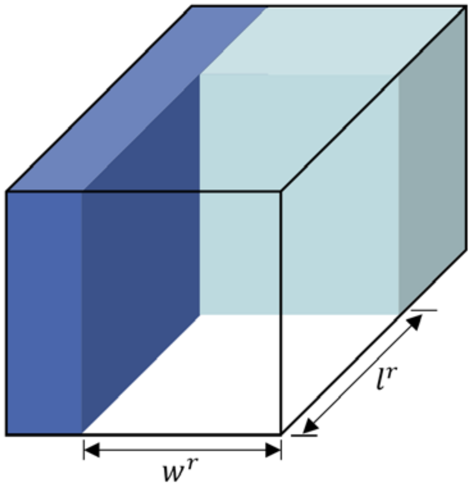

Figure 4 Residual space

position of the cuboid is fixed. The dimensions of an oriented cuboid are denoted as $mx, my,$ and $mz,$ respectively. In this paper, superscripts b, s, and w are used to represent a box, strip, and wall, respectively.

The residual space is an empty cuboid in a container and denoted by l^r and w^r as shown in Figure 4. The height of the residual space is not considered because it is always equivalent to the height of a container.

Solutions, i.e. container packing patterns, and groups of the complete-shipment are simply denoted as n-dimensional nonnegative integer vectors in which each element represents the number of packed or required boxes of each type. A group of the complete-shipment is denoted as:

$$CS = (cs_1, \dots, cs_n)$$

where cs_i is the number of box type i in the group.

3.2. Wall-building approach

The basic packing unit is a *strip* which can be thought of as a tower made up of several types of oriented boxes. When building a strip, an initial oriented box is selected and an envelope cuboid is formed. The height of the cuboid is always equal to the height of the container or some height limit, and the length and the width are equal to *mx* and *my* of the initial box. In this situation, the heights of every strip do not need to be considered, so the problem becomes the 2D knapsack problem.

Selecting an initial box type is really important because the choice determines the length and the width of a strip, the dimensions of the strip determine the depth of a wall, and the depth of the wall affects the performance of the algorithm. George and Robinson (1980) presented a ranking rule for selecting a box: among remaining boxes, select the box with the largest size of the smallest dimension because it may be difficult to pack later in the procedure. In this thesis, the smallest dimension that can be an edge of the bottom is considered the standard. In the BR test data, if γ_i is 1, then h_i must be in the vertical orientation. In this case, the smallest dimension is w_i. If α_i or β_i is 1, then h_i has the smallest dimension.

Once the first box type is selected, *mx* and *my* of the box become the length and the width of an envelope cuboid. To apply the concept of the knapsack greedy heuristic algorithm, mz is determined by the shortest possible dimension of the box. Then box candidates for the strip are sorted. The orientation of each box is determined by the following definitions:

$$H_\alpha(B_i, mx_j^s, my_j^s) = \begin{cases} l_i & \text{if} \max\{w_i, h_i\} \le mx_j^s, \min\{w_i, h_i\} \le my_j^s, \alpha_i = 1 \\ +\infty & \text{otherwise} \end{cases},$$

$$H_\beta(B_i, mx_j^s, my_j^s) = \begin{cases} w_i & \text{if} \max\{l_i, h_i\} \le mx_j^s, \min\{l_i, h_i\} \le my_j^s, \beta_i = 1 \\ +\infty & \text{otherwise} \end{cases},$$

$$H_\gamma(B_i, mx_j^s, my_j^s) = \begin{cases} h_i & \text{if} \max\{l_i, w_i\} \le mx_j^s, \min\{l_i, w_i\} \le my_j^s, \gamma_i = 1 \\ +\infty & \text{otherwise} \end{cases}.$$

These formulas confirm the available orientations of box type i. If the bottom area of one possible orientation is larger than that of the initial box, the height of this orientation is defined as $+\infty$. Then, the height of the final orientation is determined as

$$H(B_i, mx_j^s, my_j^s) = \min\{H_\alpha(B_i, mx_j^s, my_j^s), H_\beta(B_i, mx_j^s, my_j^s), H_\gamma(B_i, mx_j^s, my_j^s)\}.$$

That is, the shortest possible dimension becomes the height of all boxes of the selected type. This may increase the volume utilization so the strip includes more boxes, and it may lower the center of gravity and strengthen the stability of the load. Using boxes of the selected type, the strip is built by maximizing

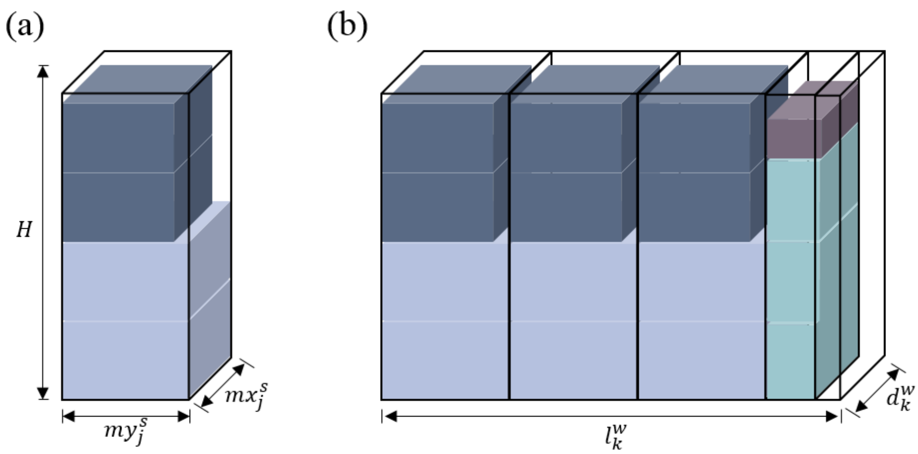

Figure 5 Definition of (a) strip and (b) wall

the volume utilization with the help of the tabu search. As shown in Figure 5 (a), the j-th strip has mx_j^s and my_j^s for the length and the width, respectively.

When strips are loaded within the container, some strips form a wall like that shown in Figure 5 (b). Once initially formed, the strip is located on the rear-left-bottom corner of the residual space and an envelope cuboid is formed. One of the dimensions of the strip becomes the depth of a wall. The length (x-axis) or the width (y-axis) and the height of the cuboid are defined as the length or the width of the residual space and the height of the container. Then available box types that satisfy $\max\{mx, my\} \leq \max\{d_k^w, l_k^w\}$ and $\min\{mx, my\} \leq \min\{d_k^w, l_k^w\}$ for all possible box orientations are selected and available strips are built.

When building available strips, the standard value (sv) of the volume utilization is introduced. If the volume utilization of a formed strip does not exceed sv, the strip is discarded. The envelope cuboid is then filled with the available strips, and the wall is built. A tabu search algorithm is also used to maximize the volume utilization. A container is then filled through the successive placing of walls.

3.3. Tree search algorithm

When a wall is built, the initial strip can be loaded in one of two ways – along the x or along the y axis – and the wall can be also formed in the direction of either the x-axis or y-axis as shown in Figure 6. It is hard to explore all nodes, which would be computationally too expensive. To overcome this difficulty, a tree search algorithm is used to find the best set of walls in terms of the volume utilization. This is a greedy and myopic heuristic method.

