การออกแบบกล่องบรรจุภัณฑ์เพื่อให้ได้จำนวนชนิดของกล่องน้อยที่สุด

นายธีระเดช ไหลสุพรรณวงศ์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาคณิตศาสตร์ประยุกต์และวิทยาการคณนา

ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์

คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

ปีการศึกษา 2560

DESIGNING PACKING BOXES TO MINIMIZE NUMBER OF BOX TYPES

Mr. Teeradech Laisupannawong

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science Program in Applied Mathematics and

Computational Science

Department of Mathematics and Computer Science

Faculty of Science

Chulalongkorn University

Academic Year 2017

Thesis Title	DESIGNING PACKING BOXES TO MINIMIZE NUMBER OF BOX TYPES
By	Mr. Teeradech Laisupannawong
Field of Study	Applied Mathematics and Computational Science
Thesis Advisor	Assistant Professor Boonyarit Intiyot, Ph.D.
Thesis Co-advisor	Associate Professor Phantipa Thipwiwatpotjana, Ph.D.

Accepted by the Faculty of Science, Chulalongkorn University in Partial Fulfillment of the Requirements for the Master's Degree

ธีระเดช ไหลสุพรรณวงศ์ : การออกแบบกล่องบรรจุภัณฑ์เพื่อให้ได้จำนวนชนิดของกล่องน้อยที่สุด. (DESIGNING PACKING BOXES TO MINIMIZE NUMBER OF BOX TYPES) อ.ที่ปรึกษาวิทยานิพนธ์หลัก : ผศ.ดร. บุญฤทธิ์ อินทิยศ, อ.ที่ปรึกษาวิทยานิพนธ์ร่วม : รศ.ดร. พันทิพา ทิพย์วิวัฒน์พจนา 89 หน้า.

ในกระบวนการบรรจุผลิตภัณฑ์ กล่องจำนวนหลายชนิดอาจถูกใช้ในกระบวนการบรรจุถ้าโรงงานมีสินค้าหรือผลิตภัณฑ์หลายชนิด การใช้ต้นทุนสำหรับการใช้กล่องหลายชนิดในการบรรจุอาจเพิ่มต้นทุนในการผลิตให้สูงขึ้น อย่างไรก็ตามเราสามารถเพิ่มประสิทธิภาพในแง่ของการลดต้นทุนและการจัดการการผลิตได้ถ้าเราสามารถออกแบบกล่องที่มีขนาดเหมาะสมและสามารถลดจำนวนชนิดของกล่องสำหรับบรรจุสินค้า ในงานวิจัยนี้เรานำเสนอฮิวริสติกอัลกอลิทึมสำหรับออกแบบกล่องเพื่อนำไปใช้บรรจุสินค้าแต่ละชนิดซึ่งมีรูปทรงสินค้าเป็นสี่เหลี่ยมเมื่อระบุจำนวนชิ้นของสินค้าต่อกล่องมาให้ วัตถุประสงค์ของอัลกอลิทึมนี้คือต้องการออกแบบกล่องทรงสี่เหลี่ยมที่มีรูปทรงใกล้เคียงลูกบาศก์มากที่สุดภายใต้ขอบเขตความยาวที่กำหนด นอกจากนี้ได้เราได้นำเสนอวิธีการฮิวริสติกและกำหนดการเชิงจำนวนเต็มสำหรับลดจำนวนชนิดของกล่องให้น้อยที่สุด กล่องชนิดหนึ่งสามารถถูกแทนด้วยกล่องอีกชนิดหนึ่งได้เมื่อเปอร์เซ็นต์ของผลต่างความยาวแต่ละด้านของกล่องสองกล่องนั้น (เทียบกับกล่องที่ใหญ่กว่า) ไม่เกินค่าค่าหนึ่งที่ระบุมาให้ ตัวอย่างปัญหาถูกแสดงเพื่อให้เห็นการใช้งานของฮิวริสติกและโมเดลที่ได้นำเสนอ

ภาควิชา	คณิตศาสตร์และ	ลายมือชื่อนิสิต ....
	วิทยาการคอมพิวเตอร์	ลายมือชื่อ อ.ที่ปรึกษาหลัก ....
สาขาวิชา	คณิตศาสตร์ประยุกต์	ลายมือชื่อ อ.ที่ปรึกษาร่วม ....
	และวิทยาการคณนา	
ปีการศึกษา	2560	

5971985023 : MAJOR APPLIED MATHEMATICS AND COMPUTATIONAL SCIENCE

KEYWORDS : DESIGNING BOX / BOX TYPE / HEURISTIC / INTEGER LINEAR PROGRAMMING

TEERADECH LAISUPANNAWONG : DESIGNING PACKING BOXES TO MINIMIZE NUMBER OF BOX TYPES. ADVISOR : ASST. PROF. BOONYARIT INTIYOT, Ph.D., COADVISOR : ASSOC. PROF. PHANTIPA THIPWIWATPOTJANA, Ph.D., 89 pp.

In a product packing procedure, many types of packing boxes may be used if a factory has several kinds of goods or products. The cost spent for many types of boxes is added to the manufacturing cost. However, it would be more efficient in the aspects of the cost reduction and the production management if we can design reasonable box sizes and can minimize the number of box types for packing goods. In this work, we propose a heuristic rectangular box design algorithm for packing each kind of rectangular goods when the number of goods per box is given. The objective of this algorithm is to design the rectangular boxes with the shapes close to cubes as much as possible under the given bounds. Furthermore, we propose a heuristic method and an integer linear programming model for minimizing the number of types of packing boxes. A smaller box could be substituted by a larger one only when the percentage change on each size of both boxes (with respect to the larger box) does not exceed a given value. Example problems are given to illustrate the use of the proposed heuristic and model.

Department :	Mathematics and	Student's Signature ....
	Computer Science	Advisor's Signature ....
Field of Study :	Applied Mathematics and	Co-advisor's Signature ....
	Computational Science	
Academic Year :	2017	

ACKNOWLEDGEMENTS

Firstly, I would like to thank my advisor Assistant Professor Boonyarit Intiyot and my co-advisor Associate Professor Phantipa Thipwiwatpotjana for their invaluable help, guidance, and encouragement. They always give good advice when I had problems or questions about this research. I could not complete this master thesis without their support.

I also would like to thank my thesis committees, Assistant Professor Krung Sinapiromsaran, Assistant Professor Petarpa Boonserm, and Associate Professor Chawalit Jeenanunta for their recommendations and suggestions.

In addition, I am thankful to my friends, AMCS students, for their suggestions and their helps, especially Mr.Thanet Markchom who helped me when I had a problem about computer programming.

Moreover, I would like to acknowledge to the Development and Promotion of Science and Technology Talents Project (DPST) for funding and support all of master degree program study.

Finally, I am most grateful to my mother and brother for their encouragement and support. They always foster me throughout the study and everything in my life.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Motivation

"A good packing design can reduce the manufacturing cost." This statement motivates the main idea of our research study which focuses on designing reasonable boxes for packing goods and reduce some box types when we have too many types of boxes. We want to design a box that can hold some number of units of the same rectangular shape product. If we know the size of the box, an arrangement of a number of the same products in this box is a special case of a container loading problem. Therefore, we try to apply the container loading problem ideas together with some other designing criteria to create a reasonable box. The term "reasonable box" is actually depending on the purpose of the packing. For example, if the box is meant to be used for packing a body size mirror, this box may be thin but the longest length could be at least 180 cm. In this case, it would not be that good if the box becomes a cube. Therefore we need to set up some criteria for our term "reasonable". First of all, the box we create must be able to hold a given number of units of the same product as mentioned earlier, where the given number is an input information from a user. This is the main requirement that the designed box should meet. Other than that is the matter of how to design it reasonably. We may also consider the criterion of the unused space in the box by limiting it proportionally to the whole box so that the items in the box would not move that much during transportation. Another aspect that is also important is to design the box to become as close to a cube as possible after the box satisfies the two criteria above. This is because a "close to a cube" box is more stable than other rectangular boxes. Under the scope of our "reasonable box", we hope to create optimization model(s) and method(s) that can help us design boxes for packing goods and reduce their types.

1.2 Objectives

We conclude our objectives of this research again as follows.

1. To design a box for packing a fixed number of the same kind of goods into one box. The box that we design will have the shape close to a cube as much as possible within some certain boundaries and criteria.

2. To propose a method and an optimization model for minimizing the number of types of packing boxes.

In this work, we will design a box for each kind of goods/products. Then all designed box types will be minimized by using the proposed method/optimization model. As a result, we will get boxes that fit our needs and can reduce the packing cost by minimizing the number of box types.

1.3 Assumptions of this work

The assumptions of this thesis are as follows.

1. Each kind of goods has rectangular shape.

2. Each unit of goods will be called an item.

3. Each item is orthogonally positioned in the container box; i.e., the edges of an item are either parallel or perpendicular to the axes of the container box.

4. Each item must be packed right side up in a box (the height of the item is perpendicular to the floor), i.e., the item can be rotated in only two cases as shown in Figure 1.1.

5. The upper bounds of the dimensions of boxes are given.

6. The number of goods per box is given.

7. One designed box for packing a kind of goods can be used to pack other kinds of goods.

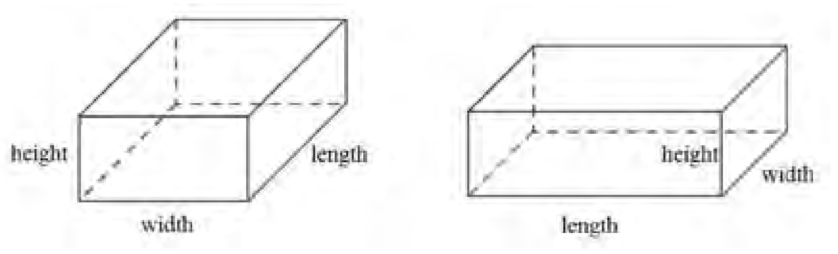


Figure 1.1: Two types of rotations for each item of goods.

1.4 Background knowledge

In this section, we present background knowledge for this research. We start with definitions of the dimensions of a box, linear programming, either-or constraints, and heuristics. After that, we describe the container loading problem which is a combinatorial problem that our model is based on.

1.4.1 Definition of the height, the length, and the width of a box

According to the announcement No.4432 (2012) from Ministry of Industry [13], the definitions of the length, the width, and the height of a box are presented below.

1. The height of a box refers to the only dimension without a flap.
2. The length of a box is always the longest side of the box that has a flap.
3. The width of a box is always the second longest side of the box that has a flap.

Figure 1.2 shows how to call each measurement of a box according to [13]. Some examples of box types are shown in Figure 1.3.

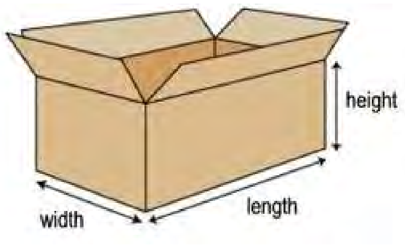


Figure 1.2**:** Height, length, and width of a box.

Credit : https://www.esupplystore.com/How-To-Measure-A-Box-ep-32-1.html

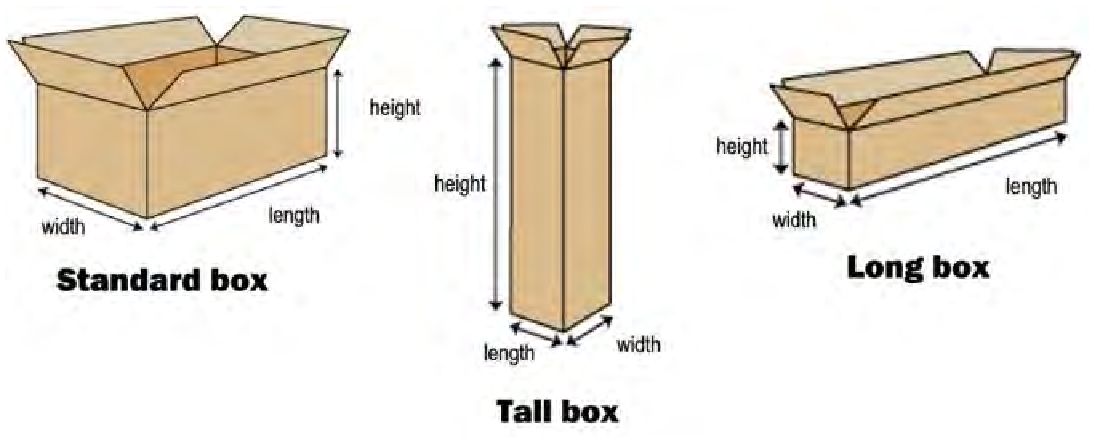


Figure 1.3**:** Some examples of box types.

Credit : https://www.esupplystore.com/How-To-Measure-A-Box-ep-32-1.html

1.4.2 **Linear programming**

A linear programming (LP) problem is an optimization problem where its objective function is a linear function and its constraints are linear equalities or linear inequalities. A linear programming problem can be formulated in a canonical form as follows:

$$\begin{aligned} \max \quad & z := c_1x_1 + c_2x_2 + ... + c_nx_n \\ \text{s.t.} \quad & a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \leq b_1 \\ & a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n \leq b_2 \\ & \qquad\qquad \vdots \\ & a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n \leq b_m \\ & x_1 \geq 0, x_2 \geq 0, ..., x_n \geq 0. \end{aligned}$$

In an LP problem, the objective function can be maximizing or minimizing. We can convert the LP problem from maximizing to minimizing by replacing the objective function z to $-z$. An LP problem with some inequalities "$\geq$" or equalities constraints can be converted to the canonical form as follows.

1. An inequality '$\geq$' type constraint
$$a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n \geq b_i$$
can be converted to an inequality '$\leq$' type constraint by multiplying -1 to the inequality:
$$-a_{i1}x_1 - a_{i2}x_2 - ... - a_{in}x_n \leq -b_i.$$

2. An equality constraint
$$a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n = b_i$$
can be converted to '$\leq$' inequality form by replacing it with two inequality constraints:
$$-a_{i1}x_1 - a_{i2}x_2 - ... - a_{in}x_n \leq -b_i$$
$$a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n \leq b_i.$$

The objective of an LP problem is to find an optimal solution. In our case, we want to maximize the objective value. A well known approach used for solving the LP problem is the simplex method. A set of vectors $(x_1, x_2, ..., x_n)$ satisfying all the constraints is called a feasible solution. A feasible solution $(x_1, x_2, ..., x_n)$ is called optimal if it maximizes the objective function. If all decision variables of the LP problem require to have integer values, the problem will be the integer linear programming (ILP) problem. The problem is called the mixed integer linear programming (MILP) problem when some decision variables are integers.

1.4.3 **Either-or constraints**

Either-or constraints are non-simultaneous logical constraints. At least one of two constraints expressing "either-or" must be hold. A general form of the either-or constraints can be expressed as

$$\left.\begin{array}{ll}\text{Either} & f(x_1, x_2, ..., x_n) \leq d_1 \\ \text{or} & g(x_1, x_2, ..., x_n) \leq d_2,\end{array}\right\} \tag{1.1}$$

where f and g are linear.

An LP problem which has either-or constraints has to be reformulated into a new problem so that all constraints in the new problem must be satisfied simultaneously. Let F_1 and F_2 be the sets of all feasible solutions of the first and the second constraint in (1.1), respectively. Suppose that all linear constraints in (1.1) are bounded in the intersection of F_1 and F_2, i.e. $\exists M_1 \in \mathbb{R}$ such that $\forall (x_1, x_2, ..., x_n) \in F_1 \cap F_2, f(x_1, x_2, ..., x_n) \leq M_1$ and $\exists M_2 \in \mathbb{R}$ such that $\forall (x_1, x_2, ..., x_n) \in F_1 \cap F_2, g(x_1, x_2, ..., x_n) \leq M_2$. The either-or constraints can become two simultaneous inequality linear constraints using an auxiliary binary variable y and a large positive number M, where $M \geq \max\{M_1, M_2\}$; i.e., the either-or constraints can be replaced by

$$\left.\begin{array}{ll} f(x_1, x_2, ..., x_n) & \leq b_1 + My \\ g(x_1, x_2, ..., x_n) & \leq b_2 + M(1-y).\end{array}\right\} \tag{1.2}$$

Note that if $y = 0$, $g(x_1, x_2, ..., x_n) \leq b_2 + M$ is redundant and the first constraint, $f(x_1, x_2, ..., x_n) \leq b_1$, is satisfied. If $y = 1$, the constraint $f(x_1, x_2, ..., x_n) \leq b_1 + M$ is redundant and the other constraint, $g(x_1, x_2, ..., x_n) \leq b_2$, is satisfied. Hence, the either-or constraints (1.1) can be added to an LP problem using the form (1.2), and the problem turns to be the MILP one due to the binary variable y.

A following example illustrates how to reformulate an LP problem with either-or constraints into the MILP problem.

Example 1. Consider the LP problem

$$\begin{array}{lll} \max & & z := 3x_1 + 2x_2 \\ \text{s.t.} & & 6x_1 + 6x_2 \leq 420 \\ & \text{Either} & 3x_1 + 6x_2 \leq 300 \\ & \text{or} & 4x_1 + 2x_2 \leq 240 \\ & & x_1 \geq 0, x_2 \geq 0. \end{array}$$

Let M be a large positive number and y be a binary variable. This LP problem can be reformulated into the MILP problem as follows.

$$\begin{array}{ll} \max & z := 3x_1 + 2x_2 \\ \text{s.t.} & 6x_1 + 6x_2 \leq 420 \\ & 3x_1 + 6x_2 \leq 300 + My \\ & 4x_1 + 2x_2 \leq 240 + M(1-y) \\ & x_1 \geq 0, x_2 \geq 0, y \in \{0,1\}. \end{array}$$

1.4.4 **Heuristics**

In the real world, many optimization problems have very large size or nonlinearity. It is not easy to find a global optimal solution within acceptable time. Heuristic algorithms are designed for finding an approximate solution of an optimization problem. The solution may not be the best of all solutions to the problem but it is acceptably good. Heuristic algorithms aim to find or to discover a solution by trial and error. In general, a heuristic algorithm will be designed upon a problem at hand. Heuristic algorithms are often used to solve NP-hard problems. The NP-hard problems are intrinsically difficult problems that are too complex to be solved in polynomial time. An example of the NP-hard problems is the container loading problem.

1.4.5 **The container loading problem**

The container loading problem (CLP) is to find the best way of loading a given set of rectangular boxes into a given rectangular container where the length of the occupied space in the container is minimized. The CLP plays a crucial role in logistics planning and scheduling. An optimal filling of a container can reduce the transportation cost. The CLP is also referred to as the packing problem, the bin packing problem and the knapsack loading problem. The packing problem aims to seek the best way of loading a given set of rectangular boxes into a large rectangular container where the volume of the container is minimized. The bin packing problem is to pack a given number of boxes into a minimum number of bins or containers. The knapsack loading problem is the problem of loading a subset of boxes into a container of fixed dimension such that the volume of the packed boxes is maximized (maximize volume utilization) or the waste space in the container is minimized. The CLP is a problem from the real world. The constraints for the CLP are as follows.

1. No two packed boxes can overlap in the container.
2. Each box lies completely inside the container.
3. Each box is orthogonally positioned in the container; i.e., the edges of a box are either parallel or perpendicular to the axes of the container.

1.4.5.1 **Model of the container loading problem**

According to Chen et al. [1] and Huang et al. [11], the CLP can be modeled as a mixed integer linear programming model. There are assumptions that the container is placed with its length along the x-axis and its width along the y-axis as well as the left-front-bottom corner of the container is fixed at the origin in 3D coordinate system as shown in Figure 1.4. In addition, this model does not consider the weight distribution of boxes in the container.

The parameters and variables used in the model of the CLP are defined as below.

