Three-Dimensional Bin Packing in Mixed-case Palletization

by

Yi Feng Yan

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Management Sciences

Waterloo, Ontario, Canada, 2015

Abstract

The Three Dimensional Bin Packing Problem (3DBPP) is within one of the broad categories of the Bin Packing Problem. The other broad categories include the One Dimensional and the Two Dimensional Bin Packing Problem. As we live in a three dimensional world, the 3DBPP can model a variety of real world problems. Some of the popular applications of the 3DBPP include the Container Loading Problem and the Pallet Packing Problem. The objective of the 3DBPP is to minimize the number of containers or pallets used given a certain number of items, while respecting the non-overlapping constraints along all three dimensions. The Open Dimension Problem (ODP), is a special case of the 3DBPP, where a given set of cargo is packed onto a single container, with one or more variable dimensions. The Single Bin Size Bin Packing Problem (SBSBPP) is another special case, where a given set of cargo is packed in bins of the same size, with the objective of minimizing the number of bins used. The SBSBPP is more difficult to solve than the ODP, as items are packed in multiple bins in the SBSBPP and in only one bin in the ODP.

In this thesis, we first propose a mixed-integer programming model for the ODP, where the objective is to minimize the highest point within the bin. We then provide a number of enhancements to improve the model. Later, a number of heuristics are proposed to find good feasible solutions within reasonable computational time. Finally the solution of the ODP is used to provide a solution to the SBSBPP.

The proposed approach is compared to well-known approaches from the literature on a standard data set. The approach was able to give reasonably good solutions to most instances within a given time frame, especially when the number of items per bin increases.

Acknowledgements

First I would like to thank my supervisers, Dr. Samir Elhedhli and Dr. Fatma Gzara, for their vigorous style, professional guidance and insightful support. I feel honoured just working along side them.

Second I would like to thank my parents for their continuous support and unconditional love.

Lastly I would like to acknowledge all the staff members of Management Sciences here at the University of Waterloo, for creating such a productive research environment. A big thank you goes to Dr. Da Lu, with his helps and hints, I was able to save an enormous amount of time on testing and debugging.

Dedication

This is dedicated to the ones I care about and love, especially my mother, who has supported me enormously.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the area of Operations Research and Optimization, the Bin Packing problem (BPP) is a well-known problem that has many real-life applications. In the problem, objects of different attributes are to be packed in the least number of bins.

The one-dimensional version of the bin packing problem has been widely studied in the literature. In this kind of problem, items of different weights are packed in fixed-capacity bins. The objective is to minimize the number of bins used while respecting the weight constraints. In the two dimensional bin packing problem, items with different length and width properties are packed in fixed area rectangles. According to Lodi et al. [13], applications of the two-dimensional bin packing problem include cutting and packing in the wood, glass, and cloth industries, goods shelving in warehousing, and newspaper paging in the publishing industry. Moreover, layout optimization is another major application.

The focus of this thesis is on the three-dimensional bin packing problem, where three-dimensional items cannot overlap and must fit within three-dimensional bins. The three-dimensional bin packing problem has a variety of applications in logistics and transportation, including, but not limited to, container loading and pallet loading problems. Because of its practical nature, the three-dimensional bin packing problem has been studied well in the OR literature. With respect to the inclusion of practically-relevant constraints however, the research done is still at the very beginning. Heuristic approaches, in particular meta-heuristics, are remain the most important class of algorithms for solving practical three-dimensional bin packing problems, as they are able to provide quality solutions in a reasonable amount of time.

The rest of this thesis is organized as follows. Chapter 2 is a detailed literature review. It summarizes the relevant literature in the order of practical constraint types, model formulations, as well as heuristic approaches. In Chapter 3, we propose an exact mathematical model to solve the Open-Dimension Problem (ODP), and add several improvements. In Chapter 4, a new heuristic approach called the “Stacking Heuristic” is introduced to reduce the run-time. Furthermore, a “Layered Heuristic” is employed to solve the Single Bin-size Bin Packing Problem (SBSBPP). In Chapter 5, we use the standard test instances from Martello et al. [14] and compare the Layered Heuristic to some well-known algorithms and meta-heuristics. We conclude this thesis in Chapter 6 and give some future research directions.

Chapter 2

Literature Review

This chapter reviews recent and relevant research on the subject of the 3DBPP. We first examine the different constraint types that can be included in the 3DBPP, then review exact solution methods proposed in the literature. We finally review heuristic solution approaches for the 3DBPP.