When branching a parent node, up to four children nodes can be made: Two nodes are derived from the initial strip $(mx \times my)$ and the other two are from the strip $(my \times mx)$. Among the four options, the node having the

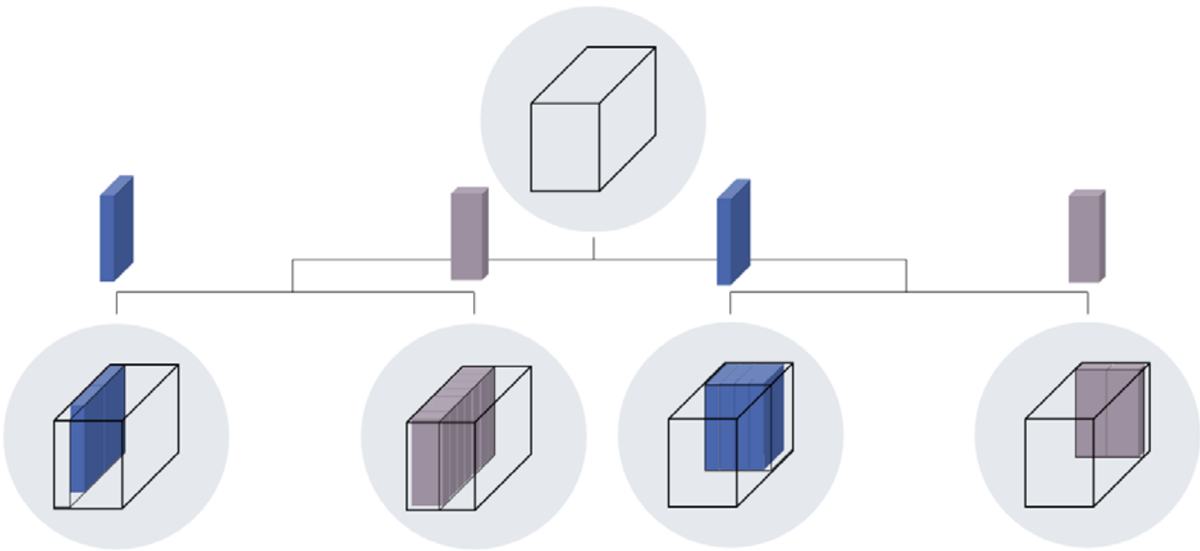

Figure 6 Tree search algorithm

highest volume utilization is selected and the other nodes are pruned. This can reduce the computation time.

All leaf nodes correspond to feasible complete container packing plans. Among them, the packing pattern of the highest volume utilization is selected as the output solution of the algorithm. The volume utilization of the pattern is defined as

$$vu(x) = \frac{\sum_{i=1}^{n} l_i \times w_i \times h_i \times x_i}{L \times W \times H}$$

where $x = (x_1, \ldots, x_n)$ is a container packing solution and each element refers to the number of packed boxes of each type.

3.4. Tabu search algorithm

An important role of the heuristic algorithm is as a means to find a good, feasible solution quickly. Greedy heuristic algorithms are used to find good, feasible strips, walls, and container packing plans. These algorithms, however, can lead to local optimal solutions, so neighborhood searches are needed to escape local optima. The tabu search is used for this purpose.

The tabu search modifies an incumbent into another solution in the neighborhood even if its solution value is worse than the value of an incumbent. The algorithm may result in cycling and so the tabu list is adopted to avoid such cycling. The specific number of recent solutions or moves is placed on the tabu list so that each is excluded during later iterations. Wolsey (1998) described a basic version of the tabu search algorithm.

Algorithm 1 Tabu search algorithm

```
TabuSearch()
Initialize an empty tabu list and a solution s
While the stopping criterion is not satisfied
        Choose a subset of non-tabu solutions
        Let s′ be the best solution of the subset
        Replace s by s′ and update the tabu list
Return the best solution s* found
```

The tabu search contains some important parameters, such as the tabu tenure. The tabu list has *t* of most recent solutions, and the number *t* is called the *tabu tenure*. The tenure is determined empirically. The iteration is also important. The number of iterations determines the stopping criterion. An adequate definition of the iteration can make good and quickly obtained results, so it is determined empirically.

3.4.1. Solution representation and initialization

The tabu search is used for maximizing the volume utilization of strips and walls. A strip is encoded in the form of $s^j = (s_1^j, s_2^j, \dots, s_n^j)$ where each element corresponds to the number of packed boxes of each type. The length of the encoded solution is always *n*. To obtain an initial solution for a strip, one stacks the initial box as high as possible. For example, if box type 1 is the largest size of the smallest dimension, one of the boxes of type 1 is selected as the initial box. Then the initial solution is as follows:

$$\left(\min\left\{\left\lfloor H/mz_1^b \right\rfloor, b_1\right\}, 0, \dots, 0\right).$$

A wall is encoded in the form of $y^k = (y_1^k, y_2^k, \dots)$ where y_j^k is the number of packed strips of type j. The number of available strip types depends on the initial strip, so the length of the solution vector varies. An initial solution for building a wall is also similar to the one for the strip. The wall that consists of only one type of strips, which is the same as the initial strip, used as the initial solution:

$$\left(\min\left\{\lfloor l_1^w / mx_1^s(my_1^s)\rfloor, \min_i\{b_i/z_{i1}\}\right\},\ 0,\ 0,\ \dots\right).$$

3.4.2. Objective function

The objective of the algorithm is to maximize the container volume utilization. The objective function for building the j-th strip is defined as:

$$f(s^j) = \frac{\sum_{i=1}^{n} l_i \times w_i \times h_i \times s_i^j}{mx_j^s \times my_j^s \times H}.$$

This objective function represents the volume utilization of the strip. Similarly, the objective function for building the k-th wall is defined as:

$$g(y^k) = \frac{\sum_j \sum_{i=1}^{n} l_i \times w_i \times h_i \times z_{ij} \times y_j^k}{l_k^w \times d_k^w \times H}$$

where z_{ij} is the number of boxes of type *i* included in strip *j*.

3.4.3. Definition of moves

In the tabu search, the term *move* means the modification of the incumbent in the neighborhood search. Designing moves is really important in the tabu search because they affect diversification in the search process such that welldesigned moves can lead to powerful local searches and a near optimal solution. At each iteration, some possible moves are selected and the moves modify the incumbent. Among adjusted solutions, the best solution created by a move is accepted even if it is worse than the incumbent.