Parameters:

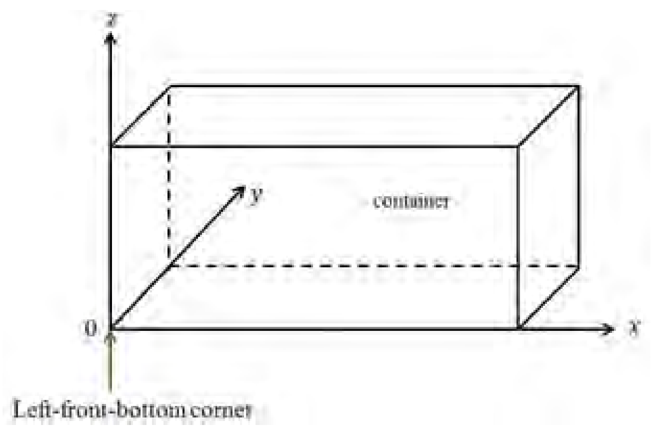


Figure 1.4**:** The left-front-bottom corner of the container.

n	Total number of the given set of boxes to be loaded.
N	The index set of boxes, $N = \{1, 2, ..., n\}$.
$\overline{x}, \overline{y}, \overline{z}$	The length, width, and height of the container, respectively.
M	A large positive number.
p_i, q_i, r_i	The length, width, and height of box i, respectively.

Variables:

x, y, z	Decision variables representing the magnitude along the x-, y-, and z-axes, of the space that required for packing all n boxes into the container, respectively. It is clear that $0 \leq x \leq \overline{x}, 0 \leq y \leq \overline{y}$, and $0 \leq z \leq \overline{z}$.
(x_i, y_i, z_i)	Decision variables representing the coordinates of the left-front-bottom corner of box i.
l_{xi}, l_{yi}, l_{zi}	Binary variables that will equal to 1 if the length of box i is parallel to the x-axis, the y-axis, and the z-axis, respectively. For example, $l_{xi} = 1$ if the length of box i is parallel to the x-axis; otherwise $l_{xi} = 0$. It is clear that $l_{xi} + l_{yi} + l_{zi} = 1$.
w_{xi}, w_{yi}, w_{zi}	Binary variables that will equal to 1 if the width of box i is parallel to the x-axis, the y-axis, and the z-axis, respectively. For example, $w_{xi} = 1$ if the width of box i is parallel to the x-axis; otherwise $w_{xi} = 0$. It is clear that $w_{xi} + w_{yi} + w_{zi} = 1$.

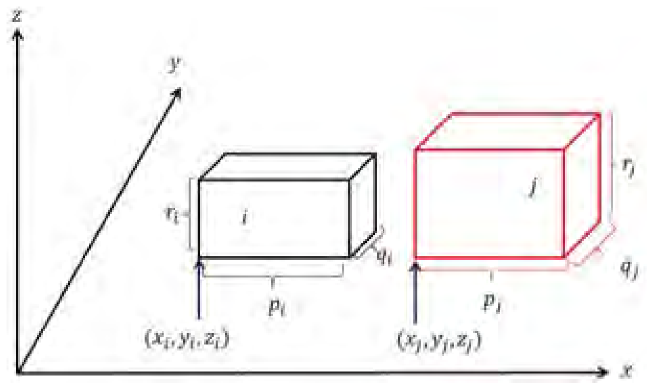
h_{xi}, h_{yi}, h_{zi}	Binary variables that will equal to 1 if the height of box i is parallel to the x-axis, the y-axis, and the z-axis, respectively. For example, $h_{xi} = 1$ if the height of box i is parallel to the x-axis; otherwise $h_{xi} = 0$. It is clear that $h_{xi} + h_{yi} + h_{zi} = 1$.
$(\alpha_{ij}, \beta_{ij}, \delta_{ij})$	Binary variables indicating the relative positions of box i and box j; i.e., (a) $(\alpha_{ij}, \beta_{ij}, \delta_{ij}) = (0, 0, 1)$ if box i is on the left of box j. (b) $(\alpha_{ij}, \beta_{ij}, \delta_{ij}) = (0, 1, 0)$ if box i is on the right of box j. (c) $(\alpha_{ij}, \beta_{ij}, \delta_{ij}) = (1, 0, 0)$ if box i is in front of box j. (d) $(\alpha_{ij}, \beta_{ij}, \delta_{ij}) = (0, 1, 1)$ if box i is behind box j. (e) $(\alpha_{ij}, \beta_{ij}, \delta_{ij}) = (1, 0, 1)$ if box i is below box j. (f) $(\alpha_{ij}, \beta_{ij}, \delta_{ij}) = (1, 1, 0)$ if box i is above box j. Note that there are 6 relative positions of any two boxes, i.e. left, right, in front of, behind, below, and above. However, the combination of the binary variables $(\alpha_{ij}, \beta_{ij}, \delta_{ij})$ yields 8 possible cases, which is 2 more cases than we need. The authors cleverly discarded 2 cases (0,0,0) and (1,1,1) by enforcing additional constraint (7) to limit the possible combinations to only 6 cases. Therefore, each relative position can be assigned to a remaining combination of $(\alpha_{ij}, \beta_{ij}, \delta_{ij})$ as shown in (a)–(f).

The mathematical model which is the mixed integer linear programming of the CLP can be stated as follows (See [1, 11]):

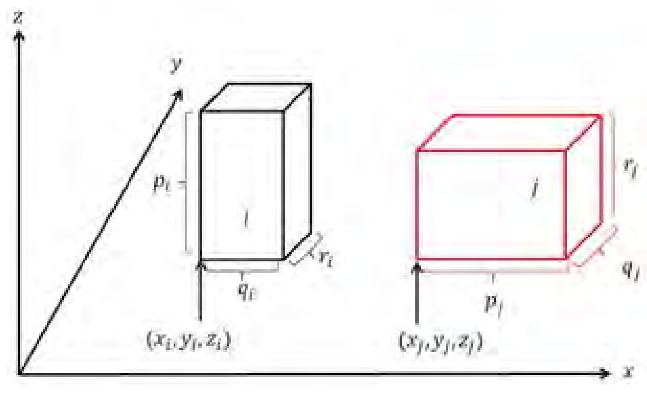
$$
\begin{array}{llrr}
\min & x & & \\
\text{s.t.} & x_i + p_i l_{xi} + q_i w_{xi} + r_i h_{xi} \le x_j + M(1 + \alpha_{ij} + \beta_{ij} - \delta_{ij}) & \forall i, j \in N, i < j, & (1) \\
& x_j + p_j l_{xj} + q_j w_{xj} + r_j h_{xj} \le x_i + M(1 + \alpha_{ij} - \beta_{ij} + \delta_{ij}) & \forall i, j \in N, i < j, & (2) \\
& y_i + p_i l_{yi} + q_i w_{yi} + r_i h_{yi} \le y_j + M(1 - \alpha_{ij} + \beta_{ij} + \delta_{ij}) & \forall i, j \in N, i < j, & (3) \\
& y_j + p_j l_{yj} + q_j w_{yj} + r_j h_{yj} \le y_i + M(2 + \alpha_{ij} - \beta_{ij} - \delta_{ij}) & \forall i, j \in N, i < j, & (4) \\
& z_i + p_i l_{zi} + q_i w_{zi} + r_i h_{zi} \le z_j + M(2 - \alpha_{ij} + \beta_{ij} - \delta_{ij}) & \forall i, j \in N, i < j, & (5) \\
& z_j + p_j l_{zj} + q_j w_{zj} + r_j h_{zj} \le z_i + M(2 - \alpha_{ij} - \beta_{ij} + \delta_{ij}) & \forall i, j \in N, i < j, & (6) \\
& 1 \le \alpha_{ij} + \beta_{ij} + \delta_{ij} \le 2 & \forall i, j \in N, i < j, & (7) \\
& x_i + p_i l_{xi} + q_i w_{xi} + r_i h_{xi} \le x & \forall i \in N, & (8) \\
& y_i + p_i l_{yi} + q_i w_{yi} + r_i h_{yi} \le y & \forall i \in N, & (9) \\
& z_i + p_i l_{zi} + q_i w_{zi} + r_i h_{zi} \le z & \forall i \in N, & (10) \\
& l_{xi} + l_{yi} + l_{zi} = 1 & \forall i \in N, & (11) \\
& w_{xi} + w_{yi} + w_{zi} = 1 & \forall i \in N, & (12) \\
& h_{xi} + h_{yi} + h_{zi} = 1 & \forall i \in N, & (13) \\
& l_{xi} + w_{xi} + h_{xi} = 1 & \forall i \in N, & (14) \\
& l_{yi} + w_{yi} + h_{yi} = 1 & \forall i \in N, & (15) \\
& l_{zi} + w_{zi} + h_{yi} = 1 & \forall i \in N, & (16)
\end{array}
$$

$x_i, y_i, z_i \ge 0, 0 \le x \le \overline{x}, 0 \le y \le \overline{y}, 0 \le z \le \overline{z}; l_{xi}, l_{yi}, l_{zi}, w_{xi}, w_{yi}, w_{zi}, h_{xi}, h_{yi}, h_{zi},$ $\alpha_{ij}, \beta_{ij},$ and δ_{ij} are binary variables.

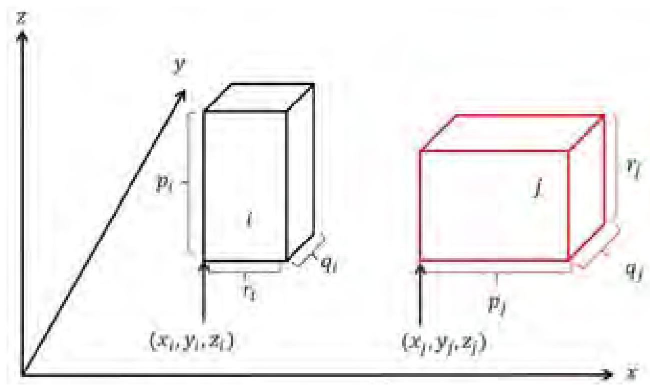
The objective function is to minimize the magnitude along the x-axis of the required space in the container for packing all boxes. Constraints (1)–(7) ensure that no two packed boxes can overlap in the container. As mentioned earlier, constraint (7) limits the possible combinations to only 6 cases. To ensure that box i and box j does not overlap, it suffices to enforce one relative position to be true between them. Constraints (1)–(6) ensure that one relative position is enforced between box i and box j. For example, if $(\alpha_{ij}, \beta_{ij}, \delta_{ij}) = (0, 0, 1)$, constraint (1) becomes $x_i + p_i l_{xi} + q_i w_{xi} + r_i h_{xi} \le x_j$ and constraints (2)–(6) will be redundant. This makes sure that box i is on the left of box j because the value x_i plus the magnitude of the side of box i that is parallel to the x-axis (depend on the rotation of box i, i.e. which one of l_{xi}, w_{xi}, and h_{xi} is equal to 1) does not



(a) Box i is on the left of box j with $l_{xi} = 1$ $(x_i + p_i \leq x_j)$.



(b) Box i is on the left of box j with $w_{xi} = 1$ $(x_i + q_i \leq x_j)$.



(c) Box i is on the left of box j with $h_{xi} = 1$ $(x_i + r_i \leq x_j)$.

Figure 1.5**:** Box i is on the left of box j.

exceed the value x_j (See Figure 1.5 for an example of each rotation of box i). The interpretations of other cases of $(\alpha_{ij}, \beta_{ij}, \delta_{ij})$ can be explained similarly. Furthermore, it suffices to define constraints (1)–(7) only when $i < j$ since if we know the position of box i relative to box j, we also know the position of box j relative to box i. Hence, in a case where $i > j$ such as $(i, j) = (7, 2)$, constraints (1)–(6) will enforce a position of box 2 relative to box 7, say, box 2 is above box 7. This also enforces box 7 to be below.

Constraints (8)–(10) serve for the fact that all boxes must be packed within the required space in the container. Constraint (8) means the value of the coordinate left-front-bottom along the x-axis of each box i (x_i) plus the magnitude of the side along the x-axis of itself can not exceed the magnitude along the x-axis of the space that required for packing all boxes (x). Constraint (9) and (10) can be explained similarly except that they consider along the y-axis and z-axis, respectively.

Constraints (11)–(13) make sure that the length, the width, and the height of the box is parallel to one of the x-axis, the y-axis, or the z-axis, respectively. The values of $l_{xi} + l_{yi} + l_{zi}$, $w_{xi} + w_{yi} + w_{zi}$, and $h_{xi} + h_{yi} + h_{zi}$ cannot be 2 or 3 because one side of a box (the length, the width, or the height) cannot be parallel to 2 or 3 axes simultaneously.

Constraints (14)–(16) ensure that there is only one of the length, the width, and the height of the box parallel to the x-axis, the y-axis, or the z-axis, respectively. The values of $l_{xi} + w_{xi} + h_{xi}$, $l_{yi} + w_{yi} + h_{yi}$, and $l_{zi} + w_{zi} + h_{zi}$ cannot be 2 or 3 because each axis (x-axis, y-axis, or z-axis) cannot be parallel to 2 or 3 sides of a box simultaneously.

In this model, the value of a large positive number M can be $M = max\{\overline{x}, \overline{y}, \overline{z}\}$ because it can certainly dominate the left-hand-side (LHS) of the constraint (1), (2), (3), (4), (5), or (6) when redundancy is needed. This is because, by constraints (8)–(10), we can see that the left-hand side (LHS) of constraint (8), (9), and (10) (which is equivalent to the LHS of constraints (1)–(2), (3)–(4), and (5)–(6), respectively) are less than or equal to the values x, y, and z, respectively. In addition, we also know that $x \leq \overline{x}$, $y \leq \overline{y}$, and $z \leq \overline{z}$. Therefore, the LHS of constraints (1)–(2), (3)–(4), and (5)–(6) will be less than or equal to $M = max\{\overline{x}, \overline{y}, \overline{z}\}$.

The CLP is an NP-hard problem [5, 11]. Hence, a heuristic method is needed for solving a large CLP.

1.4.5.2 A heuristic algorithm for solving the container loading problem

Due to the NP-hardness of the CLP in nature, it is not easy to find an optimal solution when the size of the problem is large. Many heuristic algorithms have been developed to cope this problem. We state an idea of a loading process of a heuristic algorithm used in [11] for solving the CLP in Algorithm 1. Note that the algorithm that is stated here does not consider the rotation of boxes. The parameters used in this algorithm are the same as in the model of the CLP in the previous subsection. Suppose that we have n boxes; i.e., box 1, box 2, ..., box n. In this heuristic algorithm, all boxes will be loaded one by one into the container. The heuristic algorithm for loading n boxes into the container are described in Algorithm 1.

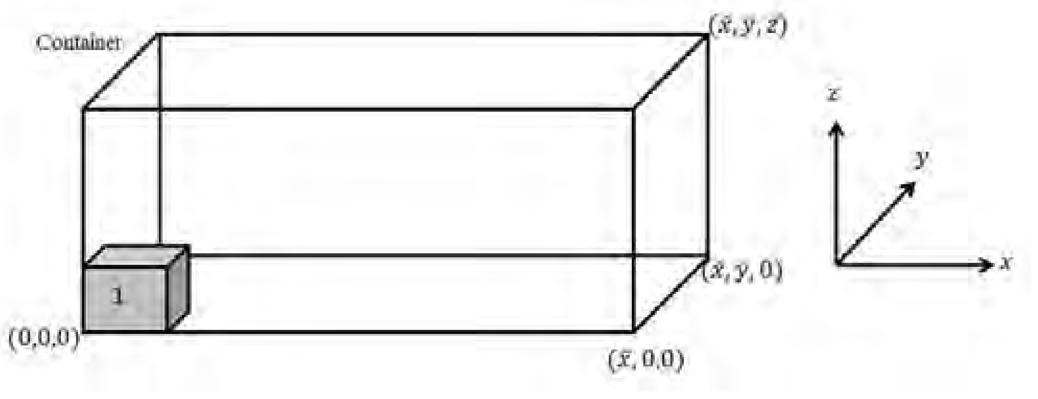
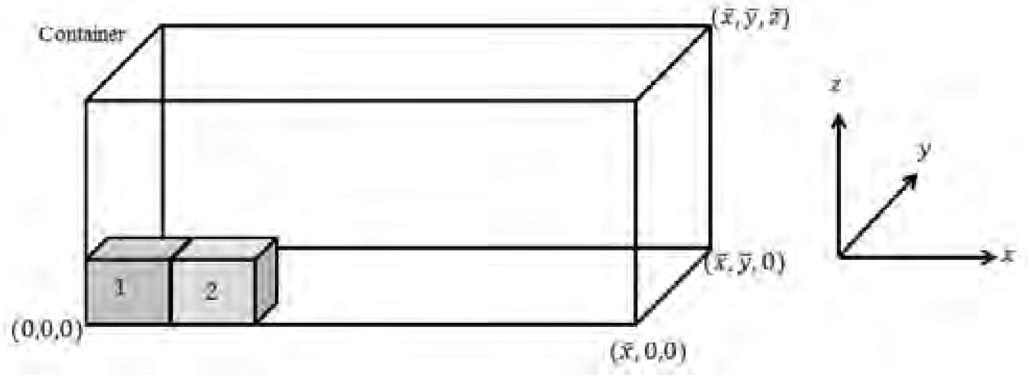


Figure 1.6: Initial step of Algorithm 1.

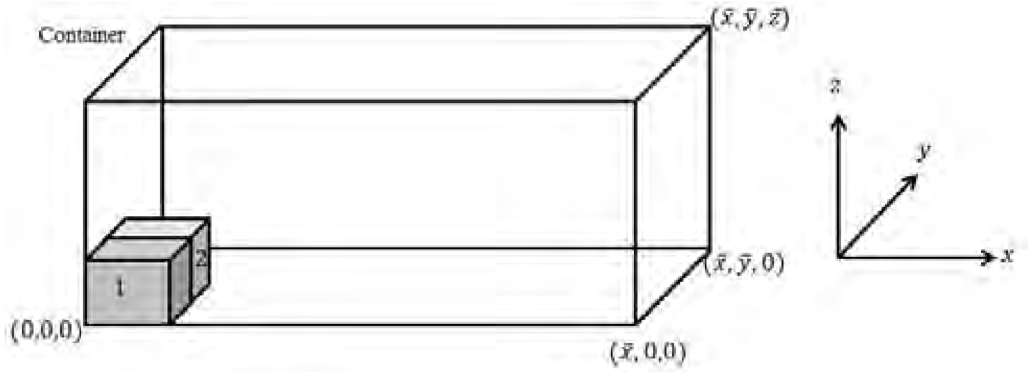
The algorithm begins with loading the first box into the container by setting its left-front-bottom corner to be at (0,0,0). For the remaining boxes $i = 2$ to n, each box i will be verified in these steps. Step 1, box i will be tried to load along the x-axis. If we can do Step 1, then the coordinate of the left-front-bottom of box i will be assigned as the coordinate of the right-front-bottom of the loaded box k which has the smallest value of $x_k + p_k$ comparing to other loaded boxes. Step 2, box i will be tried to load along the y-axis. If we can do Step 2, then the coordinate of the left front bottom of box i will be reassigned as the coordinate of the left-behind-bottom of the loaded box k which has the

Algorithm 1 Heuristic algorithm for solving the CLP (See [11])

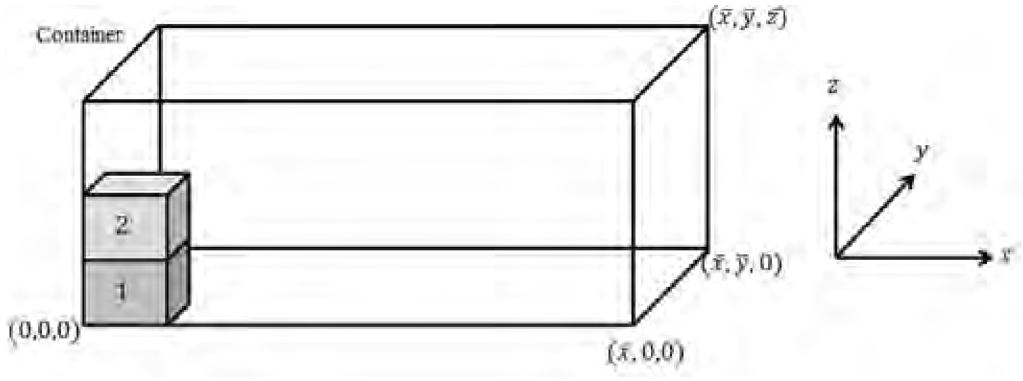
```
1: Load box $i = 1$ into the container by setting the left-front-bottom corner of
   box $i = 1$ at the origin (See Figure 1.6).
2: for $i = 2$ to $n$ do
3:     Step 1 : load box $i$ by attaching the left-front-bottom corner of box $i$ to
       the right-front-bottom corner of the loaded box $k$ such that $x_k + p_k$ is the
       smallest among all loaded boxes as well as box $i$ does not overlap with any
       loaded box and lies completely inside the container (See Figure 1.7 (a)).
4:     Step 2 : Move box $i$ by attaching the left-front-bottom corner of box $i$ to
       the left-behind-bottom corner of the loaded box $k$ such that $y_k + q_k$ is the
       smallest among all loaded boxes as well as box $i$ does not overlap with any
       loaded box and lies completely inside the container if possible (See Figure
       1.7 (b)).
5:     Step 3 : Move box $i$ by attaching the left-front-bottom corner of box $i$ to
       the left-front-top corner of the loaded box $k$ such that $z_k + r_k$ is the smallest
       among all loaded boxes as well as box $i$ does not overlap with any loaded
       box and lies completely inside the container if possible (See Figure 1.7 (c)).
6:     if all three steps cannot be done then
7:         return Infeasibility
8:     end if
9: end for
10: return $\max_{i \in N}\{x_i + p_i\}$
```



(a) Step 1 of box $i = 2$.