The tabu search is used when building a strip and a wall. In these two cases, the same definition of the move is used: In the first type of move, corresponding to Figure 7 (a), two elements of the solution vector are selected. One is selected among positive elements and 1 is subtracted from this selected element. The other element is selected among those having spare boxes or strips and 1 is added to this selected element. In the second type of the move, corresponding

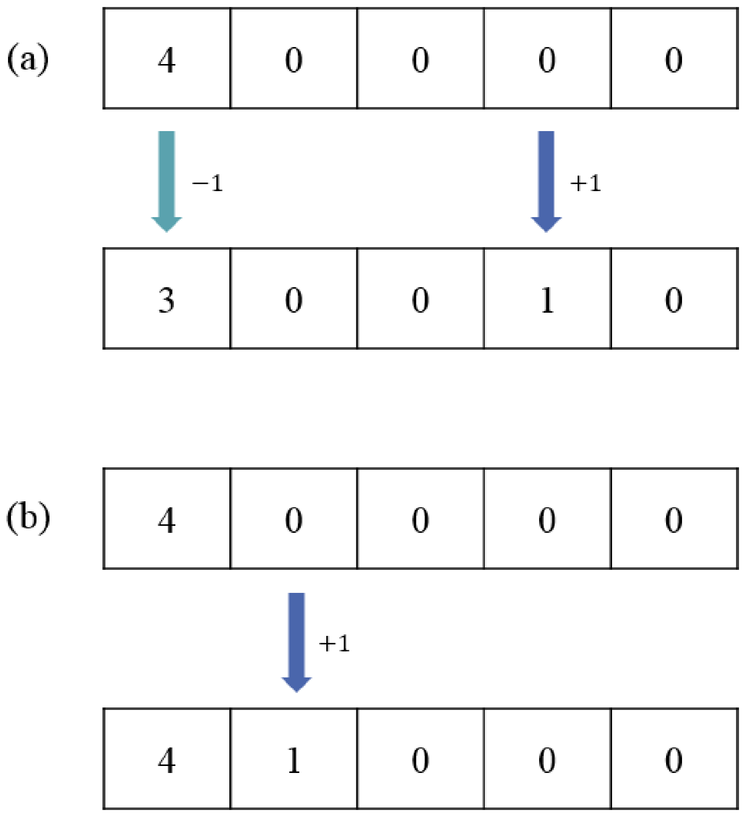

Figure 7 Examples of moves

to Figure 7 (b), just one element of the solution vector is selected. Among elements which have spare boxes or strips, an element is selected and 1 is added to the selected element.

Some moves may lead to an infeasible solution. In the strip case, a solution is feasible if $\sum_{i=1}^{n} mz_i^b \times x_i \leq H$. A solution for the wall case is feasible if $\sum_j mx_j^s(my_j^s) \times y_j \leq l_k^w$. The second type of move can lead to infeasible solutions with high probability. To guarantee the feasibility of modified solutions, the objective value of an infeasible solution is given as zero.

Chapter 4. The proposed algorithm

In this chapter, the heuristic algorithm named *HAGC (heuristic algorithm with guillotine cutting and complete-shipment constraints)* is presented. This is a hybrid algorithm composed of many small heuristic and metaheuristic parts.

4.1. Creation of an initial strip

First of all, an initial strip is needed to determine the depth of an envelope of a wall. An initial strip is made through the following algorithm.

Algorithm 2 Creation of an initial strip

CreateAnInitialStrip (B, l^r, w^r, H)

Initialize a strip $s \in \mathbb{Z}_+^n \cup \mathbf{0}$
Select boxes that can be loaded within the residual space
If there is no suitable box, **Return** $\emptyset$
Select a box that has the largest size of the smallest dimension among dimensions except for the vertical orientationas an initial box of s
Set the shortest dimension among possible dimensions as the vertical orientation of the initial box
Select boxes that can be supported completely by the initial box
Fix the orientation of each box
Build the strip with the boxes and the tabu search by maximizing $f(s)$
Return s

Some boxes that cannot be loaded within the residual space are deleted and an adequate box is selected as the initial box. The largest box of the smallest dimension is suitable for an initial box, because small or thin boxes may be loaded easily when the residual space is quite small. Then, the vertical direction of the box must be determined. To use the concept of the knapsack greedy heuristic algorithm, the largest available face is used as the bottom and the shortest edge is oriented vertically.

The determined mx and my become the length and the width of a strip. An envelope of a strip with $mx \times my \times H$ is formed and boxes fill this envelope cuboid; however, some boxes need to be thrown out to secure stability of the load. If all possible bottom areas of a box type cannot be supported completely by the initial box, then the box type is excluded from the strip's components. Then, the vertical orientation of each component box is set up. The rule is described in Section 3.2: The shortest dimension available is set in the vertical direction so volume utilization can be maximized.

The strip is built with component boxes. In this process, the tabu search, defined in Section 3.4, is used to search the neighborhood, escape local optima, and maximize the volume utilization of the strip. Finally, this algorithm produces the best strips found during the iterations.

4.2. Derivation of additional strips from the initial strip

Once an initial strip is formed, an envelope of a wall with a depth equal to one of the dimensions of the initial strip is defined. This envelope cuboid can be filled only with strips of the same type as the initial strip, but additional strips can be included in the envelope cuboid to increase diversification and maximize the volume utilization of the cuboid. Additional strips are made through the following algorithm.

Algorithm 3 Derivation of additional strips from the initial strip

DeriveStrips $(s, B, d_k^w, l_k^w, H, sv)$

Initialize a strip set S and add s to S
Select boxes that can be loaded within the envelope cuboid
For all of the boxes
    For all possible orientations of the box
        Initialize a strip $s' \in \mathbb{Z}_+^n \cup \mathbf{0}$
        Select the box as an initial box
        Set the shortest dimension among possible dimensions as the vertical orientation of the initial box
        Select boxes that can be supported completely by the initial box
        Build the strip with the boxes and the tabu search by maximizing $f(s')$
        If $f(s') > sv$ and the strip is not a duplicate
            Add s' into S
Return S

The procedure is similar to the strip-building algorithm. However, for a wall, all possible orientations of boxes are considered as an initial box to diversify components and increase the probability of maximizing the volume utilization. For each initial box with a specific orientation, a strip is built by using the tabu search. If the volume utilization of this strip exceeds the standard value (sv) and not already in strip set S, then the strip is included in strip set S. One box type can be oriented in various ways, so several strips can be derived from one box type. Once all of the box types are considered, the algorithm returns S.

4.3. Creation of walls

Figure 6 showed that each parent node can take up to four child nodes, and two cases – walls with x- and y-axis directions – are considered to simplify the algorithm. The algorithm for creating a wall along the *x*-axis is presented in the following algorithm:

Algorithm 4 Creation of a wall with an *x*-axis direction

CreateWallX (s, B, sv)

For orientations 1 and 2 of the initial strip s loaded within the residual space
    Make an envelope cuboid of a wall along the *x*-axis
    An available strip set $S =$ DeriveStrips $(s, B, d_k^w, l_k^w, H, sv)$
    Initialize $y^1 \in \mathbb{Z}_+^{|S|} \cup \mathbf{0}$ or $y^2 \in \mathbb{Z}_+^{|S|} \cup \mathbf{0}$
    Build the wall with the available strips and the tabu search by maximizing the volume utilization $g(y)$
Initialize a solution $x \in \mathbb{Z}_+^n \cup \mathbf{0}$
If $g(y^1) \geq g(y^2)$
    $x_i = \sum_j z_{ij} \times y_j^1$ for all i
Else $x_i = \sum_j z_{ij} \times y_j^2$ for all i
Return x

In the procedure, two building cases – orientations of $(mx \times my)$ and $(my \times mx)$ of the initial strip – are considered simultaneously. For each envelope cuboid, the available strip set is formed by using Algorithm 3, and the wall is filled with the strips by using the tabu search. Once two walls are built, the more suitable wall is selected based on a comparison of the two volume utilizations.