(b) Step 2 of box $i = 2$.



(c) Step 3 of box $i = 2$.

Figure 1.7: Step 1 – 3 of box $i = 2$.

smallest value of $y_k + q_k$ comparing to other loaded boxes. Lastly, box i will be tried to load along the z-axis in Step 3. If it can be done, the coordinate of the left-front-bottom of box i will be reassigned as the coordinate of the left-front-top of the loaded box k which has the smallest value of $z_k + r_k$ comparing to other loaded boxes. From this procedure, it means that the coordinate of the left-front-bottom of box i will be firstly assigned from loading along the z-axis if possible. If this cannot be done, the algorithm will backtrack to use the value from Step 2 and Step 1, respectively. In other words, any box i will be loaded along the z-axis first, the y-axis second, and the x-axis third. It corresponds to the objective function which is to minimize the magnitude along the x-axis of the space that is required for packing all n boxes. However, if all 3 steps cannot be done in any box i, the algorithm will return infeasibility since all n boxes cannot be loaded into the container. Otherwise, the algorithm will return the magnitude along the x-axis of the space that required for packing all boxes into the container.

Next, we will give a small example of the CLP and we solve it by using this heuristic algorithm.

Example 2. Consider the following small CLP. The container with the length of 6 m, the width of 3 m, and the height of 2 m is given for packing six cube boxes with the 1 m side length and one cube box with the 0.5 m side length. Let box 1 to box 6 are six cube boxes with the 1 m side length and box 7 is the cube box with the 0.5 m side length. After the example is solved with Algorithm 1, we will get the solution with objective value 1.5 as shown in Figure 1.8.

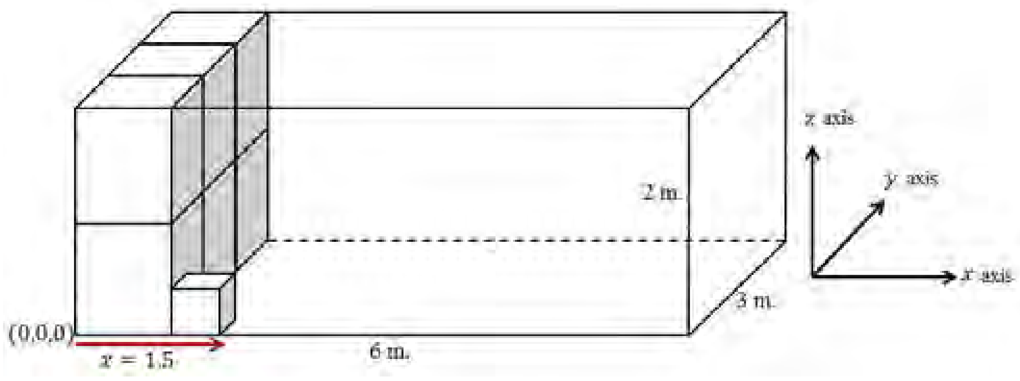


Figure 1.8: The graphical representation of a solution to the small example given by Algorithm 1.

1.4.5.3 Key to design a box by applying the container loading problem

In this subsection, we will describe how we can apply the CLP to design a set of boxes for packing goods. We have known that the CLP is to pack a given set of rectangular boxes into a given rectangular container. If we have an arrangement of boxes in the container, we can specify the rectangular required space for packing all boxes with this arrangement. The rectangular required space for packing all boxes is a subspace within the container and its left-front-bottom corner is also at the origin. For example, consider the solution in Figure 1.8, which shows the arrangement of 7 boxes in the given container with the length $\overline{x} = 6$ m, the width $\overline{y} = 3$ m, and the height $\overline{z} = 2$ m. Therefore, the magnitude along the x-axis of the rectangular required space for packing all boxes of length $x = 1.5$ m. The magnitude along the y-axis of the rectangular required space for packing all boxes of width $y = 3$ m, and the magnitude along the z-axis of the rectangular required space for packing all boxes of height $z = 2$ m. The volume of the rectangular space for packing all boxes is $1.5 \times 3 \times 2$ m^3 as shown in Figure 1.9 where the rectangular required space is indicated with dashed lines.

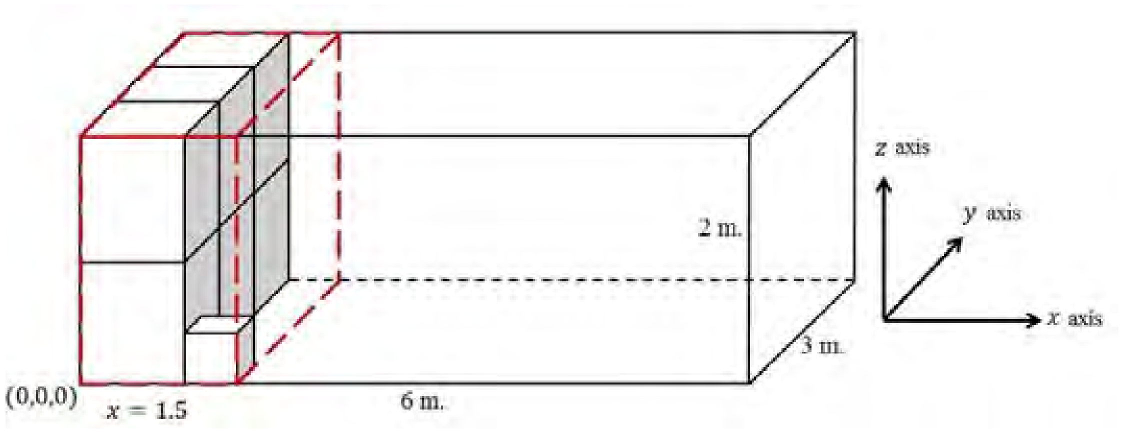


Figure 1.9: The graphical representation of the rectangular required space for packing all boxes.

From the CLP, we will apply it to design a set of boxes for packing goods in this thesis as described below.

1. We view the boxes that are packed into the container as rectangular goods that we want to design a box for packing.

2. We view the rectangular required space for packing all boxes in the CLP (or goods in our problem) as a box that we can design for packing goods. The rectangular required space or the box that we can design has the coordinate left-front-bottom corner at the origin in 3D coordinate system as the given container in the CLP. For example, the box that we can design for Example 2 with the arrangement in Figure 1.9 will have a dimension $1.5 \times 3 \times 2$.

3. We view the given container with the length $\overline{x}$, the width $\overline{y}$, and the height $\overline{z}$ as the bound space of the box that we want to design for packing our goods. In other words,

   - The given magnitude $\overline{x}$ along the x-axis is the magnitude upper bound of the side of the rectangular required space for packing all goods along the x-axis, or magnitude upper bound of the side of the box that we will design along the x-axis.
   - The given magnitude $\overline{y}$ along the y-axis is the magnitude upper bound of the side of the rectangular required space for packing all goods along the y-axis, or magnitude upper bound of the side of the box that we will design along the y-axis.
   - The given magnitude $\overline{z}$ along the z-axis is the magnitude upper bound of the side of the rectangular required space for packing all goods along the z-axis, or magnitude upper bound of the side of the box that we will design along the z-axis.

4. For a box that is designed, the measurement of the designed box are as follows.

   - The height of the designed box is always the only dimension of the rectangular required space for packing goods measured along the z-axis. This is because the goods must be packed right side up in the box by the assumption of this thesis.

- The length of the designed box is the longest side of the rectangular required space for packing goods along the x-axis and the y-axis.
- The width of the designed box is the shortest side of the rectangular required space for packing goods along the x-axis and the y-axis.

For example, consider the box we design in Example 2 with the dimension $1.5 \times 3 \times 2$. The length is 3, which is measured along the y-axis. The width is 1.5, which is measured along the x-axis and the height is 2.

However, the shape of a box that we want to design for packing goods in this thesis must be close to a cube as much as possible. Moreover, all items of goods that are packed in the designed box must be of the same kind. Therefore, the methodology from the CLP must be modified to make it fit our requirements. The methodology for our problem will be explained in Chapter 3.

1.5 Overview of thesis

This thesis consists of five chapters. Chapter 1 provides an introduction to this research study which includes motivation, objectives, assumptions, and background knowledge. The background knowledge has 5 sections, i.e., the definition of a box, linear programming, either-or constraints, heuristics, and the container loading problem. Chapter 2 is the literature review which relates to the container loading problem, the packing problem, the bin packing problem, and the knapsack loading problem. Chapter 3 is the methodology which is divided into 4 sections. In the first section, the nonlinear optimization model for designing boxes is introduced. We proposed a heuristic algorithm for solving the model for designing boxes in the second section. The third section presents a criteria for reducing the number of box types and a heuristic algorithm for minimizing the number of box types. The last section presents a binary integer linear programming (BILP) for minimizing the number of types of packing boxes based on the criteria in the third section. Numerical examples are shown in Chapter 4. Finally, the conclusions of this research study are in Chapter 5.

CHAPTER II

LITERATURE REVIEW

The container loading problem (CLP) appears in many related studies such as the packing problem [2, 3, 4], the knapsack loading problem (KLP) [5, 7, 8, 9, 10], and the bin packing problem (BPP) [12].

In 1995, a mathematical model of the CLP was first proposed by Chen et al. [1]. They proposed a mixed integer linear programming model (MILP) for the CLP. The result showed that the model can be solved for small size problems. Their study uses many binary variables to formulate the model which requires to heavy computation.

Tsai and Li [2] considered the CLP in other version which is called the packing problem. They adapted the MILP of Chen et al. [1] using considerably fewer binary variables and presented a mixed integer nonlinear programming which aims to pack a given set of boxes into a container with minimum volume. They used piecewise linearization technique to find the global optimum of the packing problem. The result showed that their method can find a global optimum within a tolerable error. Later, Tsai et al. [3] reformulated the nonlinear packing problem into a MILP using an improved piecewise linearization technique and logarithmic transformations. This approach can reduce the number of binary variables and constraints; and consequentially enhances the computational efficiency. On the other hand, Hu et al. [4] developed a novel method for solving the packing problem. They converted the nonlinear objective function in the packing problem into an increasing function with single variable and two fixed parameters. The new problem becomes a linear program which is easier to find a global optimum.

As for the BPP, Paquay et al. [12] proposed a mixed integer linear programming model of the BPP with additional constraints that encountered in the real world. They proposed the model of the BPP with the constraints that met in the air cargo industry. This specific application involves new constraints such as the stability and the fragility of cargo.

However, the CLP and the related problems are NP-hard. Only few exact methods have been suggested in the literature and it may not find an optimal solution in an acceptable time. Most researches have focused on the development of heuristic algorithms. Pisinger [5] proposed a heuristic algorithm to decompose the KLP into a number of layers which are divided into a number of strips. The decomposed sub-problem of the packing strips becomes the well-known knapsack problem. Bortfeldt and Mack [6] offered a heuristic algorithm derived from the layer building approach that was proposed by Pisinger [5] for solving the CLP. Eley [7] presented a greedy heuristic algorithm that are improved by a tree search for solving the KLP. Other heuristic algorithms such as genetic algorithms (GA) [8, 9 ,10] are used to solve the KLP. In Karabulut and Inceoglu [8], GA was used to solve the KLP with the deepest bottom left with fill. They define the length, the width, and the height of the container are along the z-axis, the x-axis, and the y-axis, respectively. An object will be firstly moved to the deepest available position (smallest z value) in the container, and then as far as to the bottom (smallest y value) in the second, and then as far as possible to the left (smallest x value) in the third. Kang et al. [9] presented the improved deepest bottom left with fill algorithm which utilizes a hybrid GA to solve the KLP. On the other hand, Goncalves and Resende [10] presented a multi-population biased random key genetic algorithm for solving the KLP. They used a maximal space representation to manage the free space in the container. In 2016, Huang et al. [11] presented a simple but effective heuristic algorithm for solving the CLP. The results of their algorithm showed that it is capable for solving the large size problems with more than two hundred boxes. The experiment results showed that their developed heuristic algorithm is more efficient than existing heuristic algorithms.

In this thesis, we modify the model of the CLP in [1, 11] and the heuristic algorithm for solving the container loading problem in [11] to design a box for packing each kind of goods. We view packing boxes that are loaded in the CLP as goods and view the space in the given container as the upper bound space of the box that we want to design as explained in Section 1.4.5.3. The space that is required for packing all goods is viewed as the designed box. We want to design a box to be close to a cube as much as possible for packing goods. Furthermore, we will propose a method as a heuristic algorithm and a mathematical model as a binary integer linear programming for minimizing the number of types of packing boxes.

CHAPTER III

METHODOLOGY

3.1 The model for designing boxes

In this section, we state the problem statement of our designing boxes problem and a model for designing boxes as follows. By assuming that we want to design a box for packing a given number of the goods of the same kind, we would like to design a box with the shape close to a cube as much as possible within the given bound of each side of the box. We try to design a box close to a cube since a cube box is more stable than other rectangular boxes. Moreover, each item must be packed right side up in the box. In addition, the volume utilization rate (VU rate) of the designed box, which is defined as the ratio of the sum of all volumes of items to the volume of box, should be relatively large because this also leads to high stability. For the model for designing boxes, it can be modified from the model of the CLP. We will view the boxes that are loaded into the given container (in the CLP) as goods (in our problem) and view the size of the given container (in the CLP) as the given bound of each side of the box that we want to design (in our problem). We change the objective function and add a constraint about the VU rate to the model of the CLP. The parameters and variables used in the model for designing boxes are defined below.

Parameters:

n	Total number of the given goods of the same kind to be packed in the same box.
N	The index set of the goods, $N = \{1, 2, ..., n\}$.
p, q, r	The length, the width, and the height of the goods, respectively.
$\overline{x}, \overline{y}, \overline{z}$	Upper bound of the magnitude of the side of the box to be designed along the x-, y-, and z-axes, respectively.

M	A large positive number, $M = max\{\overline{x}, \overline{y}, \overline{z}\}$.
V_u	Parameters indicating the lower bound of the volume utilization rate of the box to be designed, $V_u \in [0, 1]$.

Henceforth, we will call a rectangular space which has the left-front-bottom corner at (0,0,0) and the right-behind-top corner at $(\overline{x}, \overline{y}, \overline{z})$ as the *bound space.*

Variables:

x, y, z	Variables indicating the magnitude along the x-, y-, and z-axes of the required space for packing all n items of the goods into the bound space, respectively. The value $\max\{x, y\}$ is referred to as the length of the designed box, the value $\min\{x, y\}$ is referred to as the width, and the value z is referred to as the height of the designed box.
(x_i, y_i, z_i)	Variables indicating the coordinates of the left-front-bottom corner of item i.
l_{xi}, l_{yi}, l_{zi}	Binary variables indicating whether the length of item i is parallel to the x-axis, the y-axis, or the z-axis, respectively. For example, $l_{xi} = 1$ if the length of item i is parallel to the x-axis; otherwise $l_{xi} = 0$.
w_{xi}, w_{yi}, w_{zi}	Binary variables indicating whether the width of item i is parallel to the x-axis, the y-axis, or the z-axis, respectively. For example, $w_{xi} = 1$ if the width of item i is parallel to the x-axis; otherwise $w_{xi} = 0$.
h_{xi}, h_{yi}, h_{zi}	Binary variables indicating whether the height of item i is parallel to the x-axis, the y-axis, or the z axis, respectively. For example, $h_{xi} = 1$ if the height of item i is parallel to the x-axis; otherwise $h_{xi} = 0$.
$(\alpha_{ij}, \beta_{ij}, \delta_{ij})$	Binary variables indicating the relative positions of item i and item j, i.e., (a) $(\alpha_{ij}, \beta_{ij}, \delta_{ij}) = (0, 0, 1)$ if item i is on the left of item j. (b) $(\alpha_{ij}, \beta_{ij}, \delta_{ij}) = (0, 1, 0)$ if item i is on the right of item j. (c) $(\alpha_{ij}, \beta_{ij}, \delta_{ij}) = (1, 0, 0)$ if item i is in front of item j. (d) $(\alpha_{ij}, \beta_{ij}, \delta_{ij}) = (0, 1, 1)$ if item i is behind item j.

(e) $(\alpha_{ij}, \beta_{ij}, \delta_{ij}) = (1, 0, 1)$ if item i is below item j.

(f) $(\alpha_{ij}, \beta_{ij}, \delta_{ij}) = (1, 1, 0)$ if item i is above item j.

The model for designing boxes can be stated as follows:

$$
\begin{array}{llll}
\min & \max\{x, y, z\} - \min\{x, y, z\} & & \\
\text{s.t.} & x_i + pl_{xi} + qw_{xi} + rh_{xi} \leq x_j + M(1 + \alpha_{ij} + \beta_{ij} - \delta_{ij}) & \forall i, j \in N, i < j, & (1) \\
& x_j + pl_{xj} + qw_{xj} + rh_{xj} \leq x_i + M(1 + \alpha_{ij} - \beta_{ij} + \delta_{ij}) & \forall i, j \in N, i < j, & (2) \\
& y_i + pl_{yi} + qw_{yi} + rh_{yi} \leq y_j + M(1 - \alpha_{ij} + \beta_{ij} + \delta_{ij}) & \forall i, j \in N, i < j, & (3) \\
& y_j + pl_{yj} + qw_{yj} + rh_{yj} \leq y_i + M(2 + \alpha_{ij} - \beta_{ij} - \delta_{ij}) & \forall i, j \in N, i < j, & (4) \\
& z_i + pl_{zi} + qw_{zi} + rh_{zi} \leq z_j + M(2 - \alpha_{ij} + \beta_{ij} - \delta_{ij}) & \forall i, j \in N, i < j, & (5) \\
& z_j + pl_{zj} + qw_{zj} + rh_{zj} \leq z_i + M(2 - \alpha_{ij} - \beta_{ij} + \delta_{ij}) & \forall i, j \in N, i < j, & (6) \\
& 1 \leq \alpha_{ij} + \beta_{ij} + \delta_{ij} \leq 2 & \forall i, j \in N, i < j, & (7) \\
& x_i + pl_{xi} + qw_{xi} + rh_{xi} \leq x & \forall i \in N, & (8) \\
& y_i + pl_{yi} + qw_{yi} + rh_{yi} \leq y & \forall i \in N, & (9) \\
& z_i + pl_{zi} + qw_{zi} + rh_{zi} \leq z & \forall i \in N, & (10) \\
& l_{xi} + l_{yi} + l_{zi} = 1 & \forall i \in N, & (11) \\
& w_{xi} + w_{yi} + w_{zi} = 1 & \forall i \in N, & (12) \\
& h_{xi} + h_{yi} + h_{zi} = 1 & \forall i \in N, & (13) \\
& l_{xi} + w_{xi} + h_{xi} = 1 & \forall i \in N, & (14) \\
& l_{yi} + w_{yi} + h_{yi} = 1 & \forall i \in N, & (15) \\
& l_{zi} + w_{zi} + h_{yi} = 1 & \forall i \in N, & (16) \\
& h_{zi} = 1 & \forall i \in N, & (17) \\
& \dfrac{n\ p\ q\ r}{x\ y\ z} \geq V_u & & (18)
\end{array}
$$

$x_i, y_i, z_i \geq 0, 0 \leq x \leq \overline{x}, 0 \leq y \leq \overline{y}, 0 \leq z \leq \overline{z}; l_{xi}, l_{yi}, l_{zi}, w_{xi}, w_{yi}, w_{zi}, h_{xi}, h_{yi}, h_{zi},$ $\alpha_{ij}, \beta_{ij},$ and δ_{ij} are binary variables.