Once a wall is built, the selected $|S|$-dimensional vector should be converted into an n-dimensional solution vector because every wall vector has its own length and standardization is needed to reach an ultimate solution. x_i is the sum of z_{ij} multiplied by y_j for all j where z_{ij} is the number of boxes of type i included in strip j, defined in Section 3.4.2. Finally, the wall in the form of an n-dimensional vector is the output.

The algorithm for creating a wall with a y-axis direction is almost the same as the one for the x-axis direction:

Algorithm 5 Creation of a wall with a y-axis direction

CreateWallY (s, B, sv)

For orientations 1 and 2 of the initial strip s loaded within the residual space
    Make an envelope cuboid of a wall with a y-axis direction
    An available strip set $S =$ DeriveStrips $(s, B, d_k^w, l_k^w, H, sv)$
    Initialize $y^1 \in \mathbb{Z}_+^{|S|} \cup \mathbf{0}$ or $y^2 \in \mathbb{Z}_+^{|S|} \cup \mathbf{0}$
    Build the wall with the available strips and the tabu search by maximizing the volume utilization $g(y)$
Initialize a solution $x \in \mathbb{Z}_+^n \cup \mathbf{0}$
If $g(y^1) \geq g(y^2)$
    $x_i = \sum_j z_{ij} \times y_j^1$ for all i
Else $x_i = \sum_j z_{ij} \times y_j^2$ for all i
Return x

4.4. Satisfaction of the complete-shipment constraint

If the solution found from the previous algorithms does not satisfy the complete-shipment constraint, unnecessary boxes should be deleted and specific elements of the solution should be adjusted to a multiple of the complete-shipment rule. For example, for the found solution of $(30, 20, 14)$ and the complete-shipment rule $(2, 1, 1)$, $(28, 14, 14)$ is sufficient and $(2, 6, 0)$ is unnecessary and deleted form the found solution.

A sequence is used to satisfy the complete-shipment constraint. Divide x_i by cs_i for all i for which $cs_i > 0$; the least value is the multiplier m. To delete unnecessary boxes from the container, set $x_i = m \times cs_i$ for all i. Generated empty spaces can be filled with other boxes that are not elements of the complete-shipment group. To determine the boxes to fill the space, sort box types into descending order by box volume such that the largest box j that satisfies $l_i \geq l_j, w_i \geq w_j$, and $h_i \geq h_j$ is loaded within the empty space in place of deleted box i from the complete-shipment group.

4.5. Establishment of the entire algorithm

The HAGC consists of the partial algorithms 1 through 5. The overall algorithm is as follows:

Algorithm 6 Heuristic algorithm with guillotine cutting and complete-shipment constraints

HAGC (B, L, W, H, CS, ω)

Initialize a solution x and μ // μ as a temporary solution
Initialize a wall list WL and an orientation list of walls OL
Define $count \leftarrow 1$ // for increasing the multiplier
For $count \leq \rho$
    Initialize $l^r \leftarrow L, w^r \leftarrow W$, a temporary wall list TWL, and a temporary orientation list of walls TOL
    While there is a box that can be loaded within the residual space
        an initial strip $s \leftarrow$ CreateAnInitialStrip (B, l^r, w^r, H)
        an x-axis wall $x^1 \leftarrow$ CreateWallX (s, B, d_k^w, l_k^w, sv)
        a y-axis wall $x^2 \leftarrow$ CreateWallY (s, B, d_k^w, l_k^w, sv)
        If $f(x^1) \geq f(x^2)$ **Then**
            $\mu \leftarrow \mu + x^1$
            Add x^1 to TWL and 1 to TOL
        Else
            $\mu \leftarrow \mu + x^2$
            Add x^2 to TWL and 2 to TOL
        Update B, l^r, and w^r
    Adjust μ in accordance with CS
    If $vu(\mu) > vu(x)$, **Then** $x \leftarrow \mu$, $WL \leftarrow TWL, OL \leftarrow TOL$
    Reset all b_i and $b_i \leftarrow b_i - \omega \times count$
    $count \leftarrow count + 1$
Return x

To initiate the HAGC, $x = (x_1, \ldots, x_n)$, which represents a container packing plan and $\mu = (\mu_1, \ldots, \mu_n)$, which is a temporary solution for the complete-shipment constraint, are initialized. Each element is an integral nonnegative variable, meaning the number of packed boxes of each type. After a solution is found through the partial algorithms, the HAGC solution is adjusted to satisfy the complete-shipment constraint in accordance with Section 4.4. If the volume utilization of the adjusted solution is higher than that of x, then x is updated.

Some solutions feature zero for some i such that $cs_i > 0$, and in these cases, the multiplier also becomes zero. To prevent this situation, ω is introduced, and for all box types, ω boxes are deducted from each available box type. Then algorithm 6 is repeated from line 5 to line 20. After repetition ρ times, ρ solutions are achieved with $\rho - 1$ adjustments, and a solution with the highest volume utilization is selected. For example, if $(b_1, b_2, b_3) = (30, 25, 20)$, $\rho = 3$, and $\omega = 2$, three solutions are found from $(b_1, b_2, b_3) = (30, 25, 20), (28, 23, 18),$ and $(26, 21, 16)$.

Chapter 5. Computational experiments

The proposed algorithm, HAGC, was implemented in JAVA, and experiments were run on an Intel® Core™ i5-3570 CPU @ 3.40GHz processor with 8 GB RAM.

5.1. Test data and valuable resources

As mentioned in Chapter 2, Bischoff and Ratcliff (1995) proposed 700 test data (BR data) for the SLOPP, and many papers on the SLOPP have used these data for benchmarks. These data can be downloaded from http://people.brunel.ac.uk/~mastjjb/jeb/orlib/thpackinfo.html.

The BR data include seven cases, BR1 to BR7 and each case includes 100 instances. The differences among all cases represent the number of box types: 3, 5, 8, 10, 12, 15, and 20, respectively. Davies and Bischoff (1999) stipulated that BR1 to BR7 are weakly heterogeneous container packing problems. In all instances, the container is always assumed to be 20ft (i.e. $587 \times 233 \times 220\ \mathrm{cm}^3$). The BR data are composed as in Table 2, which shows one example of the BR1 data set. The length, width, and height of the table correspond to

Table 2 Example of the BR test data

Type	Length	Vert.	Width	Vert.	Height	Vert.	Quantity
1	108	0	76	0	30	1	40
2	110	0	43	1	25	1	33
3	92	1	81	1	55	1	39

$l_i, w_i,$ and h_i, and quantity corresponds to b_i. Vert. is the abbreviation of the vertical orientation from the data sets of Bischoff and Ratcliff (1995) and refers to the possibility of a vertical direction. If Vert. is 1, then this dimension can be in the vertical orientation. Each Vert. correspond to $\alpha_i, \beta_i,$ and γ_i, respectively.

JAVA is the most famous object-oriented programming (OOP) language. The OOP paradigm is based on the concept of objects, and a program is considered a set of objects. An instance casted from a class in a programming language corresponds to an object. One of the advantageous characteristics of the OPP is its reusability. Some classes made by other people can be used as a part of one's own program. We use a JAVA tabu search framework from http://www.coin-or.org/Ots/index.html.