The objective function of this model is to minimize the magnitude difference between the longest side and the shortest side of the required space for packing all n items of goods. This makes the required space for packing goods or the designed box has the shape near a cube as much as possible as we desire. Constraints (1)–(7) ensure that all items in the box cannot overlap. Constraints (8)–(10) ensure that all items are packed within the designed box and cannot penetrate the surface of the box. Constraints (11)–(16) ensure that the item faces must be parallel or perpendicular to the faces of the designed box. The constraints (1)–(16) are the same as the constraints in the CLP (page 11) and they can be explained similarly. Constraint (17) enforces that all items are packed right side up into the box according to the assumption. Constraint (18) enforces that the VU rate of the designed box must be at least the value that is given by the user. Actually, this model has higher complexity than the model of the CLP since the constraint (18) has the nonlinear term xyz. Therefore, a heuristic algorithm for solving it is necessary and is proposed in the next section.

3.2 A heuristic algorithm for designing boxes

In this section, a heuristic algorithm is proposed for designing boxes based on the model for designing boxes in Section 3.1. This algorithm is modified from the heuristic algorithm for solving the CLP in [11]. Let us we state our problem again. Assume that we have n items of a kind of goods and have the given magnitude bound of each side of the box that we want to design for packing all n items of goods. We want to design a box (the space that requires for packing all goods in the given bound space) with the shape as close to a cube as possible and the volume utilization rate of the designed box must be at least a value given by a user as described earlier. Each item of goods also must be packed right side up in the box that we want to design. The parameters used in this heuristic algorithm for designing boxes consist of the following.

T	The max number of iterations.
n	The number of goods of the same kind to be packed in the same box.
N	The index set of goods, $N = \{1, 2, ..., n\}$.
N_l	The set of indices of all loaded items, $N_l \subseteq N$.
l, w, h	The length, the width, and the height of the goods, respectively.
(p_i, q_i, r_i)	The magnitude of the side which is parallel to the x-, y-, and z-axes of item i (depending on the rotation of the item), respectively, $i \in N$.
(x_i, y_i, z_i)	The coordinate of the left-front-bottom corner of item i, $i \in N$.
P	The $n \times 6$ matrix which represents the packing pattern of goods. Each row i of P collects p_i, q_i, r_i and the coordinate of the left-front-bottom corner of item i. At the beginning, P is given by $P = \begin{bmatrix} p_1 & q_1 & r_1 & x_1 & y_1 & z_1 \\ & \vdots & & & \vdots & \\ p_n & q_n & r_n & x_n & y_n & z_n \end{bmatrix} = \begin{bmatrix} l & w & h & x_1 & y_1 & z_1 \\ & \vdots & & & \vdots & \\ l & w & h & x_n & y_n & z_n \end{bmatrix},$ where $x_i, y_i, z_i, \forall i \in N$ can be arbitrary value initially. Typically, we set $x_i = y_i = z_i = -1, \forall i \in N$ at the beginning to indicate that item i is not loaded into the box yet.
$\overline{x}, \overline{y}, \overline{z}$	Upper bound of the magnitude of the side of the box to be designed along the x-, y-, and z-axes, respectively. For simplicity, we will call a rectangular space which has the left-front-bottom corner at (0,0,0) and the right-behind-top corner at $(\overline{x}, \overline{y}, \overline{z})$ as the bound space.
x, y, z	The magnitude along the x-, y-, and z-axes of the rectangular required space for packing all n items of the goods, respectively. The value $\max\{x, y\}$, $\min\{x, y\}$, and z are referred to as the length, the width, and the height of the designed box, respectively.
f	The objective value of the model for designing boxes, i.e., $f = \max\{x, y, z\} - \min\{x, y, z\}$.
V_u	The lower bound of the volume utilization rate of the designed box which is given by a user.

According to the heuristic algorithm for solving the CLP in Huang et al. [11], they propose the condition for checking whether any two rectangular objects overlap in the container which is called the *non-overlapping condition* and the condition for checking whether each object i is positioned inside the container which is called the *non-overstepping condition.* These conditions will be used in this algorithm by viewing the items and the bound space as the rectangular objects and the container, respectively.

1. The non-overlapping condition.

For any two items i and j, they will overlap if and only if the truth value of Statement (3.1) is false,

$$(x_i{+}p_i \leq x_j) \vee (x_j{+}p_j \leq x_i) \vee (y_i{+}q_i \leq y_j) \vee (y_i{+}q_j \leq y_i) \vee (z_i{+}r_i \leq z_j) \vee (z_j{+}r_j \leq z_i). \tag{3.1}$$

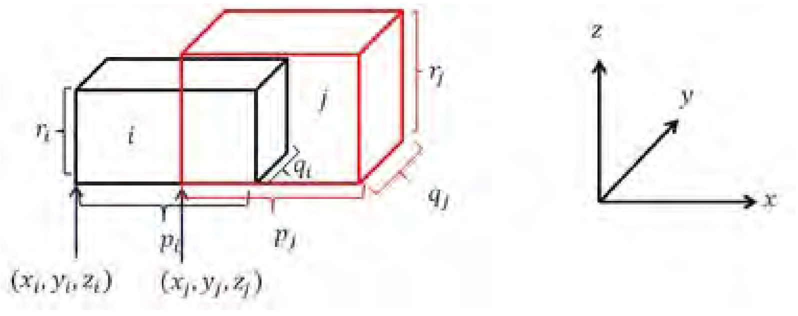


Figure 3.1: An example of two overlapping items.

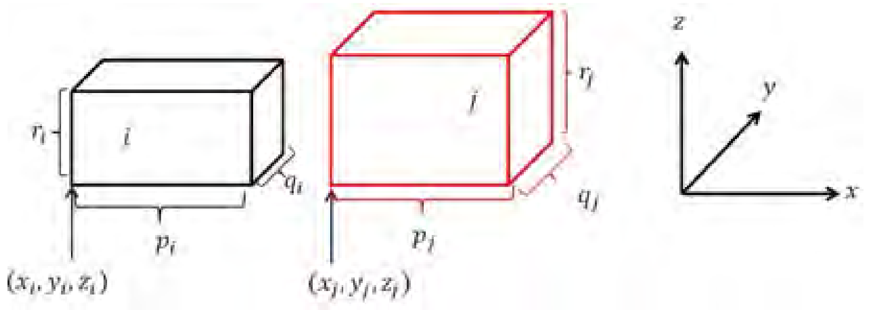


Figure 3.2: Box i is on the left of box j with $x_i + p_i \leq x_j$.

Figure 3.1 illustrates the overlapping of two items i and j. It is easy to see that all 6 conditions in Statement (3.1) are violated. But if at least one condition in Statement (3.1) holds, it guarantee that items i and j will not overlap. For example, if the first condition $x_i + p_i \leq x_j$ in Statement (3.1) hold, it means that item i is on the left of item

j as shown in Figure (3.2)

2. The non-overstepping condition.

Each item i will be positioned inside the bound space if it satisfies all 3 conditions in Statement (3.2),

$$(x_i + p_i \leq \overline{x}) \quad \wedge \quad (y_i + q_i \leq \overline{y}) \quad \wedge \quad (z_i + r_i \leq \overline{z}). \tag{3.2}$$

The main idea of the proposed heuristic for designing boxes is as follows. All items will be loaded one by one into the bound space where the first item will be loaded into the bound space by setting its coordinate of the left-front-bottom corner at (0,0,0). Then the remaining items $i = 2$ to n will be loaded one by one into the bound space such that the magnitude of the required space for packing goods along the x-, y-, and z-axes should expand approximately in the same rate (so that it is still as close to a cube as possible). The heuristic algorithm for designing boxes is shown in Algorithm 2. The inputs of the algorithm include the number of iteration T, the number of goods to be packed n, the upper bounds of the dimensions of the designed box $\overline{x}, \overline{y}, \overline{z}$, the lower bound of the VU rate of the designed box V_u, and the dimensions of goods l, w, h. First, P is initialized by setting $p_i = l, q_i = w, r_i = h$ and $x_i = y_i = z_i = -1$, $\forall i \in N$. The best packing pattern P^* and the VU rate of the best packing pattern V^* are set to be null. The value f^* which implies the objective value of the best packing pattern is initially set to be infinity. For each iteration $r = 1$ to T, the first item will be positioned at (0,0,0) in the bound space and set $N_l = \{1\}$. Then for each item $i = 2$ to n, we will find indices i_1, i_2, and i_3 from N_l (Algorithm 3). The indices i_1, i_2, and i_3 are defined as follows.

- The index i_1 is the index of a loaded item that has the smallest distance along the x-axis from the origin to its right-front-bottom corner. Moreover, item i_1 must be such that item i can be loaded into the bound space by attaching the left-front-bottom corner of item i to the right-front-bottom corner of item i_1 without overlapping with other loaded items as well as staying within the bound space. (If

Algorithm 2 Heuristic algorithm for designing boxes

```
    Input : $T,\ n,\ \overline{x},\overline{y},\overline{z},\ V_u,\ l,w,h$
 1: Set $P = \begin{bmatrix} p_1 & q_1 & r_1 & x_1 & y_1 & z_1 \\ & \vdots & & & \vdots & \\ p_n & q_n & r_n & x_n & y_n & z_n \end{bmatrix}, p_i = l, q_i = w, r_i = h, x_i = y_i = z_i = -1,$
        $\forall i \in N,\ P^* = null,\ f^* = \infty,\ V^* = null.$
 2: for $r = 1$ to $T$ do
 3:   if $p_1 \le \overline{x}$ and $q_1 \le \overline{y}$ and $r_1 \le \overline{z}$ then
 4:     Set $x_1 = y_1 = z_1 = 0$ and $N_l = [1]$
 5:     for $i = 2$ to $n$ do
 6:       Do Find indices and obtain $i_1, i_2, i_3$ (See Algorithm 3.)
 7:       if $i_1 = i_2 = i_3 = null$ then
 8:         break
 9:       else
10:         Set $Candidate = \{x_{i_1} + p_{i_1}, y_{i_2} + q_{i_2}, z_{i_3} + r_{i_3}\}$
11:         Set $low = \min(Candidate)$. If there are more than one element in
            $Candidate$ list which equal to the minimum, each of them may be
            $low$ with equal probability.
12:         if $low == x_{i_1} + p_{i_1}$ then
13:           Set $x_i = x_{i_1} + p_{i_1}, y_i = y_{i_1}$ and $z_i = z_{i_1}$
14:         else if $low == y_{i_2} + q_{i_2}$ then
15:           Set $x_i = x_{i_2}, y_i = y_{i_2} + q_{i_2}$ and $z_i = z_{i_2}$
16:         else
17:           Set $x_i = x_{i_3}, y_i = y_{i_3}$ and $z_i = z_{i_3} + r_{i_3}$
18:         end if
19:         Set $N_l = N_l \cup \{i\}$
20:       end if
21:     end for
22:   end if
23:   if length$(N_l) == n$ then
24:     Set $x = \max_{i \in N}\{x_i + p_i\},\ y = \max_{i \in N}\{y_i + q_i\},\ z = \max_{i \in N}\{z_i + r_i\}$
25:     Set $f = \max\{x,y,z\} - \min\{x,y,z\},\ V = \dfrac{n\ l\ w\ h}{x\ y\ z}$
26:     if $f < f^*$ and $V \ge V_u$ then
27:       Set $f^* = f, P^* = P, x^* = x, y^* = y, z^* = z$ and $V^* = V$
28:     end if
29:   end if
30:   for $i = 1$ to $n$ do
31:     $a$ = random number in [0,1]
32:     if $a > 0.5$ then
33:       Set $(p_i, q_i) = (q_i, p_i)$
34:     end if
35:   end for
36: end for
37: return $P^*, x^*, y^*, z^*, V^*, f^*$
```

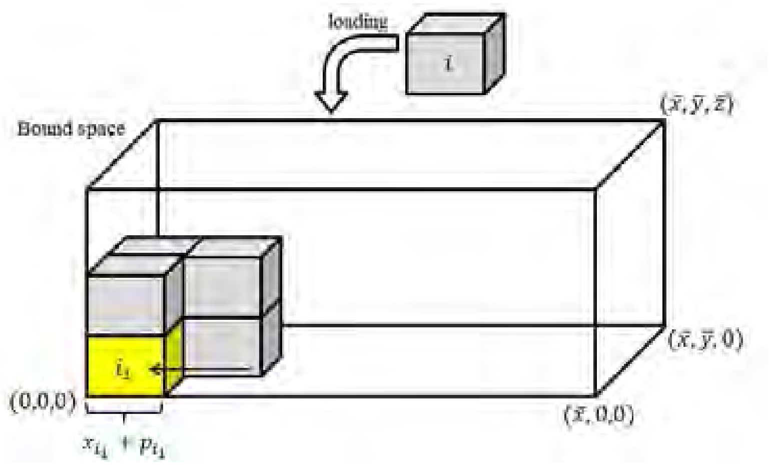
Algorithm 3 Find indices.

```
1: Define $x_i = x_{i'} + p_{i'},\ y_i = y_{i'}$ and $z_i = z_{i'}$ where $i' \in N_l$ to be used in (3.1) and (3.2).
2: Find $i_1 = \underset{i' \in N_l}{\text{argmin}}\{x_{i'} + p_{i'} | (3.1) \equiv True, \forall j \in N_l - \{i'\}$ and $(3.2) \equiv True\}$.
   If more than one item can be $i_1$, select the item with the smallest $z_{i_1}$.
3: if $i_1 = null$ then
4:    Set $x_{i_1} + p_{i_1} = \infty$
5: end if
6: Define $x_i = x_{i'},\ y_i = y_{i'} + q_{i'}$ and $z_i = z_{i'}$ where $i' \in N_l$ to be used in (3.1) and (3.2).
7: Find $i_2 = \underset{i' \in N_l}{\text{argmin}}\{y_{i'} + q_{i'} | (3.1) \equiv True, \forall j \in N_l - \{i'\}$ and $(3.2) \equiv True\}$.
   If more than one item can be $i_2$, select the item with the smallest $z_{i_2}$.
8: if $i_2 = null$ then
9:    Set $y_{i_2} + q_{i_2} = \infty$
10: end if
11: Define $x_i = x_{i'},\ y_i = y_{i'}$ and $z_i = z_{i'} + r_{i'}$ where $i' \in N_l$ to be used in (3.1) and (3.2).
12: Find $i_3 = \underset{i' \in N_l}{\text{argmin}}\{z_{i'} + r_{i'} | (3.1) \equiv True, \forall j \in N_l - \{i'\}$ and $(3.2) \equiv True\}$.
13: if $i_3 = null$ then
14:    Set $z_{i_3} + r_{i_3} = \infty$
15: end if
```

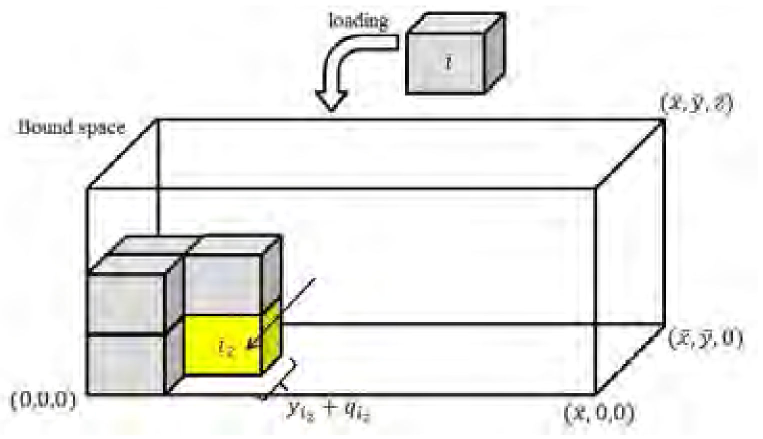
more than one item can be item i_1, we firstly choose the item that has z_{i_1} the smallest so that item i will be supported by the floor of the bound space or other packed item. See an example in Figure 3.3 (a))

- The index i_2 is the index of a loaded item that has the smallest distance along the y-axis from the origin to its left-behind-bottom corner. Moreover, item i_2 must be such that item i can be loaded into the bound space by attaching the left-front-bottom corner of item i to the left-behind-bottom corner of item i_2 without overlapping with other loaded items as well as staying within the bound space. (If more than one item can be item i_2, we firstly choose the item that has z_{i_2} the smallest so that item i will be supported by the floor of the bound space or other packed item. See an example in Figure 3.3 (b))
- The index i_3 is the index of a loaded item that has the smallest distance along the z-axis from the origin to its left-front-top corner. Moreover, item i_3 must be such that item i can be loaded into the bound space by attaching the left-front-bottom corner of item i to the left-front-top corner of item i_3 without overlapping with other loaded items as well as staying within the bound space (See an example in Figure 3.3 (c)).

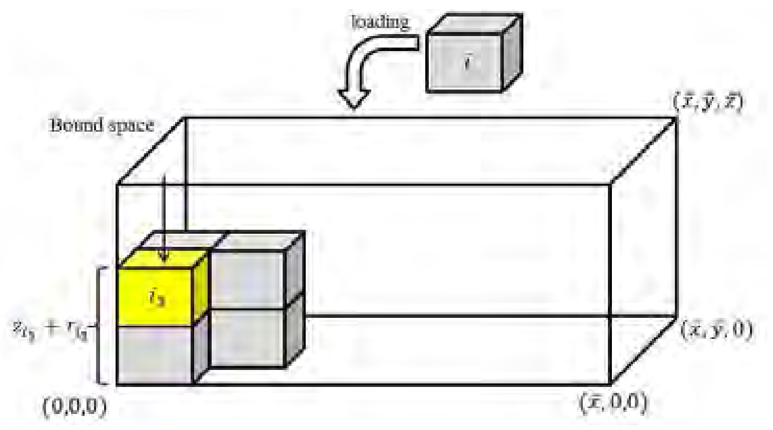
If we cannot find all valid indices i_1, i_2 and i_3, the iteration r will be broken since the remaining item i to item n cannot be loaded into the bound space. Otherwise, we will consider that item i should be loaded. We compare the distances $x_{i_1} + p_{i_1}, y_{i_2} + q_{i_2}$, and $z_{i_3} + r_{i_3}$ and set the smallest distance as '*low*'. If there are more than one '*low*' value, we will randomly choose one of them. For example, if $x_{i_1} + p_{i_1} = y_{i_2} + q_{i_2} = z_{i_3} + r_{i_3} < \infty$, then each value can be *low* with probability $\frac{1}{3}$. Then we will load item i by attaching it to one of items i_1 (along the x-axis), i_2 (along the y-axis), or i_3 (along the z-axis) depending on which one is corresponding to *low*. For example, in the situation in Figure 3.3, item i will be loaded by attaching it to item i_1 along the x-axis as shown in Figure 3.4. It means that we always firstly load item i along the direction that has the smallest magnitude of the occupied space. This make the magnitude of the required space for packing goods along the x-, y-, and z-axes expand in approximately the same rate and it makes the



(a) An example of finding index i_1.



(b) An example of finding index i_2.



(c) An example of finding index i_3.

Figure 3.3: An example of finding index i_1, i_2, and i_3.

designed box has the shape close to a cube as we desire. When there are more than one value that can be *low*, we randomly pick one. If all n items can be loaded into the bound space, the values x, y, z (which are the magnitude of the required space for packing all goods along the x-, y-, and z-axes, respectively), the value f (which is the objective value), and the value V (which is the VU rate of the designed box) of iteration r can be computed. If $f < f^*$ and the VU rate of the designed box in this iteration is at least the minimum VU rate which can be accepted by the user, the value f^*, P^*, x^*, y^*, z^* and V^* are updated. Then the value of p_i and q_i for each $i \in N$ can be swapped with probability 0.5 to generate different rotation of each item of goods. The value of h is fixed in the 3^{rd} column of P since each item must be packed right side up into the designed box by the assumption. The items are then reloaded into the bound space in the next iteration and this essentially yields a different packing pattern. After iteration $r = T$, the algorithm will terminate and return P^*, x^*, y^*, z^*, V^* and f^*. The length, width, and height of the designed box are $\max\{x^*, y^*\}, \min\{x^*, y^*\}$ and z^*, respectively. The objective value is f^*, the VU rate of the designed box is V^*, and the arrangement of goods in the box can be derived from P^*.