The Computational Infrastructure for Operations Research (COIN-OR) Foundation, Inc., is a non-profit educational and scientific foundation for managing the COIN-OR project. Corporate members of the foundation include IBM and Maximal Software, among others, and a strategic partner is the INFORMS Computing Society. The COIN-OR project is an initiative to develop open-source software for the OR community. The project has developed and released open tabu search classes for JAVA to help users implement popular metaheuristic algorithms in well-defined, object-oriented designs.

Classes such as Solution, ObjectiveFunction, Move, MoveManager, TabuList, TabuSearch, etc. were used in the experiment. Some critical parameters were determined empirically and definitions and configurations of some classes are described in Section 3.4.

5.2. Test results without the complete-shipment constraint

Eley (2003) is the only research that has considered and published on the complete-shipment constraint, and the related paper dealt with a multiple container packing problem. So, the algorithm without the complete-shipment constraint is tested and compared to other relevant algorithms. In this case, line 17 of Algorithm 6 is omitted.

Figure 8 is an example of the container packing procedure and a solution for a test instance of BR1 of the BR data shown in Table 2. The figures represent top views of the container and each square refers to a strip; for example, ① × 7 describes a strip that consists of seven boxes of type 1. Other numbers in the

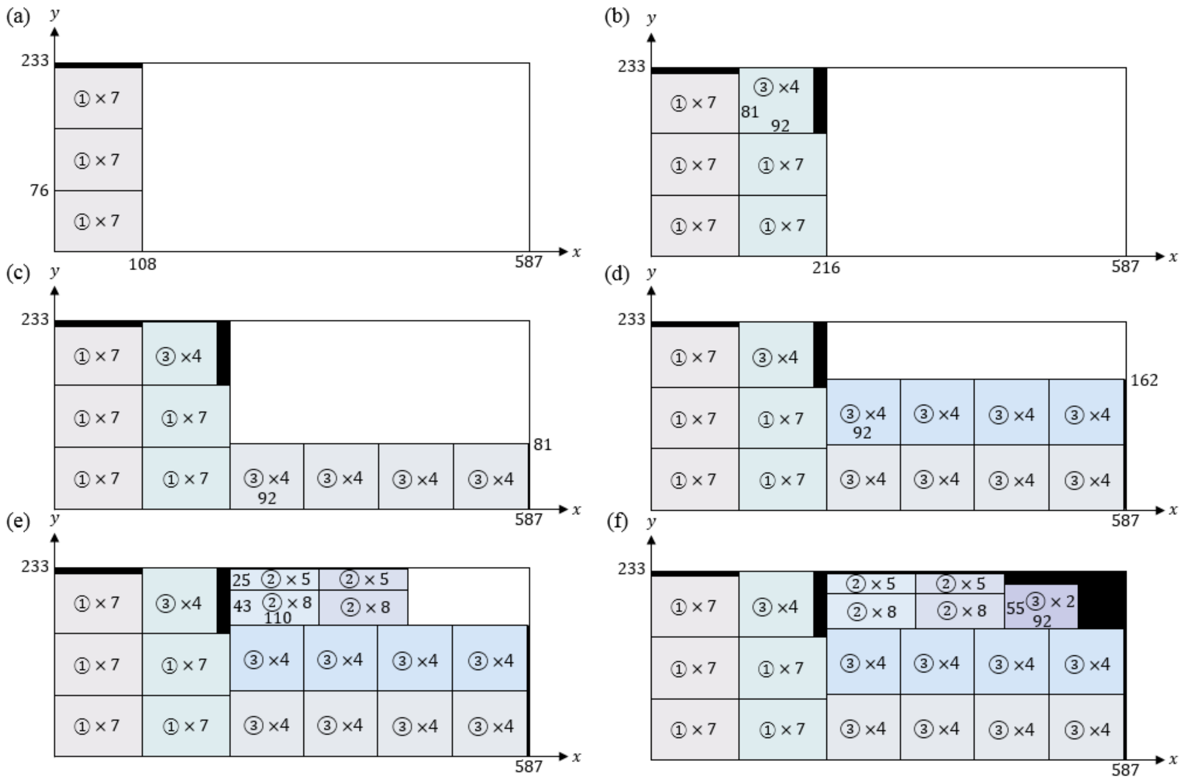

Figure 8 Container packing procedure and a solution of an example from the BR1 data

figure, such as 233, 76, and, 108, represent lengths or widths of boxes and the container. Among three box types, the largest with the smallest dimension is type 1 because only the z-axis dimension of type 1 can be in the vertical orientation, and $76cm$ is the smallest dimension of a box of type 1 that can be used as mx or my. So, in the first phase shown in Figure 8 (a), type 1 is selected as the initial box and a wall in the y-axis direction is built. This wall is the one selected among four possible walls mentioned in Section 3.3. and Figure 6. In Figure 8 (f), a guillotine cutting pattern is completed. The final solution is $x = (35, 26, 38)$ and the volume utilization is

$$\begin{aligned} vu(x) &= \frac{\sum_{i=1}^{3} x_i \times l_i \times w_i \times h_i}{L \times W \times H} \times 100 \\ &= \frac{35 \times 246{,}240 + 26 \times 118{,}250 + 38 \times 409{,}860}{587 \times 233 \times 220} \times 100 \\ &= 90.62\%. \end{aligned}$$

The results of the proposed algorithm for the 700 instances are now compared with the results of Bischoff and Ratcliff (1995), Bortfeldt and Gehring (2001), and Liu et al. (2014); all of these papers fulfilled the orientation and guillotine cutting constraints. Table 3 shows the computational test results of the algorithms for the data from BR1 to BR7. All of the volume utilization and computation time data in this table represent the average values of the 100 instances for each case.

Many parameters of HAGC were determined empirically, and some correlations exist between parameters. In every cases, ρ was three. Figure 9 shows the correlation between computation times and iteration. The computation time increases as the number of box types increases. Moreover, the computation time tends to increase as the iteration increases. Figure 10

Table 3 Comparison of test results for the 700 instances from BR1-BR7 without the complete-shipment constraint

Test case		BR1	BR2	BR3	BR4	BR5	BR6	BR7	Mean BR1-BR7
No. of box types		3	5	8	10	12	15	20	
Bischoff and Ratcliff (1995)	Volume utilization (%)	81.76	81.70	82.98	82.60	82.76	81.50	80.51	81.97
	Computation time (sec.)	-	-	-	-	-	-	-	-
Bortfeldt and Gehring (2001)	Volume utilization (%)	87.81	89.40	90.48	90.63	90.73	90.72	90.65	90.06
	Computation time (sec.)	-	-	-	-	-	-	-	316.00
Liu et al. (2014)	Volume utilization (%)	90.57	91.46	92.39	92.33	92.42	92.35	92.11	91.95
	Computation time (sec.)	61.13	64.37	64.40	63.34	59.52	73.63	86.80	67.60
HAGC without the complete-shipment constraint	Volume utilization (%)	85.86	86.41	87.49	87.32	87.21	86.93	85.95	86.74
	Computation time (sec.)	0.05	0.07	0.15	0.28	0.41	0.62	1.29	0.41
	Standard value (%)	79.00	78.00	77.00	76.00	75.00	74.00	73.00	-
	Tabu tenure (unit)	23	37	45	50	55	57	61	-
	Iteration (cycle)	50	50	50	60	60	60	70	-
	ω	3	1	1	1	1	1	1	-

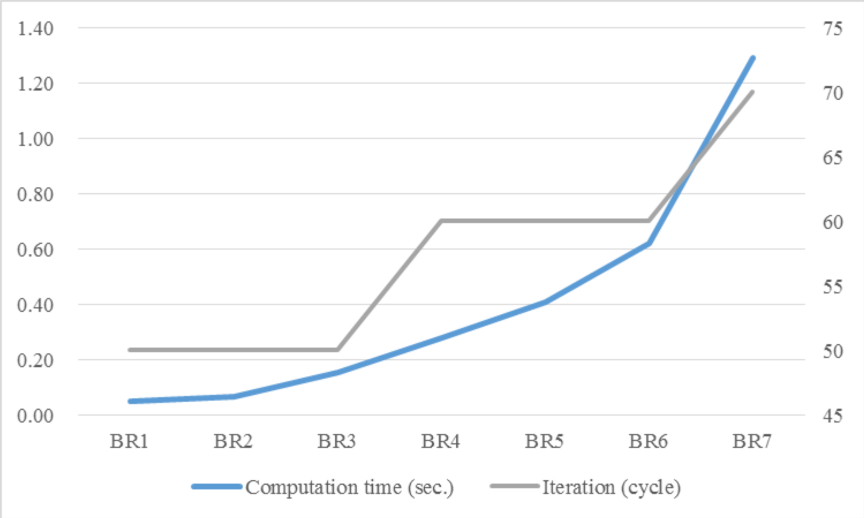

Figure 9 Correlation between computation time and iteration

shows that all tabu tenures are lower than the respective number of iterations. When a tabu tenure is between 50 and 59, the optimal iteration is 60, but in an unexpected outcome, the optimal iteration was 50, not 30 or 40, when the optimal tabu tenure was 23 in BR1. Figure 11 shows that the standard value decreases monotonically and the tabu tenure increases monotonically.

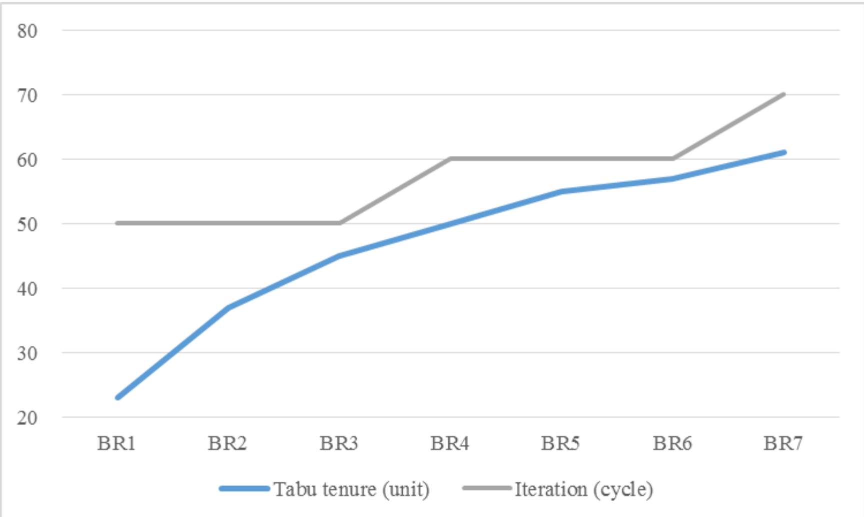

Figure 10 Correlation between the tabu tenure and the iteration

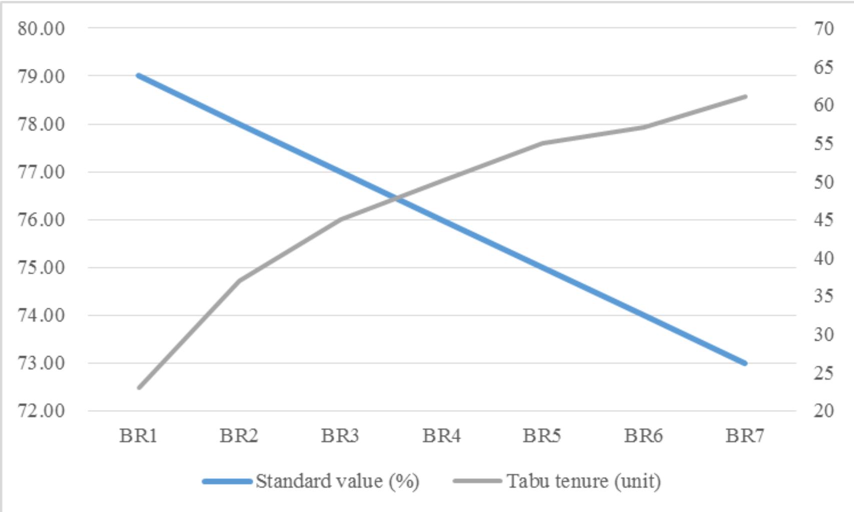

Figure 11 Trend of the standard value and the tabu tenure

Figure 12 shows the comparison of the HAGC to relevant algorithms. The volume utilization of the HAGC is better than that of Bischoff and Ratcliff (1995), but worse than that of the other researchers. The mean gap of the volume utilization is about 3.7% of that of Bortfeldt and Gehring (2001) and about 5.7% of that of Liu et al. (2014).One of the reasons is that maximizing the volume utilization is the principal objective for the three relevant algorithms,

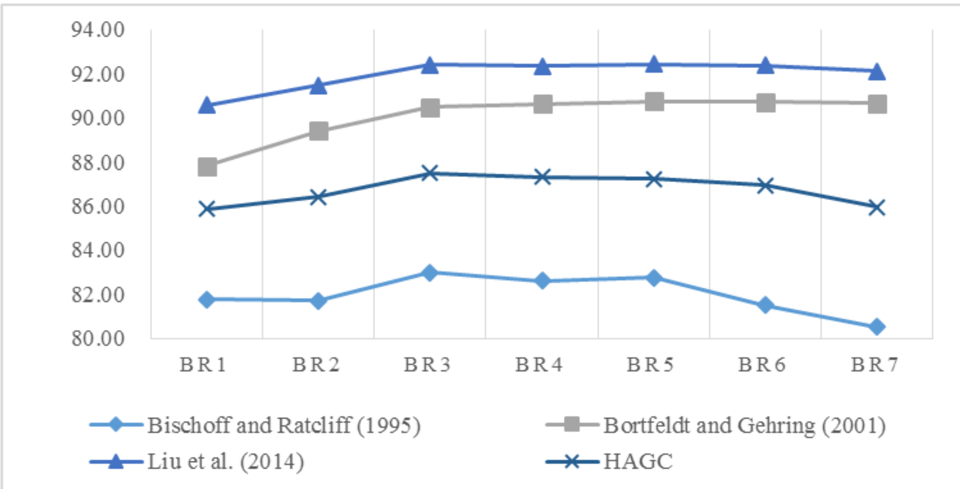

Figure 12 Comparison of the HAGC with other relevant algorithms

but the principal objective of the HAGC is to satisfy the complete-shipment constraint. That is why the volume utilization of the HAGC is worse that of the other algorithms. In addition, Bortfeldt and Gehring (2001) devised a two-phased hybrid genetic algorithm, which improved a solution in the second phase and facilitated diversified exploration for finding a solution. Although the HAGC uses the tabu search, the use is quite restricted. The tree search algorithm is one of the exhaustive search methods; i.e. it is a greedy heuristic algorithm. In the case of Liu et al. (2014), IP models were used to build strips and walls, but the procedure was not made clear. The authors described generalized processes for finding a container packing pattern.