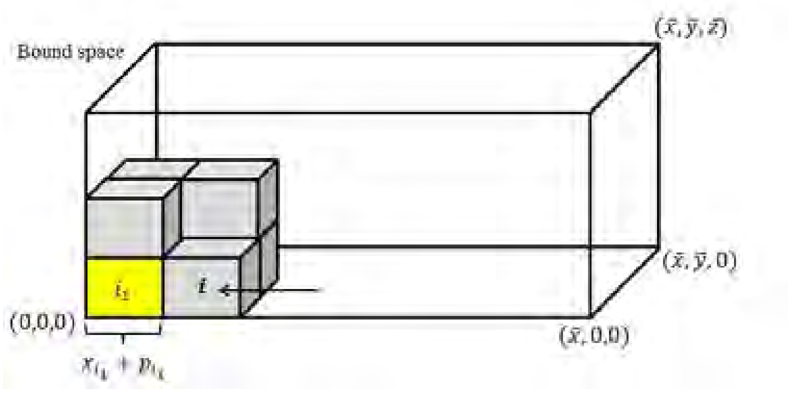


Figure 3.4: Box i adjoining box i_1.

The proposed heuristic algorithm in this section can be used to design a box's dimensions for packing a fixed number of the same kind of goods in to one box which is the first objective of this thesis. Next, we will discuss the second objective of this thesis which is to propose a method for minimizing the number of types of packing box in the next section.

3.3 A heuristic algorithm for minimizing number of types of packing boxes

In product packing, reducing some box types for packing goods when we have several types of boxes can increase efficiency and reduce the packing cost. In this section, we propose appropriate criteria to reduce the number of box types and propose a method which is a heuristic algorithm for minimizing the number of types of packing boxes. The problem statement can be described as follows. Suppose that we have a set of different box types for packing goods. The objective is to minimize the number of box types for packing goods as many as possible under the criteria that a particular box type could be substituted by a bigger box type if the size difference (length, width, height) in percentage of the larger box stays within a specific bound which is given by a user. It means that the smaller box can be discarded and the bigger box will be used instead. In other words, the objective is to find the minimum number of box types after indicating the discarding boxes and knowing the replacement boxes.

Assume that we have n different box types for packing. Let L_i, W_i, and H_i be the length, the width, and the height respectively of box $i, i \in \{1, 2, ..., n\}$. For simplicity, we will call "box type i" as "box i" and "boxes of different sizes" as "boxes". From the criteria in the problem statement, box j can be legitimately substituted by box i if and only if the following 5 conditions are met:

1. Box i is sidewise longer than box j. This makes sense because if this condition is true, all items that are packed in box j can be certainly in box i. In other words, all 3 conditions in Statement (3.3) must hold:

$$(L_i \geq L_j) \ \wedge \ (W_i \geq W_j) \ \wedge \ (H_i \geq H_j). \tag{3.3}$$

2. The difference of each side does not exceed a certain percentage (given by a user) of the larger one, say $100t\%$. This enforces the substituting box to be not too much larger than the discarded box. In other word, all 3 conditions in Statement (3.4)

must hold:

$$\left(\frac{L_i - L_j}{L_i} \leq t\right) \wedge \left(\frac{W_i - W_j}{W_i} \leq t\right) \wedge \left(\frac{H_i - H_j}{H_i} \leq t\right). \tag{3.4}$$

3. For any box which is discarded, there is only one box substitutes it.
4. If a box is discarded, then it cannot substitute other boxes.
5. There is no double replacement that violates the size percentage difference bound. For example, if box i substitutes box j and box k substitutes box i, it seems that we can reduce 2 box types and it potentially means that box k substitutes both box i and box j but the length difference in percentage between box k and box j may violate Condition 2.

In this section, we propose a heuristic algorithm based on these 5 conditions for minimizing the number of types of packing boxes. The parameters used in this algorithm consist of the following.

Num	The maximun number of iterations.
n	The total number of the given boxes.
N	The index set of boxes, $N = \{1, 2, ..., n\}$.
L_i, W_i, H_i	The length, width, and height of box $i, i \in N$.
T	The matrix $T = [T_{ij}]_{n\times n}$. Each row i of T is corresponding to box $i \in N$ and each column j of T is corresponding to box $j \in N$. The element T_{ij} is equal to 1 if box i substitutes box j. At the beginning, T is initialized as $T = [-1]_{n\times n}$.
t	Parameter implying the maximum percentage difference of each box side of any 2 boxes that we accept to discard the smaller box. The parameter t is given by the user, where $t \in [0, 1]$.
f	The parameter implying the objective value of the problem; i.e., $f = n - \sum_{T_{ij}=1} T_{ij}$, the difference of n and sum of all T_{ij} such that $T_{ij} = 1$.

Algorithm 4 Heuristic algorithm for minimizing number of types of packing boxes

```
    Input : $Num,\ \ n,\ \ L_i, W_i, H_i,\ \ t$
 1: Set $T = [-1]_{n\times n},\ T^* = \phi,\ f^* = n.$
 2: **for** $i = 1$ to $n$ **do**
 3:     Set $T_{ii} = 0$
 4: **end for**
 5: **for** $i = 1$ to $n$ **do**
 6:     **for** $j = 1$ to $n$ **do**
 7:         **if** $L_i - L_j < 0$ or $W_i - W_j < 0$ and $H_i - H_j < 0$ **then**
 8:             Set $T_{ij} = 0$
 9:         **end if**
10:     **end for**
11: **end for**
12: **for** $i = 1$ to $n$ **do**
13:     **for** $j = 1$ to $n$ **do**
14:         **if** $L_i - L_j > tL_i$ or $W_i - W_j > tL_i$ and $H_i - H_j > tL_i$ **then**
15:             Set $T_{ij} = 0$
16:         **end if**
17:     **end for**
18: **end for**
19: Set $T^d = T$
20: **for** $m = 1$ to $Num$ **do**
21:     **for** $j = 1$ to $n$ **do**
22:         **if** There is no $-1$ in column $j$ of $T^d$ **then**
23:             **pass**
24:         **else**
25:             Let $I$ be the list of all indices of $i$ which $T^d_{ij} = -1$
26:             Let $a$ be a random number from $I$
27:             Set $T^d_{aj} = 1$ and $T^d_{kj} = 0, \forall k \neq a$
28:             Set $T^d_{jk} = 0, \forall k \in N$
29:             Set $T^d_{ka} = 0, \forall k \in N$
30:         **end if**
31:     **end for**
32:     Set $f = n - \sum_{\forall i \in N} \sum_{\forall j \in N} T^d_{ij}$
33:     **if** $f < f^*$ **then**
34:         Set $f^* = f,\ T^* = T^d,$
35:     **end if**
36:     Set $T^d = T$
37: **end for**
38: **return** $T^*, f^*$
```

The heuristic algorithm for minimizing the number of types of packing boxes is shown in Algorithm 4. The inputs of this algorithm include the maximum number of iterations Num, the total number of boxes n, the sizes of all boxes $L_i, W_i, H_i,\ \forall i \in N$, and the bound of the percentage difference for each side between a discarded box and its substituting box, t. The output are f^* and T^*, which are the best objective value and the best substitution matrix that have been found, respectively. First, the matrix T is initialized by setting each component as -1 as well as setting f^* as n and T^* as null. Next, we set all element $T_{ii} = 0, \forall i \in N$ because it is the same box. Then, we check each pair of boxes i and j against Statement (3.3) and (3.4). If Statement (3.3) does not hold for a pair of boxes i and j, the element T_{ij} in T will be assigned to be zero since box i is not bigger than box j in every dimensions. If Statement (3.4) does not hold for a pair of boxes i and j, the element T_{ij} in T will be assigned to be zero since the size difference in percentage of these two boxes does not stay in the given bound. At this point, the element T_{ij} is either 0 or -1, where -1 means box i can subsitute box j with respect to the conditions $1-2$ of the criteria for legitimate box substitution. Next, the algorithm will repeat the following solution generation procedure for Num iterations. In each iteration, T is duplicated as T^d and each column j in T^d which corresponds to box j is checked. If there is no the element -1 in column j of T^d, then we go check the next column because there is no box i which can substitute box j. Otherwise, we set I to be the list of all indices of the row i such that $T^d_{ij} = -1$ and randomly select an element of I, say a. We set T^d_{aj} to be 1 and set $T^d_{kj}, \forall k \neq a$ to be 0. It means that we randomly select one box, say box a, to replace box j and box j is discarded. Then T^d_{jk} is set to be zero for all $k \in N$ since box j must be discarded and, therefore, it cannot substitute other boxes. Furthermore T^d_{ka} is also set to be zero for all $k \in N$ so that no other box can substitute box a. This guarantees that there is no double replacement which may violate the assumption in Statement (3.4). After all columns are processed, each element of T^d is either 0 or 1. The element T^d_{ij} which is equal to 1 implies that we use box i instead of box j and it satisfies all 5 conditions for box substitution mentioned earlier. Next the objective value f of the current solution T^d is calculated. The value f^* and the matrix T^* are updated if $f < f^*$ and the solution generation procedure repeats. Finally, the algorithm returns f^* which is the total number of the remaining boxes after the substitution and T^* which gives the substitution information, i.e. which box is discarded and which box substitutes it.

Although the heuristic algorithm can find a good solution of a large size problem within an acceptable time, it cannot guarantee to find the optimal solution of the problem. In the next section, we present another way for minimizing the number of packing box types under the same criteria but can guarantee to find the optimal solution which is a binary integer linear programming model.

3.4 A binary integer linear programming model for minimizing number of types of packing boxes

In this section, we propose a binary integer linear programming model for minimizing the number of types of packing boxes which is the same problem described in Section 3.3. The parameters and variables used in the proposed model are defined below.

Parameters:

n	The total number of the given boxes.
N	The index set of boxes, $N = \{1, 2, ..., n\}$.
M	An arbitraly large positive number.
ε	An infinitesimally small positive number.
L_i, W_i, H_i	Parameter indicating the length, the width, and the height of box i, respectively.
t	Parameter implying the maximum percentage difference of each box side of any 2 boxes that we accept to discard the smaller box. The parameter t is given by the user, where $t \in [0, 1]$.

Variables:

T_{ij}	Binary variable which is equal to 1 if box i substitutes box j.
$y_{ij}, a_{ij}, b_{ij}, c_{ij}, d_{ij}$	Auxilary binary variables.

In this model, the variable T_{ij} is defined only when $i \neq j$. The proposed model can be stated as follows.

$$\begin{aligned}
\min \quad & n - \sum_{\substack{\forall i \in N}} \sum_{\substack{\forall j \in N \\ i \neq j}} T_{ij} \\
\text{s.t.} \quad & [|L_i - L_j| + |W_i - W_j| + |H_i - H_j|] - [(L_i - L_j) + (W_i - W_j) + (H_i - H_j)] \\
& \qquad \leq M(1 - y_{ij}) \quad \forall i, j \in N, i \neq j, && (1) \\
& T_{ij} \leq y_{ij} \quad \forall i, j \in N, i \neq j, && (2) \\
& (L_i - L_j) - t(L_i) \leq M(1 - a_{ij}) \quad \forall i, j \in N, i \neq j, && (3) \\
& T_{ij} \leq a_{ij} \quad \forall i, j \in N, i \neq j, && (4) \\
& (W_i - W_j) - t(W_i) \leq M(1 - b_{ij}) \quad \forall i, j \in N, i \neq j, && (5) \\
& T_{ij} \leq b_{ij} \quad \forall i, j \in N, i \neq j, && (6) \\
& (H_i - H_j) - t(H_i) \leq M(1 - c_{ij}) \quad \forall i, j \in N, i \neq j, && (7) \\
& T_{ij} \leq c_{ij} \quad \forall i, j \in N, i \neq j, && (8) \\
& \sum_{\substack{\forall i \in N \\ i \neq j}} T_{ij} \leq 1 \quad \forall j \in N, && (9) \\
& T_{ij} - 1 \leq M(1 - d_{ij}) - \epsilon \quad \forall i, j \in N, i \neq j, && (10) \\
& T_{jk} \leq d_{ij} \quad \forall i, j, k \in N, i \neq j \neq k, && (11) \\
& T_{ki} \leq d_{ij} \quad \forall i, j, k \in N, i \neq j \neq k, && (12)
\end{aligned}$$

$T_{ij}, y_{ij}, a_{ij}, b_{ij}, c_{ij}, d_{ij}$ are binary variables.

The objective function is to minimize the number of box types within the criteria that a particular box could be substituted by a bigger box if their relative size difference does not exceed $100t\%$. Constraints (1)–(2) ensure that if box i is not bigger than box j in all dimensions, then T_{ij} is forced to be zero. This is because if box i is bigger than box j, then $L_i - L_j \geq 0, W_i - W_j \geq 0$, and $H_i - H_j \geq 0$. The left-hand-side (LHS) of constraint (1) becomes 0 and the variable y_{ij} in constraint (1) can be 0 or 1 and so can T_{ij} by constraint (2). On the other hand, if there is one dimension of box i is not bigger than the corresponding dimension of box j, it means that $L_i - L_j < 0$ or $W_i - W_j < 0$ or $H_i - H_j < 0$. Then LHS of constraint (1) is positive and the variable y_{ij} in constraint (1) must be 0. Therefore, by constraint (2), T_{ij} is equal to 0. Constraints (3)–(4) ensure that if the length of box i is longer than the length of box j and the difference exceeds $100t\%$ of the larger box, then T_{ij} is forced to be zero. This is because if LHS of constraint (3) is positive, then the variable a_{ij} in constraint (3) can be only 0 and then T_{ij} is equal to 0 by constraint (4). On the other hand, if LHS of constraint (3) is not positive, then the variable a_{ij} in constraint (3) can be 0 or 1 and so can T_{ij} by constraint (4). However, if LHS of constraint (3) is negative but $L_i - L_j < 0$, it means that the length of box i is not large enough to replace box j. Hence, by constraints (1)–(2), T_{ij} is forced to be 0 eventually. Constraints (5)–(6) and (7)–(8) are similar to constraints (3)–(4), but they consider the width and height instead of the length, respectively. Constraint (9) ensures that if a box is discarded, it can be substituted by only one box. Constraints (10) and (11) ensure that if box j is discarded, then it cannot substitute the other boxes. This is because if T_{ij} is equal to 1 in constraint (10) or box j is discarded, then the variable d_{ij} in constraint (10) must be 0. Hence $T_{jk} = 0, \forall k \neq j$ by constraint (11). Constraints (10) and (12) enforce that if box i substitutes box j, then box i cannot be substituted by any box so that there is no double replacement which violates the assumption of the size percentage difference bound. This is because if T_{ij} is equal to 1 in constraint (10), the variable d_{ij} in constraint (10) can be only 0. Hence $T_{ki} = 0, \forall k \neq i$ by constraint (12).

In this model, the value of the large positive number M that ensure the suitable requirement in constraints (1),(3),(5),(7), and (10) can be set as $12 \max_{i \in N}\{L_i, H_i\}$. This is because the worst case or the maximum value of LHS of constraint (1) will occur when $L_i - L_j < 0, W_i - W_j < 0$, and $H_i - H_j < 0$. For simplicity, we let $A = \max_{i \in N}\{L_i, H_i\}$. Note that $L_i \geq W_i$ by assumption. Then,

$$\begin{aligned}
LHS &= |L_i - L_j| + |W_i - W_j| + |H_i - H_j| - (L_i - L_j) - (W_i - W_j) - (H_i - H_j),\\
&= 2|L_i - L_j| + 2|W_i - W_j| + 2|H_i - H_j|,\\
&\leq 2(|L_i| + |L_j|) + 2(|W_i| + |W_j|) + 2(|H_i| - |H_j|),\\
&\leq 2(A + A) + 2(A + A) + 2(A + A),\\
&= 12A.
\end{aligned}$$

If we use $M = 12 \max_{i \in N}\{L_i, H_i\}$, it is easy to see that this value is also large enough to eliminate constraints (3),(5),(7), and (10) when the binary variable a_{ij}, b_{ij}, c_{ij}, or d_{ij}

is equal to 0.

For the value of ε in constraint (10), we can use any value in the range (0,1). This is because we want the value of ε to be positive so that whenever the variable T_{ij} in constraint (10) is equal to 1, the variable d_{ij} in constraint (10) can be only 0. Moreover, we want ε to be small enough so that whenever T_{ij} in constraint (10) is equal to 0, the variable d_{ij} in constraint (10) can be either 0 or 1. Hence, ε needs to be less than 1. Therefore, we can set the value $0 < \varepsilon < 1$ in this model.

Numerical examples are given in the next Chapter to illustrate the application of our approaches.

CHAPTER IV

NUMERICAL EXPERIMENTS

In this chapter, numerical examples are generated which contain 50 kinds of goods as well as the upper bound of each dimension and the lower bound of the VU rate of the box to be designed for each kind of goods. Next, the heuristic algorithm for designing boxes in Section 3.2 is applied to the data to design a box for packing each kind of goods. Then we will minimize all designed boxes by using the proposed heuristic algorithm for minimizing the number of packing boxes in Section 3.3 and the BILP model for minimizing the number of packing boxes in Section 3.4 so that we will get minimal box types for packing all goods in the generated data. In addition, we will compare the efficiency of the heuristic algorithm for minimizing the number of packing boxes and the BILP model for minimizing the number of packing boxes.

4.1 The generated data set

The data used in our experiment are generated in the following steps.

1. The length (l), the width (w), and the height (h) of each kind of goods $k \in \{1, 2, ..., 50\}$ are randomly generated from small sizes to large sizes. The maximum size of the longest side of goods in this data set is no more than 50 cm.

2. The number of goods per box (n) of each kind of goods is randomly selected from 5 to 50 units. In addition, the parameter n for larger goods will be smaller than n for smaller goods since one unit of the goods with larger size has to use more the occupied space. Hence, the range for the random number of goods per box of each kind of goods $k, k \in \{1, 2, ..., 50\}$ depends on the maximum of the length, the width, and the height of the goods, or $\max\{l_k, w_k, h_k\}$, as shown in Table 4.1.

3. The upper bounds for the dimensions of the box to be designed (i.e., $\overline{x}, \overline{y}$, and $\overline{z}$)

Table 4.1**:** The criteria for defining the range for the random number of goods per box for goods of kind k, $k \in \{1, 2, ..., 50\}$.

Conditions	The range for the random number of goods per box of goods of kind k
$\max\{l_k, w_k, h_k\} < 10$	[5,50]
$10 \leq \max\{l_k, w_k, h_k\} < 20$	[5,40]
$20 \leq \max\{l_k, w_k, h_k\} < 30$	[5,35]
$30 \leq \max\{l_k, w_k, h_k\} < 40$	[5,20]
$40 \leq \max\{l_k, w_k, h_k\} \leq 50$	[5,15]

of each kind of goods $k \in \{1, 2, ..., 50\}$ are randomly selected within the range $[\max\{l_k, w_k, h_k\}, 100]$. We set the maximum to be 100 cm. so that the designed box is not too large.

4. The lower bounds of volume utilization rate (V_u) of the box to be designed for goods of kinds 1–30 and 31–50 are 70% and 75%, respectively.

The length (l), the width (w), the height (h), the number of each kind of goods per box (n), the upper bounds of box dimensions ($\overline{x}, \overline{y}, \overline{z}$), and the lower bound of VU rate (V_u) of each kind of goods used in our experiment are shown in Table 4.2.

Table 4.2**:** The generated data set.