As shown in Table 3, the HAGC features faster computation than the others even if computational environments differ from each other. None of the related papers clarified that computation times are either averages of one instance or total times of 100 instances, but computation times of the HAGC are regarded as the average times of each instance. As the volume utilizations are mean values, it is reasonable to display average times. The mean computation time of the HAGC is about 770 times (or 7.7 times) faster than that of Bortfeldt and Gehring (2001) and about 165 times (or 1.7 times) faster than that of Liu et al. (2014).

5.3. Test results with the complete-shipment constraint

Under the complete-shipment constraint, complying with the complete-shipment rule and maximizing the volume utilization are important. Experiments using the HAGC with the complete-shipment constraint were

Table 4 Comparison of test results for the 700 instances from BR1-BR7 with the complete-shipment constraint

Test case		BR1	BR2	BR3	BR4	BR5	BR6	BR7	Mean BR1-BR7
No. of box types		3	5	8	10	12	15	20	-
Average no. of boxes per type		50	27	17	13	11	9	7	-
HAGC without the complete-shipment constraint	Volume utilization (%)	85.86	86.41	87.49	87.32	87.21	86.93	85.95	86.74
	Computation time (sec.)	0.05	0.07	0.15	0.28	0.41	0.62	1.29	0.41
HAGC with the complete-shipment constraint	Volume utilization (%)	73.10	77.26	80.09	80.55	80.94	81.17	80.86	79.14
	Computation time (sec.)	0.04	0.09	0.26	0.37	0.58	0.87	1.72	0.56
	Standard value (%)	79.00	78.00	77.00	76.00	75.00	74.00	73.00	-
	Tabu tenure (unit)	20	33	47	47	45	57	57	-
	Iteration (cycle)	40	50	50	50	60	60	70	-
	ω	2	1	1	1	1	1	1	-

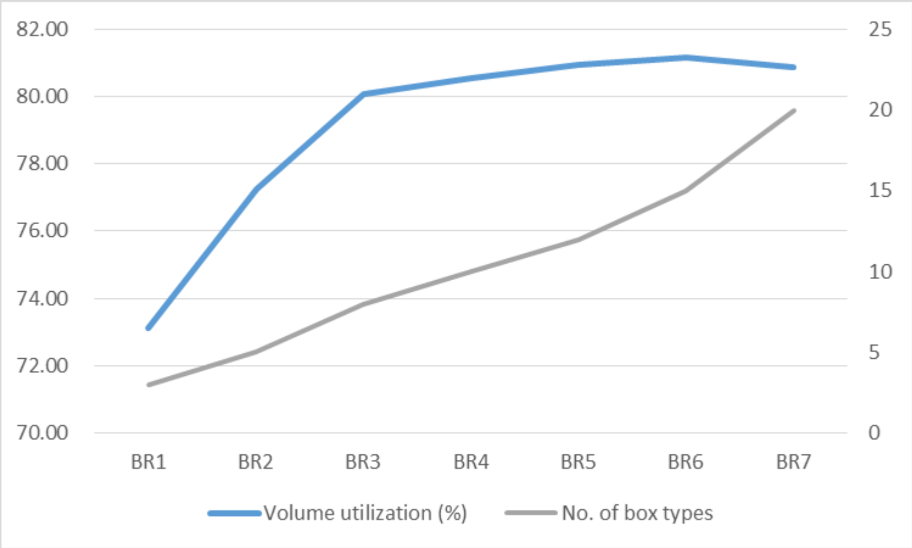

Figure 13 Correlation between the volume utilizations and number of box types

conducted using the same BR data as in the tests without the complete-shipment constraint, and the results were compared with the results of the HAGC evaluation without the complete-shipment constraint. The results are shown in Table 4 and the average number of boxes per box type, as excerpted from Fanslau and Bortfeldt (2010), was added in the table.

Figure 13 shows that as the number of box types increases, the volume utilization also increases. However, the gradient of the curve of the volume utilizations gradually decreases, and when the number of box types is 20, the gradient finally becomes a negative number. This situation can be analyzed by acknowledging that the more box types, the easier it is to fill up empty spaces during the complete-shipment process. However, generally the average number of boxes per type decreases as the number of box types increases. So, available boxes of each type are typically insufficient and it is not easy to increase the volume utilization.

The complete-shipment constraint leads to a decrease in the volume

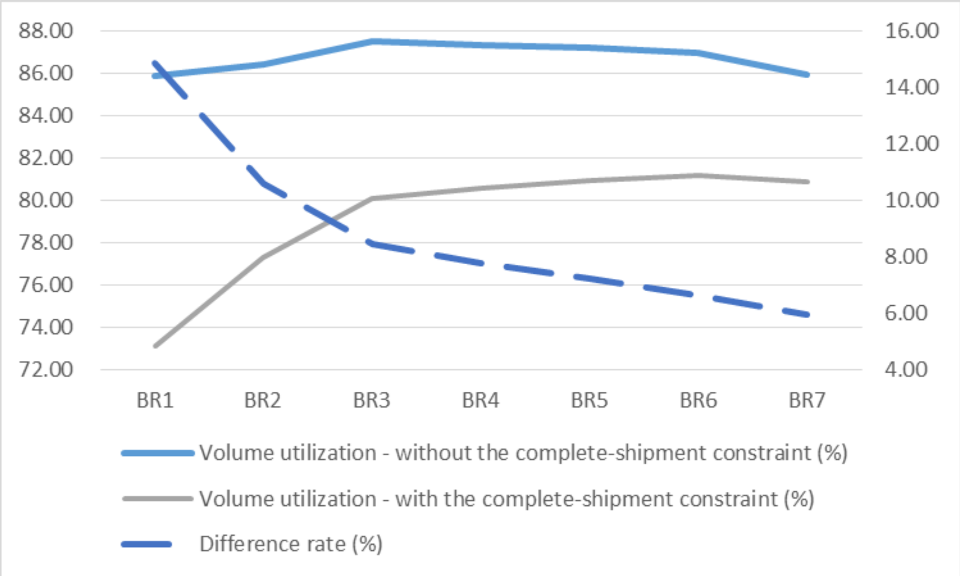

Figure 14 Comparison of two volume utilizations

utilizations as shown in Figure 14. However, the difference rate decreases constantly as the number of box types increases. The reason is that the more box types, the easier it is to fill up empty spaces during the complete-shipment process, similar to the above analysis.

Chapter 6. Conclusions

This paper proposed the container packing algorithm named HAGC (heuristic algorithm with guillotine cutting and complete-shipment constraints) that satisfies three constraints. HAGC is a hybrid approach, combining a tree search algorithm and tabu search algorithms. The wall-building approach was used to make a guillotine cutting pattern. The performance of HAGC was evaluated with computational experiments by using the well-known test data from Bischoff and Ratcliff (1995), so that the volume utilization and the computation time of HAGC could be compared with several other relevant algorithms. When comparing the results, the complete-shipment constraint was not considered. This is because only one paper, Eley (2003), considered the constraint with a multiple container packing problem.

In terms of the volume utilization, HAGC did not perform as well as the algorithms of Bortfeldt and Gehring (2001) and Liu et al. (2014). However, the computation time was shorter. With respect to the complete-shipment constraint, no benchmark paper was found at the time of this writing. Therefore, HAGC with the complete-shipment constraint was compared with HAGC without the constraint. Through experiments, it was confirmed that relative volume utilizations in terms of the HAGC values without the constraint increase as the number of box types increases.

Many researchers have developed two-phased heuristic algorithms, which typically consist of a basic greedy heuristic algorithm and an improvement algorithm. HAGC can be used as a basic greedy heuristic algorithm in the role of offering an initial solution. Then, this research can be extended to improve the performance of HAGC by devising an improvement phase. Also, a strip-building approach is not suitable for considering the complete-shipment constraint. For example, if the complete-shipment group is $(2, 1, 1)$, making a strip with these four boxes is quite inefficient in terms of the volume utilization. Therefore, research on a 3D-CPP with the complete-shipment constraint can be done with a block-building approach.

In addition to these suggestions, some other possibilities for future research appear promising. Other practical constraints, such as the weight limit, need to be considered. More realistic algorithms should be developed. One container packing manager said that existing algorithms offer quite complicated packing patterns and these cannot be implemented within a reasonable time. Therefore, most packing methods depend on managers' experience and some luck. The HAGC gives simple and intuitive packing patterns for manager consideration.

Bibliography

[1] Alvarez-Valdes, R., A. Parajon and J. M. Tamarit (2002). "A tabu search alogorithm for large-scale guillotine (un)constrainted two-dimensional cutting problems." Computers & Operations Research **29**: 925-947.

[2] Amossen, R. R. and D. Pisinger (2010). "Multi-dimensional bin packing problems with guillotine constraints." Computers & Operations Research **37**(11): 1999-2006.

[3] Bischoff, E. E. and M. S. W. Ratcliff (1995). "Issues in the development of approaches to container loading." Omega **23**(4): 377-390.

[4] Bortfeldt, A. and H. Gehring (2001). "A hybrid genetic algorithm for the container loading problem." European Journal of Operational Research **131**: 143-161.

[5] Bortfeldt, A., H. Gehring and D. Mack (2003). "A parallel tabu search algorithm for solving the container loading problem." Parallel Computing **29**(5): 641-662.

[6] Bortfeldt, A. and S. Jungmann (2012). "A tree search algorithm for solving the multi-dimensional strip packing problem with guillotine cutting constraint." Annals of Operations Research **196**(1): 53-71.

[7] Bortfeldt, A. and G. Wäscher (2013). "Constraints in container loading – A state-of-the-art review." European Journal of Operational Research **229**(1): 1-20.

[8] Clautiaux, F., A. Jouglet and A. Moukrim (2013). "A New Graph-Theoretical Model for the Guillotine-Cutting Problem." INFORMS Journal on Computing **25**(1): 72-86.

[9] Crainic, T. G., G. Perboli and R. Tadei (2009). "TS2PACK: A two-level tabu search for the three-dimensional bin packing problem." European Journal of Operational Research **195**(3): 744-760.

[10] Davies, A. P. and E. E. Bischoff (1999). "Weight distribution considerations in container loading." European Journal of Operational Research **114**: 509-527.

[11] de Armas, J., G. Miranda and C. León (2012). "Improving the efficiency of a best-first bottom-up approach for the Constrained 2D Cutting

Problem." European Journal of Operational Research **219**(2): 201-213.

[12] Dolatabadi, M., A. Lodi and M. Monaci (2012). "Exact algorithms for the two-dimensional guillotine knapsack." Computers & Operations Research **39**(1): 48-53.

[13] Eley, M. (2002). "Solving container loading problems by block arrangement." European Journal of Operational Research **141**: 393-409.

[14] Eley, M. (2003). "A bottleneck assignment approach to the multiple container loading problem." OR Spectrum **25**: 45-60.

[15] Fanslau, T. and A. Bortfeldt (2010). "A Tree Search Algorithm for Solving the Container Loading Problem." INFORMS Journal on Computing **22**(2): 222-235.

[16] Feng, X., I. Moon and J. Shin (2015). "Hybrid genetic algorithms for the three-dimensional multiple container packing problem." Flexible Services and Manufacturing Journal **27**(2-3): 451-477.

[17] George, J. A. and D. F. Robinson (1980). "A heuristic for packing boxes into a container." Computers & Operations Research **7**: 147-156.

[18] Gilmore, P. C. and R. E. Gomory (1965). "Multistage cutting stock problems of two and more dimensions." Operations research **13**(1): 94-120.

[19] Gonçalves, J. F. and M. G. C. Resende (2012). "A parallel multi-population biased random-key genetic algorithm for a container loading problem." Computers & Operations Research **39**(2): 179-190.

[20] Liu, J., Y. Yue, Z. Dong, C. Maple and M. Keech (2011). "A novel hybrid tabu search approach to container loading." Computers & Operations Research **38**(4): 797-807.

[21] Liu, S., W. Tan, Z. Xu and X. Liu (2014). "A tree search algorithm for the container loading problem." Computers & Industrial Engineering **75**: 20-30.

[22] Moon, I. and T. V. L. Nguyen (2014). "Container packing problem with balance constraints." OR Spectrum **36**(4): 837-878.

[23] Pisinger, D. (2002). "Heuristics for the container loading problem." European Journal of Operational Research **141**: 382-392.

[24] Ren, J., Y. Tian and T. Sawaragi (2011). "A tree search method for the container loading problem with shipment priority." European Journal of Operational Research **214**(3): 526-535.

[25] Russo, M., A. Sforza and C. Sterle (2013). "An improvement of the knapsack function based algorithm of Gilmore and Gomory for the unconstrained two-dimensional guillotine cutting problem." International Journal of Production Economics **145**(2): 451-462.

[26] Wäscher, G., H. Haußner and H. Schumann (2007). "An improved typology of cutting and packing problems." European Journal of Operational Research **183**(3): 1109-1130.

[27] Wolsey, L. A. (1998). Integer Programming, Wiley-Interscience.

초 록

이 논문은 3차원 컨테이너 적재 문제 (3D-CPP)를 풀 수 있는 트리 탐색 알고리즘을 제시한다. 3D-CPP에 대한 많은 현실적인 제약 조건이 존재하고, 이 논문에서는 화물 방향 제약과 길로틴 절단, 완전 선적 제약을 고려하고 있다. 벽 구축 접근법과 타부 서치를 이용하여 적재율을 최대화하는 알고리즘을 개발하였다. 많은 연구자들이 사용한 BR 실험 데이터를 이용하여 실험한 결과, 본 알고리즘은 빠른 시간 안에 적재율이 상당히 높은 화물 적재 패턴을 찾을 수 있음을 확인하였다. 또한, 본 알고리즘을 통해 완전 선적 조건을 만족하면서 작업자가 쉽게 이해하고 빠르게 구현할 수 있는 화물 적재 패턴을 구할 수 있다.

주요어 : 컨테이너 패킹 문제, 길로틴 절단 패턴, 완전 선적, 벽 구축 접근법, 트리 탐색, 타부 서치

학　번 : 2014-20631