Goods	l	w	h	n	$\overline{x}$	$\overline{y}$	$\overline{z}$	V_u
1	9.1	8.2	6.4	28	39	71	29	0.7
2	9.6	5.5	5.8	48	44	72	24	0.7
3	8.1	4.9	5.5	18	54	39	57	0.7
4	8	6.6	6	25	16	41	48	0.7
5	8.5	7.6	6.1	41	40	57	47	0.7
6	7.5	7.1	6.6	22	72	83	38	0.7
7	8.3	7	4.4	46	36	22	38	0.7
8	8	4.9	4.5	27	90	58	23	0.7
9	7.6	7.1	6.5	43	99	16	62	0.7
10	6.8	6.6	5.5	33	45	35	44	0.7
11	7.6	5.1	5.9	40	70	26	30	0.7

Table 4.2 – *Continued*

Goods	l	w	h	n	$\overline{x}$	$\overline{y}$	$\overline{z}$	V_u
12	9.6	9	7	38	80	82	64	0.7
13	8.5	8.2	5.5	34	79	41	94	0.7
14	9.3	6.2	4.8	20	57	51	14	0.7
15	8.4	5.9	5.5	14	59	61	15	0.7
16	9.2	7.6	7.6	16	36	63	65	0.7
17	17.8	14.8	18.7	30	82	73	80	0.7
18	18.4	9.7	10.9	12	92	52	59	0.7
19	18.9	14.3	7.9	26	79	85	36	0.7
20	20	9.3	12	26	70	71	31	0.7
21	11.3	11.3	6	13	57	28	48	0.7
22	11.9	11.8	12.5	38	79	84	69	0.7
23	11	10.4	15	14	98	40	53	0.7
24	12.5	12.5	15	25	31	83	80	0.7
25	19.4	12.4	8	7	38	68	36	0.7
26	16.4	12.7	8	10	72	89	54	0.7
27	16.9	13.9	8	31	53	66	37	0.7
28	19.9	8.9	7.5	24	91	65	53	0.7
29	20.4	9.4	15.5	15	35	82	64	0.7
30	20.4	9.3	7.5	34	71	49	25	0.7
31	26	8.5	12	24	41	86	89	0.75
32	23	13.9	10.5	22	46	85	48	0.75
33	26	19	14.5	12	86	57	92	0.75
34	26.8	17.4	12	31	94	46	51	0.75
35	21.9	20.8	11	16	54	51	60	0.75
36	25.1	17.5	10.5	22	59	93	83	0.75
37	28.5	19	13.2	8	31	41	53	0.75
38	22.8	8.1	10	19	72	97	24	0.75
39	23.5	11.9	11	14	69	44	62	0.75
40	21	10.6	8	17	56	87	84	0.75
41	38.9	20.4	21	13	99	78	88	0.75
42	33.4	16	17	14	70	55	62	0.75
43	30	12.1	18.2	15	52	70	39	0.75
44	31.5	16	22	9	50	74	86	0.75
45	37.4	14.9	25.5	11	82	80	59	0.75
46	35.1	21	13	10	67	46	89	0.75
47	42	10.9	25	7	74	95	54	0.75
48	44	19	22.5	12	80	74	100	0.75
49	43.5	22	20.5	8	91	86	44	0.75
50	40.6	18.9	20.5	9	42	63	70	0.75

4.2 The heuristic algorithm for designing boxes

In this section, we apply the proposed heuristic algorithm for designing box to the data set in Table 4.2. To design boxes, we set the number of iterations (T) of the heuristic algorithm for each kind of goods to 1000. The algorithm is implemented in Matlab version R2014b. All the experiments were run on a personal computer equipped with Intel Core i7 CPU, 8 GB RAM and Window 7 64-bit operating system. After performing the heuristic algorithm with the inputs from Table 4.2, we get outputs x^*, y^*, and z^* which are the magnitude of the required space for packing goods along the x-, y-, and z-axes, respectively as well as V^* which is the volume utilization rate of the designed box and the objective value f^* which is the difference between the largest and the smallest dimensions. The outputs for each kind of goods are reported in Table 4.3. The value $\max\{x^*, y^*\}, \min\{x^*, y^*\}$, and z^* which are referred to as the length (L), the width (W), and the height (H) of the designed box as well as the CPU time (in seconds) to get the solutions are also reported in Table 4.3. If $L = \max\{x^*, y^*\} = x^*$, it means that the length and the width of the designed box are along the x-axis and the y-axis, respectively. If $L = \max\{x^*, y^*\} = y^*$, it means that the length and width of the designed box are along the y-axis and the x-axis, respectively.

Table 4.3: The computational results from the box designing heuristic using the generated data set.

Goods	x^*	y^*	z^*	V^*	f^*	L	W	H	CPU time
1	26.4	26.4	25.6	0.7495	0.8	26.4	26.4	25.6	40.04s
2	28.8	28.8	23.2	0.7634	5.6	28.8	28.8	23.2	178.77s
3	17.9	17.9	16.5	0.7432	1.4	17.9	17.9	16.5	12.08s
4	16	26.4	24	0.7813	10.4	26.4	16	24	24.51s
5	25.5	30.4	24.4	0.8542	6	30.4	25.5	24.4	122.19
6	20.7	20.7	19.8	0.7898	0.9	20.7	20.7	19.8	22.49s
7	24.9	21	26.4	0.8519	5.4	24.9	21	26.4	129.15s
8	17.8	17.8	18	0.8351	0.2	17.8	17.8	18	38.20s
9	35.5	15.2	32.5	0.8600	20.3	35.5	15.2	32.5	114.12s
10	20.4	20.4	20	0.8897	0.4	20.4	20.4	20	66.79s
11	22.9	22.9	23.6	0.7391	0.7	22.9	22.9	23.6	111.6s
12	28.8	36	28	0.7917	8	36	28.8	28	102.19s
13	25.5	25.5	27.5	0.7289	2	25.5	25.5	27.5	71.65s

Table 4.3 – *Continued*

Goods	x^*	y^*	z^*	V^*	f^*	L	W	H	CPU time
14	27.8	24.8	9.6	0.8333	18.3	27.8	24.8	9.6	13.96s
15	20.2	20.2	11	0.8502	9.2	20.2	20.2	11	5.09s
16	22.8	22.8	22.8	0.7173	0	22.8	22.8	22.8	8.75s
17	59.2	59.2	56.1	0.7517	3.1	59.2	59.2	56.1	52.27s
18	36.8	29.1	21.8	1	15	36.8	29.1	21.8	3.62s
19	47.5	47.5	31.6	0.7786	15.9	47.5	47.5	31.6	31.09s
20	55.8	57.2	24	0.7576	33.2	57.2	55.8	24	26.05s
21	22.6	22.6	24	0.8125	1.4	22.6	22.6	24	4.52s
22	47.4	47.4	37.5	0.7917	9.9	47.4	47.4	37.5	98.98s
23	31.8	31.8	30	0.7919	1.8	31.8	31.8	30	6.06s
24	25	50	60	0.7813	35	50	25	60	26.49s
25	24.8	31.8	24	0.7117	7.8	31.8	24.8	24	1.03s
26	29.1	25.4	32	0.7045	6.6	29.1	25.4	32	2.60s
27	47.7	47.7	32	0.8001	15.7	47.7	47.7	32	47.35s
28	39.8	37.7	30	0.7082	9.8	39.8	37.7	30	27.80s
29	29.8	40.8	46.5	0.7886	16.7	40.8	29.8	46.5	6.16s
30	50.1	48.3	22.5	0.8886	27.6	50.1	48.3	22.5	51.92s
31	34.5	43	48	0.8938	13.5	43	34.5	48	21.45s
32	46	46	42	0.8310	4	46	46	42	16.75s
33	45	52	43.5	0.8444	8.5	52	45	43.5	3.74s
34	87	44.2	48	0.9398	42.8	87	44.2	48	26.80s
35	43.8	43.8	44	0.9498	0.2	43.8	43.8	44	6.30s
36	42.6	52.5	52.5	0.8642	9.9	52.5	42.6	52.5	18.26s
37	28.5	38	52.8	1	24.3	38	28.5	52.8	0.86s
38	45.6	40.5	20	0.9500	25.6	45.6	40.5	20	11.77s
39	35.7	35.7	44	0.7680	8.3	35.7	35.7	44	5.30s
40	42	31.8	24	0.9444	18	42	31.8	24	10.42s
41	79.7	77.8	42	0.8319	37.7	79.7	77.8	42	4.37s
42	64	49.4	51	0.7888	14.6	64	49.4	51	4.48s
43	48.4	60.5	36.4	0.9298	24.1	60.5	48.4	36.4	3.96s
44	48	63	44	0.75	19	63	48	44	1.68s
45	59.6	67.2	51	0.7653	16.2	67.2	59.6	51	2.55s
46	56.1	42	52	0.7821	14.1	56.1	42	52	1.73s
47	42	43.6	50	0.875	8	43.6	42	50	0.96s
48	63	63	67.5	0.8425	4.5	63	63	67.5	2.88s
49	66	65.5	41	0.8855	25	66	65.5	41	1.14s
50	40.6	56.7	61.5	1	20.9	56.7	40.6	61.5	1.52s

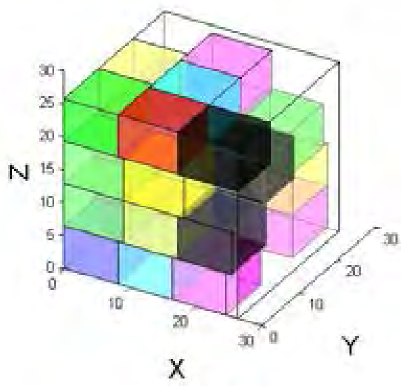
Examples of output P^* (an arrangement pattern matrix that gives the information about the position of the left-front-bottom corner and the rotation of each item of the goods of one kind) are given in Equation (4.1) and (4.2) for the 47^{th} and the 48^{th} kind of goods, respectively.

$$P^* = \begin{array}{c|cccccc|} & p_i & q_i & r_i & x_i & y_i & z_i \\ \text{item } i{=}1 & 42 & 10.9 & 25 & 0 & 0 & 0 \\ \text{item } i{=}2 & 42 & 10.9 & 25 & 0 & 10.9 & 0 \\ \text{item } i{=}3 & 42 & 10.9 & 25 & 0 & 21.8 & 0 \\ \text{item } i{=}4 & 42 & 10.9 & 25 & 0 & 0 & 25 \\ \text{item } i{=}5 & 42 & 10.9 & 25 & 0 & 10.9 & 25 \\ \text{item } i{=}6 & 42 & 10.9 & 25 & 0 & 21.8 & 25 \\ \text{item } i{=}7 & 42 & 10.9 & 25 & 0 & 32.7 & 0 \end{array} \tag{4.1}$$

$$P^* = \begin{array}{c|cccccc|} & p_i & q_i & r_i & x_i & y_i & z_i \\ \text{item } i{=}1 & 19 & 44 & 22.5 & 0 & 0 & 0 \\ \text{item } i{=}2 & 19 & 44 & 22.5 & 19 & 0 & 0 \\ \text{item } i{=}3 & 19 & 44 & 22.5 & 0 & 0 & 22.5 \\ \text{item } i{=}4 & 44 & 19 & 22.5 & 19 & 0 & 22.5 \\ \text{item } i{=}5 & 19 & 44 & 22.5 & 19 & 19 & 22.5 \\ \text{item } i{=}6 & 44 & 19 & 22.5 & 0 & 44 & 0 \\ \text{item } i{=}7 & 19 & 44 & 22.5 & 38 & 0 & 0 \\ \text{item } i{=}8 & 19 & 44 & 22.5 & 38 & 19 & 22.5 \\ \text{item } i{=}9 & 19 & 44 & 22.5 & 0 & 0 & 45 \\ \text{item } i{=}10 & 19 & 44 & 22.5 & 19 & 0 & 45 \\ \text{item } i{=}11 & 19 & 44 & 22.5 & 38 & 0 & 45 \\ \text{item } i{=}12 & 44 & 19 & 22.5 & 0 & 44 & 45 \end{array} \tag{4.2}$$

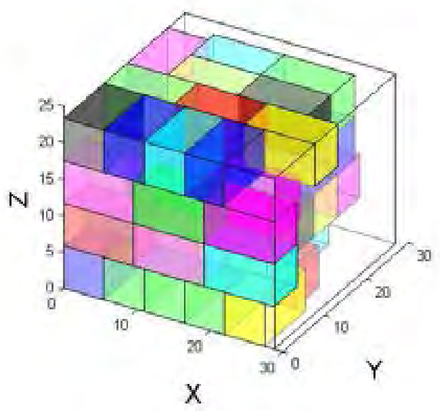
The results in Table 4.3 show that the proposed heuristic algorithm for designing a box that is close to a cube for packing each kind of goods within the given bounds gives solutions within acceptable times. The values in column of f^* in Table 4.4 are the best found objective values of our problem, $f = \max\{x, y, z\} - \min\{x, y, z\}$. Note that the objective value f^* of a kind of goods may be large (i.e., the magnitudes of the longest side and the shortest side of the designed box for packing all items of goods of that kind are rather different) due to the limitation of the upper bounds. For example, if we consider the goods of the 9^{th} kind, the length of the box for packing this kind of goods is $L = 35.5$ which is the magnitude of the longest side of this box and the width $W = 15.2$ which is the shortest side. The value L is significantly different from W since the upper bound of box dimension along the x-axis of this box is $\overline{x} = 99$ which is high and the upper bound of box dimension along the y-axis is only $\overline{y} = 16$ which is low. Therefore, the width of this box is inevitably different from the length significantly. In addition, there are only 3 designed boxes (for the $18^{th}, 37^{th}$, and 50^{th} goods) where the VU rate is equal to 1. The solutions with the VU rate equal to 1 are attractive but rarely occur since it is not a part of the objective function and it is not forced by a constraint. For applications where the VU rate should be near 1 as much as possible, the heuristic algorithm can be adjusted accordingly through the objective function and/or a constraint.

Figures 4.1 (a) – (h) show some graphical representations of the solutions. They show the designed box and the goods arrangement within the box for goods of kind k where $k = 1, 2, 17, 18, 31, 32, 47, 48$, respectively [1].

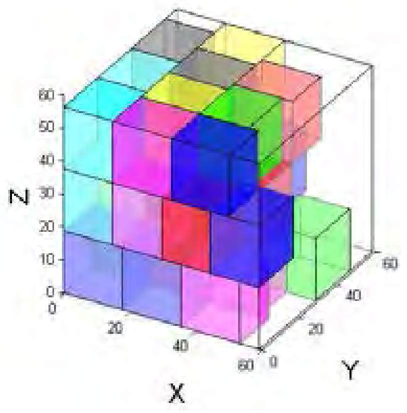


(a) The graphical representation of the designed box for packing 1^{st} kind of goods and their arrangement pattern.

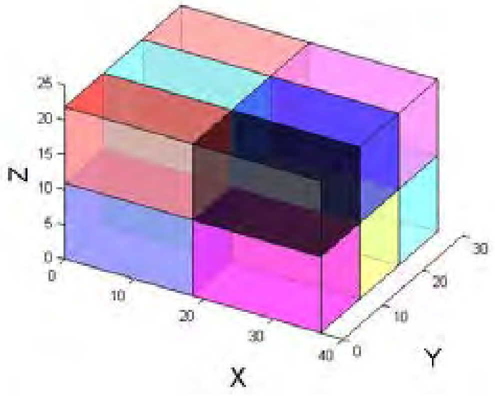
[1]The goods are of the same kind, but we color them differently for easy viewing of the arrangement patterns.



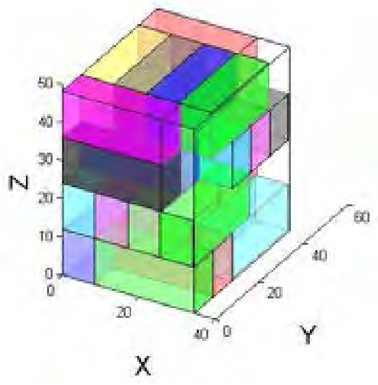
(b) The graphical representation of the designed box for packing 2nd kind of goods and their arrangement pattern.



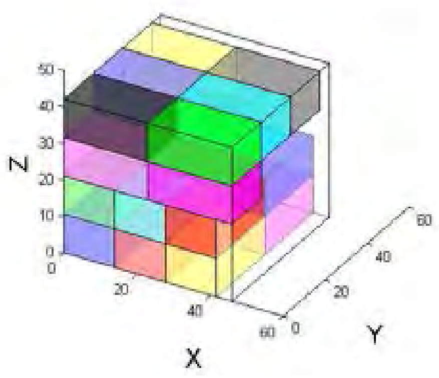
(c) The graphical representation of the designed box for packing 17th kind of goods and their arrangement pattern.



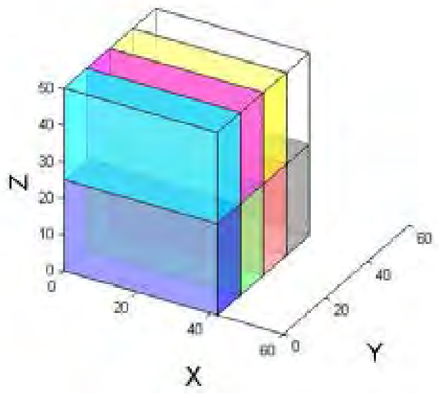
(d) The graphical representation of the designed box for packing 18th kind of goods and their arrangement pattern.



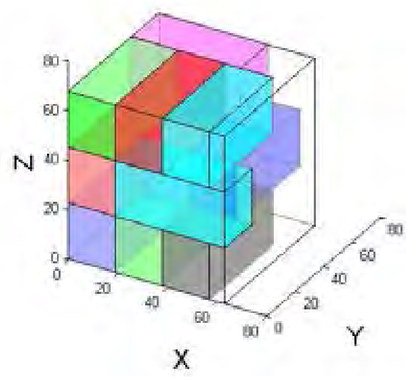
(e) The graphical representation of the designed box for packing 31st kind of goods and their arrangement pattern.



(f) The graphical representation of the designed box for packing 32nd kind of goods and their arrangement pattern.



(g) The graphical representation of the designed box for packing 47th kind of goods and their arrangement pattern.



(h) The graphical representation of the designed box for packing 48th kind of goods and their arrangement pattern.

Figure 4.1: The graphical representations of some solutions.

4.3 The heuristic algorithm for minimizing number of types of packing boxes

In this section, we will minimize the number of packing box types (50 types) that are designed in Section 4.2 by using the proposed heuristic algorithm for minimizing the number of types of packing boxes in Section 3.3. The parameters of the length (L), the width (W), and the height (H) of all boxes are in columns of L, W, and H in Table 4.3. This algorithm also has the input t, the bound of the difference percentage for each side between of a discarded box and its substituting box. We vary this parameter t as 5%, 10%, 15%, and 20% and implement the algorithm in Matlab version R2014b. All the experiments were run on a personal computer equipped with Intel Core i7 CPU, 8 GB RAM and Window 7 64-bit operating system. The number of iterations Num that we use, the CPU time (in seconds), and the value of f^* which imply the best number of remaining box types after substitution in each case of the parameter t are reported in Table 4.4.

The results in Table 4.4 show that if we accept the value of t at 5%, 10%, 15%, and

Table 4.4: The computational results of the heuristic algorithm for minimizing the number of packing box types.

t	Num	f^*	CPU time
5%	10	48	0.0308s
10%	10	48	0.0351s
15%	50	40	0.0445s
20%	50	30	0.0536s

20%, we can reduce at most 2, 2, 10, and 20 box types, respectively. Consequently, the number of remaining box types after substitution in each case is 48, 48, 40, and 30 types, respectively. The time for solving the problem in each case is less than 0.1 second.

Table 4.5 displays the output matrix T^* for each case which gives the information about which box is discarded and which box substitutes it. The elements of T^* are either 0 or 1, so we will mention only the elements T^*_{ij} which is equal to 1 in Table 4.5.

Table 4.5: List of $T^*_{ij} = 1$ for each case of t.

t	List of $T^*_{ij} = 1$
5%	$T_{11,16}, T_{27,19}$
10%	$T_{11,16}, T_{27,19}$
15%	$T_{1,21}, T_{10,8}, T_{12,25}, T_{13,11}, T_{16,6}, T_{20,30}, T_{27,19}, T_{31,29}, T_{40,18}, T_{42,44}$
20%	$T_{6,3}, T_{6,8}, T_{12,2}, T_{12,5}, T_{12,25}, T_{13,7}, T_{13,11}, T_{13,16}, T_{20,30}, T_{21,10}$, $T_{23,1}, T_{27,19}, T_{31,29}, T_{35,39}, T_{36,47}, T_{40,18}, T_{41,49}, T_{44,33}, T_{45,42}, T_{48,17}$

From the Table 4.5, if we accept the value of t at 5% or 10%, we can discard at most 2 boxes. Specifically, we can discard the 16^{th} and 19^{th} box and use the 11^{th} and 27^{th} box instead, respectively. However, if we accept the the value of t at 15%, we can discard more boxes. Specifically, we can discard at most 10 boxes, including the $21^{st}, 8^{th}, 25^{th}, 11^{th}, 6^{th}, 30^{th}, 19^{th}, 29^{th}, 18^{th}$, and 44^{th}. The boxes for substituting them are the $1^{st}, 10^{th}, 12^{th}, 13^{th}, 16^{th}, 20^{th}, 27^{th}, 31^{st}, 40^{th}$, and 42^{th}, respectively. The results for $t = 20\%$ can also be explained in the similar manner. The results from this section will be compared with the results from the BILP for minimizing the number of box types in the next section.

4.4 The binary integer linear programming for minimizing number of types of packing boxes

In this section, the 50 box types that we design in Section 4.2 are minimized by using the BILP model in Section 3.4. The parameter t is varied in the same manner as in Section 4.3. The model are solved using CPLEX version 12.6.3 running on the same computer. The results of the problems are reported in Table 4.6 and will be compared with the results from the heuristic algorithm in Section 4.3.

Table 4.6: The computational results of the BILP model for minimizing the number of packing box types.

t	Optimal objective value	List of $T_{ij} = 1$	CPU time
5%	48	$T_{11,16}, T_{27,19}$	2.40s
10%	48	$T_{11,16}, T_{27,19}$	2.53s
15%	40	$T_{1,11}, T_{1,21}, T_{12,25}, T_{16,6}, T_{16,10}, T_{20,30}, T_{27,19},$ $T_{31,29}, T_{40,18}, T_{42,44}$	2.29s
20%	30	$T_{6,3}, T_{6,8}, T_{11,10}, T_{11,16}, T_{12,2}, T_{12,5}, T_{12,25},$ $T_{23,1}, T_{23,13}, T_{26,7}, T_{27,19}, T_{30,38}, T_{31,29}, T_{35,39}$ $T_{36,47}, T_{40,18}, T_{41,49}, T_{42,33}, T_{45,44}, T_{48,17}$	2.51s

The results from Table 4.6 show that the objective value in each case that we get from the BILP model is equal to the objective value from the proposed heuristic algorithm for minimizing the number of box types in Section 4.3. It means that the proposed heuristic algorithm is useful since it can find an optimal solution in all cases of consideration. The list of T_{ij} in the cases $t = 15\%$ and $t = 20\%$ in Table 4.6 are different from the results in Table 4.5, which indicates that the problems have alternative optimal solutions. The CPU time for solving the BILP in each case is around 2.5 seconds which is greater than the CPU time required by the heuristic algorithm in Section 4.3.

Next, we consider the case of $t = 30\%$. The results after solving the problem by using the proposed heuristic algorithm in Section 3.3 with the number of iterations Num set to 10^5 and the proposed BILP in Section 3.4 are shown in Table 4.7.

Table 4.7**:** The results of case $t = 30\%$ from the two methods.

Method	The objective value	CPU time
The proposed heuristic	22	20.69s
The BILP model	20	2.55s

From Table 4.7, we can see that the BILP model gives the solution which is the optimal solution with 20 box types within 2.55 seconds but the heuristic algorithm gives the solution 22 which is not optimal solution. Moreover, the heuristic algorithm uses more time for solving with 20.69 seconds. This happens because if we adjust the parameter t in the BILP model, the size of the model does not change. It is only an adjustment of one parameter (coefficient) of the model in CPLEX solver and it should use comparable time for solving the problem. However, if we considerably increase the parameter t in the heuristic algorithm, the size of problem will increase significantly because it will increase the number of possible cases of -1 in the matrix T to be considered.

The generated data set in Section 4.1 and all results from Sections 4.2 – 4.4 altogether are examples of how to design packing boxes to minimize the number of packing box types. Any industries can use the proposed heuristic for designing boxes and the proposed BILP model or the proposed heuristic for minimizing number of box types for their goods so that they can design a box that is close to a cube for packing their goods and can reduce the packing cost by minimizing the number of box types. In the next section, we will show more examples to compare the efficiency and benefit of the proposed heuristic algorithm for minimizing the number of packing boxes and the BILP model for minimizing the number of packing boxes.

4.5 The comparison of efficiency of the heuristic algorithm and binary integer linear programming for minimizing number of types of packing boxes

In this section, we will compare the efficiency of the heuristic algorithm and the BILP for minimizing the number of packing boxes with more large size examples. We will use the sizes of 50 boxes that we design from Section 4.2 and we also randomly generate 70 boxes which are boxes No.51–120 to use in this section. (Therefore, the total number of box types used in this section is 120.) The sizes of boxes No.51–120 are shown in Table 4.8.

We consider problems which are to minimize the number of boxes where the number of boxes (n) are as follows:

- 60 boxes (box No.1–60)
- 70 boxes (box No.1–70)
- 80 boxes (box No.1–80)
- 90 boxes (box No.1–90)
- 100 boxes (box No.1–100)
- 110 boxes (box No.1–110)
- 120 boxes (box No.1–120)

For simplicity, we will call all problems ($n = 60$ to 120) as the problem set. We consider the problem set with various bounds of the difference percentage (t), i.e., 5%, 10%, 15%, 20% and 30%. We solve the problem set with each case t by both the heuristic algorithm and the BILP model for minimizing the number of packing box types. We

Table 4.8**:** The sizes of boxes No.51 – 120.

Box	l	w	h
51	10	10	10
52	15	15	40
53	17	11	6
54	20	20	60.5
55	20.5	14.5	6
56	25	17	9
57	25	25	25
58	27.3	22.5	30
59	30	20	11
60	35	22	14
61	35.8	30.5	80
62	36	31	26
63	40	40	50
64	40	24	16
65	45	40	35
66	45	30	20
67	49	43	31
68	50	45	30
69	51	43	35
70	53.6	21.7	42
71	55	45	40
72	60	60	20
73	60	20	45
74	63.3	37.5	40
75	66	60	100
76	69	50	44
77	70	40	10
78	74.5	49.3	50
79	78	40	55
80	80	80	40
81	80	40	20
82	84	33	22
83	87.6	74	60.5
84	93	34	20
85	95	50	26

Box	l	w	h
86	100	30	30
87	105	32	37
88	110	70	25
89	112.5	40.5	10
90	120	60	35
91	100.5	33.8	19.5
92	71.5	49.5	14.5
93	50.3	50.3	30
94	70.5	48.5	24.5
95	63.5	44	21.5
96	66	42	19.5
97	71	46	24
98	57	46	11
99	69	52	9.3
100	67	50.3	28
101	64	40	28.5
102	70.5	49.5	16
103	28.5	16.5	15.25
104	51	17.75	13.35
105	26	19	23.5
106	31	23	27
107	35.5	24.75	21.25
108	39.4	29.5	17.5
109	36	31	13
110	36	31	26
111	36.5	32.4	62.25
112	45	40	35
113	55	45	40
114	80	80	20
115	31.75	24.1	16.8
116	57	26.5	33
117	40	30.5	24.1
118	45	40	35
119	55	45	40
120	60	50	45

implement the heuristic algorithm in Matlab and the BILP model in CPLEX on the same computer configurations described in the previous section.

Table 4.9**:** Efficiency comparison in case $t = 5\%$ of the two methods.

n	Objective value (optimal)	CPLEX CPU times	Average Matlab CPU times to get optimal
60	58	2.02 s	0.02775 s
70	68	2.15 s	0.02867 s
80	78	2.69 s	0.02999 s
90	88	3.12 s	0.03236 s
100	98	4.02 s	0.02748 s
110	107	5.18 s	0.02954 s
120	113	6.13 s	0.02692 s

Table 4.9 shows the results for the problem set with $t = 5\%$. First, we obtain the optimal objective value of each problem and the CPLEX CPU time as shown in the second and the third columns, respectively. For the heuristic algorithm, we solve each problem in the problem set in Matlab until the objective value of the yielded solution is equal to the optimal solution obtained by the BILP model. We solve each problem with the heuristic algorithm 10 times and find the average computational time to get the optimal solution, which is shown in the last column of Table 4.10. From these results, we can see that the average computational time of the heuristic algorithm of each problem is less than the CPLEX CPU time to get the optimal solution.

Table 4.10**:** Efficiency comparison in case $t = 10\%$ of the two methods.

n	Objective value (optimal)	CPLEX CPU times	Average Matlab CPU times to get optimal
60	56	2.09 s	0.03142 s
70	65	2.31 s	0.02896 s
80	74	2.87 s	0.02670 s
90	84	3.34 s	0.02866 s
100	94	4.07 s	0.02621 s
110	103	5.05 s	0.03252 s
120	108	6.03 s	0.03069 s

Table 4.11**:** Efficiency comparison in case $t = 15\%$ of the two methods.

n	Objective value (optimal)	CPLEX CPU times	Average Matlab CPU times to get optimal
60	48	2.00 s	0.28777 s
70	54	2.23 s	0.30612 s
80	64	2.94 s	0.32949 s
90	74	3.37 s	0.30901 s
100	82	4.25 s	0.32448 s
110	90	5.00 s	0.32170 s
120	94	6.13 s	0.41607 s

Tables 4.10 and 4.11 show the results of the problem set with $t = 10\%$ and $t = 15\%$, respectively. The second and third columns of the tables show the optimal solution of each problem and the CPLEX CPU time for solving each problem. The last column of Tables 4.10 and 4.11 show the average Matlab computational time (from 10 repetitions) to get the optimal solution of each problem. We can see that, in both cases $t = 10\%$ and $t = 15\%$, the average computational times to get the optimal solution of the heuristic algorithm of all problems are less than the CPLEX CPU times as in case $t = 5\%$.

Table 4.12**:** Efficiency comparison in case $t = 20\%$ of the two methods.

n	Objective value (optimal)	Average objective value of heuristic	CPLEX CPU times	Matlab CPU times
60	36	36	1.98 s	$\approx$ 1.98 s
70	40	41	2.53 s	$\approx$ 2.53 s
80	49	51	3.06 s	$\approx$ 3.06 s
90	58	60	3.55 s	$\approx$ 3.55 s
100	66	67	4.41 s	$\approx$ 4.41 s
110	73	74	5.37 s	$\approx$ 5.37 s
120	77	78	6.52 s	$\approx$ 6.52 s

Table 4.12 shows the results for the problem set with $t = 20\%$. We obtain the optimal objective value of each problem and the CPLEX CPU time as shown in the second and forth columns, respectively. For the heuristic algorithm, we solve each problem in the problem set by setting the computational time in Matlab is similar to the CPLEX CPU time to get the optimal solution. We run heuristic algorithm 10 times and find the

average objective value. The average objective value of each problem is shown in the third column of Table 4.10. The results show that the heuristic algorithm can find the objective value which is equal to the optimal solution within the CPLEX CPU time in the problem with 60 boxes only. But in other problems (70–120 boxes), the average objective value from the heuristic algorithm is slightly higher than the optimal solution.

Table 4.13**:** Efficiency comparison in case $t = 30\%$ of the two methods.

n	Objective value (optimal)	Average objective value of heuristic	CPLEX CPU times	Matlab CPU times
60	25	29	2.32 s	$\approx$ 2.32 s
70	27	32.5	2.64 s	$\approx$ 2.64 s
80	35	40.5	2.99 s	$\approx$ 2.99 s
90	43	48.3	3.58 s	$\approx$ 3.58 s
100	49	53.8	4.24 s	$\approx$ 4.24 s
110	53	58.9	5.36 s	$\approx$ 5.36 s
120	55	62.2	6.08 s	$\approx$ 6.08 s

Table 4.13 shows the results for the problem set with $t = 30\%$. The optimal solution and the CPU time that used for solving each problem in CPLEX are shown in the second and forth columns, respectively. For the heuristic algorithm, we perform similarly to the case $t = 20\%$, i.e., we solve each problem in the problem set by setting the computational time in Matlab similar to the CPLEX CPU time to get the optimal solution. Each problem is run 10 times and the average objective value of these 10 runs of each problem is shown in the third column of Table 4.13. The results show that the heuristic algorithm cannot find the optimal solution within CPLEX CPU time in all problems. Furthermore, the average objective value from the heuristic algorithm of each problem is rather different from the optimal solution.

The results from all Tables 4.9–4.13 imply that the heuristic algorithm for minimizing the number of box types can give goods result if the the bound of the difference percentage for each side between of a discarded box and its substituting box (t) is low. But if the parameter t is high, the heuristic algorithm cannot find an optimal solution of the problem within the time to get optimal solution from CPLEX. However, we usually set the parameter t low when we try to substitute a box with a bigger box in practice so that the substituting box is not much larger than the discarded box. Therefore, the heuristic algorithm for minimizing the number of packing boxes can be a good option to reduce the number of box types.

CHAPTER V

CONCLUSIONS

5.1 Conclusion of this work

This research study proposes a heuristic for designing packing boxes to pack rectangular goods/products of the same kind. The shape of the designed box will be close to a cube as much as possible within some boundaries and the volume utilization rate of the designed box will be at least a value given by the user. It can be an option for any industry who tries to design a cube packing box for packing their goods. An advantage of a box with a shape near a cube is that it is more stable than other rectangular box in every rotation. Moreover, our heuristic also considers the handling of empty space in the box by forcing the volume utilization rate to be at least a lower bound which may be derived from some limitations in the transportation. This helps increase efficiency when the goods are to be transported. In addition, this research study proposes a heuristic and a mathematical model for minimizing the number of types of packing boxes. They can be options to reduce the number of box types when there are several box types to be used in product packing. This can be valuable for any packing industries who seek to manage the number of box types in their business. The heuristic and model can reduce some box types which leads to a lower packing cost. Example problems are provided to illustrate the proposed heuristic and the model. The obtained results show satisfactory performances of both approaches. When the parameter t is no more than 15%, the heuristic obtains equivalent solution to that of the exact method with less computational time.

5.2 Discussion and future works

1. The heuristic algorithm for designing box in this work does not consider the limitation of stackablility of goods in the box. There may be a stacked item that is

supported by other packed items but the area where they actually touch is small which can cause instability. However, filling the gap with some cushioning materials such as foam peanuts can help elevate this problem.

2. In this study, the heuristic for designing a packing box is designed with near-cube shape in mind. Further research is needed to extend or modify the heuristic to accommodate other general rectangular shapes of a box.

3. The container loading problem (CLP) that is applied in this research does not consider the weight distribution of boxes in the container and the center of gravity. Further research should take these requirements into consideration for more realistic applications and safety concerns.

4. The efficiency of the heuristic for minimizing the number of box types drastically declined as the parameter t becomes too large due to the combinatorial nature of the feasible solutions. More study is needed to improve the efficiency of the heuristic. However, although the heuristic does not guarantee an optimal solution, unlike the mathematical model approach which needs a solver to obtain an optimal solution, the heuristic can be implemented using any programming language.

5. The problem of minimizing the number of types of packing boxes requires the parameter t, which is the upper bound of the percentage difference between the size of the discarded box and the substituting box, from the user. For some users who cannot decide on the value of t but wish to eliminate a fixed number of box types, the problem can be changed to minimizing the percentage difference given the number of box types they want to discard. The solution will give the least upper bound on the percentage difference given the fixed number of box types are discarded. Then, the user can decide whether the resulting upper bound is acceptable and pursue the suggested substitution plan. If the solution is not acceptable, the user can adjust the parameter (number of discarded box types) and resolve the problem. With minor modification to the mathematical model in Section 3.4, we can obtain a mathematical model for the new problem as follows.

Parameters:

n	The total number of the given boxes.
N	The index set of boxes, $N = \{1, 2, ..., n\}$.
M	An arbitraly large positive number.
ε	An infinitesimally small positive number.
L_i, W_i, H_i	Parameter indicating the length, the width, and the height of box i, respectively.
m	The total number of boxes to be discarded.

Variables:

x	Decision variable implying the bound of the difference percentage for each side between of a discarded box and is subsituting box.
T_{ij}	Decision binary variable which is equal to 1 if box i substitutes box j.
$y_{ij}, a_{ij}, b_{ij}, c_{ij}, d_{ij}$	Auxilary binary variables.

Mathematical Model:

$$
\begin{aligned}
\min \quad & x \\
\text{s.t.} \quad & x \leq 1 && && (1) \\
& [|L_i - L_j| + |W_i - W_j| + |H_i - H_j|] - [(L_i - L_j) + (W_i - W_j) + (H_i - H_j)] \\
& \qquad \leq M(1 - y_{ij}) && \forall i, j \in N, i \neq j, && (2) \\
& T_{ij} \leq y_{ij} && \forall i, j \in N, i \neq j, && (3) \\
& (L_i - L_j) - x(L_i) \leq M(1 - a_{ij}) && \forall i, j \in N, i \neq j, && (4) \\
& T_{ij} \leq a_{ij} && \forall i, j \in N, i \neq j, && (5) \\
& (W_i - W_j) - x(W_i) \leq M(1 - b_{ij}) && \forall i, j \in N, i \neq j, && (6) \\
& T_{ij} \leq b_{ij} && \forall i, j \in N, i \neq j, && (7) \\
& (H_i - H_j) - x(H_i) \leq M(1 - c_{ij}) && \forall i, j \in N, i \neq j, && (8) \\
& T_{ij} \leq c_{ij} && \forall i, j \in N, i \neq j, && (9) \\
& \sum_{\substack{\forall i \in N \\ i \neq j}} T_{ij} \leq 1 && \forall j \in N, && (10) \\
& T_{ij} - 1 \leq M(1 - d_{ij}) - \epsilon && \forall i, j \in N, i \neq j, && (11) \\
& T_{jk} \leq d_{ij} && \forall i, j, k \in N, i \neq j \neq k, && (12) \\
& T_{ki} \leq d_{ij} && \forall i, j, k \in N, i \neq j \neq k, && (13) \\
& \sum_{\forall i \in N} \sum_{\substack{\forall j \in N \\ i \neq j}} T_{ij} = m && && (14)
\end{aligned}
$$

$$x \geq 0; \; T_{ij}, y_{ij}, a_{ij}, b_{ij}, c_{ij}, d_{ij} \in \{0, 1\}$$

This mathematical model is presented in detail in Laisupannawong et al. [16].

REFERENCES

[1] C.S. Chen, S.M. Lee, and Q.S. Shen, “An analytical model for the container loading problem”, *European Journal of Operational Research*, vol. 80, pp. 68–76, 1995.

[2] J.F. Tsai and H.L. Li, “A global optimization method for packing problems”, *Engineering Optimization*, vol. 38, pp. 687–700, 2006.

[3] J.F. Tsai, P.C. Wang, and M.H. Lin, “A global optimization approach for solving three-dimensional open dimension rectangular packing problems”, *Optimization*, vol. 64, pp. 2601–2618, 2015.

[4] N.Z. Hu, H.L Lee, and J.F. Tsai, “Solving packing problems by a distributed global optimization algorithm”, *Mathematical Problems in Engineering*, doi: 10.1155/2012/931092, 2012.

[5] D. Pisinger, “Heuristics for the container loading problem”, *European Journal of Operational Research*, vol. 141, pp. 382–392, 2002.

[6] A. Bortfeldt and D. Mack, “A heuristic for the three-dimensional strip packing problem”, *European Journal of Operational Research*, vol. 183, pp. 1267–1279, 2007.

[7] M. Elay, “Solving container loading problems by block arrangement”, *European Journal of Operational Research*, vol. 141, pp. 393–409, 2002.

[8] K. Karabulut and M.M. Inceoglu, “A hybrid genetic algorithm for packing in 3D with deepest bottom left with fill method”, *Advances in Information Systems*, pp. 441–450, 2004.

[9] K. Kang, I. Moon, and H. Wang, “A hybird genetic algorithm with a new packing strategy for the three-dimensional bin packing problem”, *Applied Mathematics and Computation*, vol. 219, pp. 1287–1299, 2012.

[10] J.F. Goncalves and M.G. Resende, "A parallel multi-population biased random-key genetic algorithm for a container loading problem", *Computers and Operations Research*, vol. 39, pp. 179–190, 2012.

[11] Y.H. Huang, F.J. Hwang, and H.C. Lu, "An effective placement method for the single container loading problem", *Computers & Industrial Engineering*, vol. 97, pp. 212–221, 2016.

[12] C. Paquay, M. Schyns, and S. Limbourg, "Three dimensional bin packing problem applied to air cargo", *Colloque International SIL 2011*, Casablanca, Morocco, 15–16 December 2011.

[13] Ministry of Industry Announcement No. 4432, 2012, *Corrugated fibreboard boxes*, accessed 16 August 2017, http://www.ratchakitcha.soc.go.th/DATA/PDF/2555/E/129/12.PDF.

[14] D.S. Chen, R.G. Batson, and Y. Dang, *Applied integer programming: Modeling and solution*, John Wiley & Sons, 2010.

[15] F.S. Hillier and G.J. Lieberman, *Introduction to mathematical programming*, 2^{nd} ed., McGraw-Hill, 1995.

[16] T. Laisupannawong, B. Intiyot and P. Thipwiwatpotjana, "Mixed integer linear programming model for reducing types of packing boxes", *Proceedings Operations Research Network Conference 2018*, Pattaya, Thailand, 23–24 April 2018, pp. 96–100.

APPENDICES

APPENDIX A : Matlab code for the heuristic algorithm for designing boxes.

```
tic
T = 1000; n = 28; Vu = 0.5;           %input
xbar = 39; ybar = 71; zbar = 29;
w = 8.2; l = 9.1; h = 6.4;

P = [l*ones(n,1) w*ones(n,1) h*ones(n,1) -ones(n,3)];
InfinitY = 10000;
P_star = 0; f_star = InfinitY;
V_star = 0;

for r = 1:T
   if P(1,1)>xbar || P(1,2)>ybar || P(1,3)>zbar
      display('infeasible'),r
   else
      P(1,4) = 0; P(1,5) = 0; P(1,6) = 0; %load box 1
      Nl = 1;

      i = 2;
      xfi = P(1,4)+P(1,1);            %find index i1 for box 2
      yfi = P(1,5);
      zfi = P(1,6);
      if xfi+P(i,1) <= xbar && yfi+P(i,2) <= ybar && zfi+P(i,3) <=
          zbar
         i1 = 1;
      else
         i1 = -1;
      end
      xfi = P(1,4);                   %find index i2 for box 2
      yfi = P(1,5)+P(1,2);
      zfi = P(1,6);
      if xfi+P(i,1) <= xbar && yfi+P(i,2) <= ybar && zfi+P(i,3) <=
          zbar
```

```
            i2 = 1;
        else
            i2 = -1;
        end
        xfi = P(1,4);                     %find index i3 for box 2
        yfi = P(1,5);
        zfi = P(1,6)+P(1,3);
        if xfi+P(i,1) <= xbar && yfi+P(i,2) <= ybar && zfi+P(i,3) <=
            zbar
            i3 = 1;
        else
            i3 = -1;
        end
        if i1 == -1 && i2 == -1 && i3 == -1
            display('infeasible'),r
        else
            P = Loadbox(P,i,i1,i2,i3);
            Nl = [Nl i];

            for i = 3:n
                i1 = Find_index_x(Nl,P,i,xbar,ybar,zbar);
                i2 = Find_index_y(Nl,P,i,xbar,ybar,zbar);
                i3 = Find_index_z(Nl,P,i,xbar,ybar,zbar);
                if i1 == -1 && i2 == -1 && i3 == -1
                    display('infeasible'),r
                    x_z = InfinitY
                    break
                else
                    P = Loadbox(P,i,i1,i2,i3);
                    Nl = [Nl i];
                end
            end
        end
    end
    if length(Nl) == n
        P;
        XP = P(1:n,4)+P(1:n,1);
        x = max(XP);
        YP = P(1:n,5)+P(1:n,2);
        y = max(YP);
        ZP = P(1:n,6)+P(1:n,3);
        z = max(ZP);
        xyz = sort([x y z]);
        f = xyz(3)-xyz(1);
        V = (n*l*w*h)/prod(xyz);
        if f < f_star && V >= Vu
            f_star = f;
            P_star = P;
            x_star = x;
            y_star = y;
            z_star = z;
            V_star = V;
        end
    end

    for i = 1:n
        a = randi(2);
        if a == 1
            mark = P(i,1);
            P(i,1) = P(i,2);
            P(i,2) = mark;
        end
    end
end
f_star
P_star
x_star
y_star
```

```
z_star
V_star

%plot result
clf;
figure(1);
hold on;
for i = 1:size(P_star,1)
    if mod(i,7) == 1
        cube_plot([P_star(i,4),P_star(i,5),P_star(i,6)],P_star(i,1),
            P_star(i,2),P_star(i,3),'b');
    elseif mod(i,7) == 2
        cube_plot([P_star(i,4),P_star(i,5),P_star(i,6)],P_star(i,1),
            P_star(i,2),P_star(i,3),'g');
    elseif mod(i,7) == 3
        cube_plot([P_star(i,4),P_star(i,5),P_star(i,6)],P_star(i,1),
            P_star(i,2),P_star(i,3),'r');
    elseif mod(i,7) == 4
        cube_plot([P_star(i,4),P_star(i,5),P_star(i,6)],P_star(i,1),
            P_star(i,2),P_star(i,3),'c');
    elseif mod(i,7) == 5
        cube_plot([P_star(i,4),P_star(i,5),P_star(i,6)],P_star(i,1),
            P_star(i,2),P_star(i,3),'m');
    elseif mod(i,7) == 6
        cube_plot([P_star(i,4),P_star(i,5),P_star(i,6)],P_star(i,1),
            P_star(i,2),P_star(i,3),'y');
    elseif mod(i,7) == 0
        cube_plot([P_star(i,4),P_star(i,5),P_star(i,6)],P_star(i,1),
            P_star(i,2),P_star(i,3),'k');
    end
end
cube_plot([0,0,0],x_star,y_star,z_star,'w');
daspect([1 1 1])
xlabel('X','FontSize',18)
```

```
ylabel('Y','FontSize',18)
zlabel('Z','FontSize',18)
h = gca;
material metal
alpha('color');
alphamap('rampup');
view(30,30);
hold off;
toc
```

The following functions are subroutines of the heuristic algorithm for designing boxes which include Loadbox, Find_index_x, Find_index_y, and Find_index_z.

```
function [temp1] = Loadbox(P,i,i1,i2,i3)
   if i1 > 0 && i2 > 0 && i3 > 0
       Candidate = [P(i1,4)+P(i1,1),P(i2,5)+P(i2,2),P(i3,6)+P(i3,3)];
       low = min(Candidate);
       index = find(Candidate == low);
       if length(index) == 3
           k = randi(3);
           if k == 1
               P(i,4) = P(i1,4)+P(i1,1);
               P(i,5) = P(i1,5);
               P(i,6) = P(i1,6);
           elseif k == 2
               P(i,4) = P(i2,4);
               P(i,5) = P(i2,5)+P(i2,2);
               P(i,6) = P(i2,6);
           else
               P(i,4) = P(i3,4);
               P(i,5) = P(i3,5);
               P(i,6) = P(i3,6)+P(i3,3);
           end
       elseif length(index) == 2
```

```
            k = randi(2);
            if index(k) == 1
                P(i,4) = P(i1,4)+P(i1,1);
                P(i,5) = P(i1,5);
                P(i,6) = P(i1,6);
            elseif index(k) == 2
                P(i,4) = P(i2,4);
                P(i,5) = P(i2,5)+P(i2,2);
                P(i,6) = P(i2,6);
            else
                P(i,4) = P(i3,4);
                P(i,5) = P(i3,5);
                P(i,6) = P(i3,6)+P(i3,3);
            end
        else
            if low == P(i1,4)+P(i1,1)
                P(i,4) = P(i1,4)+P(i1,1);
                P(i,5) = P(i1,5);
                P(i,6) = P(i1,6);
            elseif low == P(i2,5)+P(i2,2)
                P(i,4) = P(i2,4);
                P(i,5) = P(i2,5)+P(i2,2);
                P(i,6) = P(i2,6);
            else
                P(i,4) = P(i3,4);
                P(i,5) = P(i3,5);
                P(i,6) = P(i3,6)+P(i3,3);
            end
        end
    elseif i1>0 && i2>0 && i3<0
        if P(i1,4)+P(i1,1) == P(i2,5)+P(i2,2)
            k = randi(2);
            if k == 1
                P(i,4) = P(i1,4)+P(i1,1);
```

```
                    P(i,5) = P(i1,5);
                    P(i,6) = P(i1,6);
                else
                    P(i,4) = P(i2,4);
                    P(i,5) = P(i2,5)+P(i2,2);
                    P(i,6) = P(i2,6);
                end
            elseif P(i1,4)+P(i1,1) < P(i2,5)+P(i2,2)
                P(i,4) = P(i1,4)+P(i1,1);
                P(i,5) = P(i1,5);
                P(i,6) = P(i1,6);
            else
                 P(i,4) = P(i2,4);
                 P(i,5) = P(i2,5)+P(i2,2);
                 P(i,6) = P(i2,6);
            end
        elseif i1>0 && i2<0 && i3>0
            if P(i1,4)+P(i1,1) == P(i3,6)+P(i3,3)
                k = randi(2);
                if k == 1
                    P(i,4) = P(i1,4)+P(i1,1);
                    P(i,5) = P(i1,5);
                    P(i,6) = P(i1,6);
                else
                    P(i,4) = P(i3,4);
                    P(i,5) = P(i3,5);
                    P(i,6) = P(i3,6)+P(i3,3);
                end
            elseif P(i1,4)+P(i1,1) < P(i3,6)+P(i3,3)
                P(i,4) = P(i1,4)+P(i1,1);
                P(i,5) = P(i1,5);
                P(i,6) = P(i1,6);
            else
                P(i,4) = P(i3,4);
                P(i,5) = P(i3,5);
                P(i,6) = P(i3,6)+P(i3,3);
            end
        elseif i1<0 && i2>0 && i3>0
            if P(i2,5)+P(i2,2) == P(i3,6)+P(i3,3)
                k = randi(2);
                if k == 1
                    P(i,4) = P(i2,4);
                    P(i,5) = P(i2,5)+P(i2,2);
                    P(i,6) = P(i2,6);
                else
                    P(i,4) = P(i3,4);
                    P(i,5) = P(i3,5);
                    P(i,6) = P(i3,6)+P(i3,3);
                end
            elseif P(i2,5)+P(i2,2) < P(i3,6)+P(i3,3)
                P(i,4) = P(i2,4);
                P(i,5) = P(i2,5)+P(i2,2);
                P(i,6) = P(i2,6);
            else
                P(i,4) = P(i3,4);
                P(i,5) = P(i3,5);
                P(i,6) = P(i3,6)+P(i3,3);
            end
        elseif i1>0 && i2<0 && i3<0
             P(i,4) = P(i1,4)+P(i1,1);
             P(i,5) = P(i1,5);
             P(i,6) = P(i1,6);
        elseif i1<0 && i2>0 && i3<0
             P(i,4) = P(i2,4);
             P(i,5) = P(i2,5)+P(i2,2);
             P(i,6) = P(i2,6);
        else
             P(i,4) = P(i3,4);
            P(i,5) = P(i3,5);
            P(i,6) = P(i3,6)+P(i3,3);
        end
        temp1 = P;
end
```

```
function [temp2] = Find_index_x(Nl,P,i,xbar,ybar,zbar)
    Iipx = [];
    for ipx = Nl
        xfi = P(ipx,4)+P(ipx,1);
        yfi = P(ipx,5);
        zfi = P(ipx,6);
        for j = Nl
            if j==ipx
                continue
            end
            count = 0;
            if xfi+P(i,1) > P(j,4)
                count = count+1;
            end
            if P(j,4)+P(j,1) > xfi
                count = count+1;
            end
            if yfi+P(i,2) > P(j,5)
                count = count+1;
            end
            if P(j,5)+P(j,2) > yfi
                count = count+1;
            end
            if zfi+P(i,3) > P(j,6)
                count = count+1;
            end
            if P(j,6)+P(j,3) > zfi
```

```
                count = count+1;
            end
            if count == 6 || xfi+P(i,1)> xbar || yfi+P(i,2)> ybar || zfi+
                P(i,3)> zbar
                break
            end
        end
        if count == 6 || xfi+P(i,1)> xbar || yfi+P(i,2)> ybar || zfi+P(i
            ,3)> zbar
            continue
        else
            Iipx = [Iipx ipx];
        end
    end
    if isempty(Iipx)
        i1 = -1;
    else
        XIipx = P(Iipx,4)+P(Iipx,1);
        miin = min(XIipx);
        indexx = find(XIipx == miin);
        if length(indexx) == 1
            i1 = Iipx(indexx);
        else
            candidate = Iipx(indexx);
            zcan = P(candidate,6);
            [zmin,izmin] = min(zcan);
            i1 = candidate(izmin);

        end
    end
    temp2 = i1;
end
```

```
function [temp3] = Find_index_y(Nl,P,i,xbar,ybar,zbar)
    Iipy = [];
    for ipy = Nl
        xfi = P(ipy,4);
        yfi = P(ipy,5)+P(ipy,2);
        zfi = P(ipy,6);
        for j = Nl
            if j==ipy
                continue
            end
            count = 0;
            if xfi+P(i,1) > P(j,4)
                count = count+1;
            end
            if P(j,4)+P(j,1) > xfi
                count = count+1;
            end
            if yfi+P(i,2) > P(j,5)
                count = count+1;
            end
            if P(j,5)+P(j,2) > yfi
                count = count+1;
            end
            if zfi+P(i,3) > P(j,6)
                count = count+1;
            end
            if P(j,6)+P(j,3) > zfi
                count = count+1;
            end
            if count == 6 || xfi+P(i,1)> xbar || yfi+P(i,2)> ybar || zfi+
                P(i,3)> zbar
                break
            end
        end
        if count == 6 || xfi+P(i,1)> xbar || yfi+P(i,2)> ybar || zfi+P(i
            ,3)> zbar
            continue
        else
            Iipy = [Iipy ipy];
        end
    end
    if isempty(Iipy)
        i2 = -1;
    else
        YIipy = P(Iipy,5)+P(Iipy,2);
        miin = min(YIipy);
        indexy = find(YIipy == miin);
        if length(indexy) == 1
            i2 = Iipy(indexy);
        else
            candidate = Iipy(indexy);
            zcan = P(candidate,6);
            [zmin,izmin] = min(zcan);
            i2 = candidate(izmin);
        end
    end
    temp3 = i2;
end
```

```
function [temp4] = Find_index_z(Nl,P,i,xbar,ybar,zbar)
    Iipz = [];
    for ipz = Nl
        xfi = P(ipz,4);
        yfi = P(ipz,5);
        zfi = P(ipz,6)+P(ipz,3);
        for j = Nl
            if j==ipz
            continue
        end
        count = 0;
        if xfi+P(i,1) > P(j,4)
            count = count+1;
        end
        if P(j,4)+P(j,1) > xfi
            count = count+1;
        end
        if yfi+P(i,2) > P(j,5)
            count = count+1;
        end
        if P(j,5)+P(j,2) > yfi
            count = count+1;
        end
        if zfi+P(i,3) > P(j,6)
            count = count+1;
        end
        if P(j,6)+P(j,3) > zfi
            count = count+1;
        end
        if count == 6 || xfi+P(i,1)> xbar || yfi+P(i,2)> ybar || zfi+P(i,3)> zbar
            break
        end
    end
    if count == 6 || xfi+P(i,1)> xbar || yfi+P(i,2)> ybar || zfi+P(i,3)> zbar
        continue
    else
        Iipz = [Iipz ipz];
    end
end
if isempty(Iipz)
```

```
        i3 = -1;
    else
        ZIipz = P(Iipz,6)+P(Iipz,3);
        [miin,imiin] = min(ZIipz);
        i3 = Iipz(imiin);
    end
    temp4 = i3;
end
```

APPENDIX B : Matlab code of the heuristic algorithm for minimizing the number of types of packing boxes.

```
tic
Dimension = importdata('sizeset1.txt');
L = Dimension(:,1);
W = Dimension(:,2);
H = Dimension(:,3);
n = length(L);
t = 0.20;
Num = 50;
Tstar = []; fstar = n;
T = -ones(n,n);

for i = 1:n
    T(i,i) = 0;
end
for i = 1:n
    for j = 1:n
        if L(i)-L(j)<0 || W(i)-W(j)<0 || H(i)-H(j)<0
            T(i,j) = 0;
        end
    end
end
for i = 1:n
    for j = 1:n
        if L(i)-L(j)> t*L(i) || W(i)-W(j)> t*W(i) || H(i)-H(j)> t*H(i)
            T(i,j) = 0;
        end
    end
end
T;
Td = T;
for m = 1:Num
```

```
    for j = 1:n
        I = find(Td(:,j)==-1);
        if isempty(I);
            continue
        else
            a = I(randi(length(I)));
            Td(:,j)= 0; Td(a,j) = 1;
            Td(j,:)= 0; Td(:,a) = 0;
        end
    end
    f = n-sum(sum(Td));
    if f < fstar
        fstar = f;
        Tstar = Td;
    end
    Td = T;
end
fstar
Tstar;
toc
```

APPENDIX C : IBM ILOG OPL CPLEX code for the binary integer linear programming model for minimizing the number of types of packing boxes.

This section includes the code for import and export data and the code of the binary integer linear programming model for minimizing the number of types of packing boxes.

```
 SheetConnection filein("Size of box.xlsx");
 L from SheetRead(filein, "Sheet2!F2:F51");
 W from SheetRead(filein, "Sheet2!G2:G51");
 H from SheetRead(filein, "Sheet2!H2:H51");

 SheetConnection fileout("Val Tij.xlsx");
 T to SheetWrite(fileout, "Sheet3!B2:AY51");
```

```
/*** Parameters ***/
 int n = 50;
 range R = 1..n;
 int M = 1000;
 float ep = 0.1;
/*** Input ***/
 float L[R] = ...;
 float W[R] = ...;
 float H[R] = ...;
 float t = 0.05;
/*** Decision Variable ***/
 dvar boolean T[R][R];
 dvar boolean y[R][R];
 dvar boolean a[R][R];
 dvar boolean b[R][R];
 dvar boolean c[R][R];
 dvar boolean d[R][R];
/*** Objective_function ***/
 minimize n - sum(j in R)sum(i in R)T[i][j];
```

```
/*** Constraints ***/
 subject to {
   forall (i in R)
     T[i][i] == 0;
   forall(i in R)
     forall (j in R)
       abs(L[i]-L[j])+abs(W[i]-W[j])+abs(H[i]-H[j])-(L[i]-L[j])-(W[i]-W
            [j])-(H[i]-H[j])<=M*(1-y[i][j]);
   forall(i in R)
     forall (j in R)
       T[i][j] <= y[i][j];
   forall(i in R)
     forall (j in R)
       (L[i]-L[j])-t*L[i] <= M*(1-a[i][j]);
   forall(i in R)
     forall (j in R)
       T[i][j] <= a[i][j];
   forall(i in R)
     forall (j in R)
       (W[i]-W[j])-t*W[i] <= M*(1-b[i][j]);
   forall(i in R)
     forall (j in R)
       T[i][j] <= b[i][j];
   forall(i in R)
     forall (j in R)
       (H[i]-H[j])-t*H[i] <= M*(1-c[i][j]);
   forall(i in R)
     forall (j in R)
       T[i][j] <= c[i][j];
   forall(j in R)
     sum(i in R)T[i][j] <= 1;
   forall(i in R)
    forall(j in R)
      T[i][j]-1 <= (1-d[i][j])*M-ep;
```

```
  forall(i in R)
    forall(j in R)
      forall(k in R)
        T[j][k] <= d[i][j];
  forall(i in R)
    forall(j in R)
      forall(k in R)
        T[k][i] <= d[i][j];
}
```

BIOGRAPHY

Name	Mr. Teeradech Laisupannawong
Date of Birth	23 November 1993
Education	B.S. (Mathematics) (First-Class Honours), 2016
	Department of Mathematics, Faculty of Science,
	Kasetsart University

Publication

- T. Laisupannawong, B. Intiyot, and P. Thipwiwatpotjana, “Mixed integer linear programming model for reducing types of packing boxes”, *Proceedings Operations Research Network Conference 2018*, Pattaya, Thailand, 23–24 April 2018, pp. 96-100.