According to Wäscher et al. [19], seven sub-categories exist within the 3DBPP. The Single Stock-size Cutting Stock problem(SSSCSP) is suitable when the items to be loaded are weakly heterogeneous in size and the bins are identical. The Multiple Stock-size Cutting Stock problem(MSSCSP) is very similar to the SSSCSP, with the only difference being the bins are weakly heterogeneous. By their definition, the term "weakly heterogeneous" means that items or bins can be grouped into relatively few classes, for which they are identical with respect to shape and size. The Residual Cutting stock problem(RCSP) is best used when the item sizes are weakly heterogeneous, but the bins are highly heterogeneous. By definition, the term “highly heterogeneous” means only very few elements are of identical shape and size. The Single Bin-Size Bin Packing Problem(SBSBPP) describes the situation where the item sizes are highly heterogeneous, but the bins are identical. The Multiple Bin-size Bin Packing problem is the same as the SBSBPP except that the bin sizes are weakly heterogeneous. If both items and bin sizes are highly heterogeneous, it is called a Residual Bin Packing Problem(RBPP). The Open Dimension Problem(ODP) refers to packing items into a single container of which one or more dimensions are variable.

2.1 Practical Constraint Types

According to Bortfeldt and Wäscher [4], most approaches proposed in the literature lack practical value as they do not pay enough attention to constraints encountered in practice. They summarize and categorize the types of constraints that are commonly seen when modeling the problem. Container (Pallet)-related constraints include weight-limit and weight distribution, where the first imposes an overall limit on the weight of the container, and the second requires the weight of the items to be spread out as evenly as possible on the container (pallet) floor. The second type of constraints are item-related constraints. They include loading priorities, orientation constraints, and stacking constraints. The loading priorities decide which items must be loaded first. The orientation constraints can restrict the items in vertical and/or horizontal directions. The stacking constraints restrict the way items can be placed on top of each other. The third are cargo related constraints; they include complete shipment, and allocation constraints. The complete shipment constraints force the loading of items belonging to the same shipment. The allocation constraints prohibit some items from being placed in the same container, e.g. perfumes and foods. The fourth are positioning constraints. For example, large items are required to be placed in the corner of a pallet. The fifth are vertical and horizontal stability constraints as well as complexity constraints. The vertical stability constraint ensures that items withstand gravitational force and prevents them from falling down. The horizontal stability constraints refer to the ability of the items to withstand their inertia. For the complexity constraints, the most prominent one is the guillotine cutting constraint. A guillotine pattern represents a pattern that can be described and packed easily. However, guillotine patterns are often not acceptable in pallet loading where they would require additional operations like shrink-wrapping or inter-locking to secure the items.

2.2 Model Formulations

For the general three-dimensional bin-packing problem, Chen et al. [5] give a zero-one mixed integer programming model that is guaranteed to lead to an optimal solution. The problem involves placing non-uniform rectangular items into unequal-sized containers. The model includes orientation constraints, multiple item sizes and container sizes, as well as weight balance. However, the model was found inefficient when the number of items increased. The number of variables and constraints grow at rates of $3n^2$ and $3.5n^2$, with n representing the number of items.

Building onto the model proposed by Chen et al. [5], Wu et al. [20] formulated a single container problem with variable bin height. The authors tested the algorithm on small bin and large bin setups. The items are of 10 different types and occasionally some items have customized sizes. The authors consider moderate heterogeneity. After allowing the Cplex solver to run a maximum of two hours, the result is compared to the actual stacking height of the items, where experienced operators use rule-of-thumb to decide the packing patterns. Although the algorithm is proven to generate better patterns than human operators, a run-time of two hours is inappropriate for guiding on-site operations. The authors then proposed a genetic algorithm based heuristic. The heuristic is shown to be more efficient when packing items into large bins as opposed to small bins. Run time of the heuristic was no more than one minute.

Up to this point the possible packing patterns discussed are referred to as orthogonal packing patterns in the literature, see Den Boef et al. [8], in which the boxes are orthogonally packed with their edges parallel to the container edges. Two special cases of packing patterns are Guillotine and Robot Packable patterns.

According to Bortfeldt and Wäscher [4], a pattern is guillotine-cuttable if a cut parallel to the container faces can divide the boxes into two disjoint subsets (also called cut slices) and no box is split by the cut. The cutting of the boxes are done recursively in stages; in each stage all cuts must be parallel. A cut slice is in the k th stage if it has depth k in the recursion tree. Figure 2.1 shows a two-dimensional example with a two-stage guillotine packing. In the first stage, cuts labeled 1 and 2 are performed on the boxes to form three slices. Similarly in the second stage, cuts 3, 4, and 5 cut each slice entirely. Figure 2.2 shows a packing that is not guillotine-cuttable.

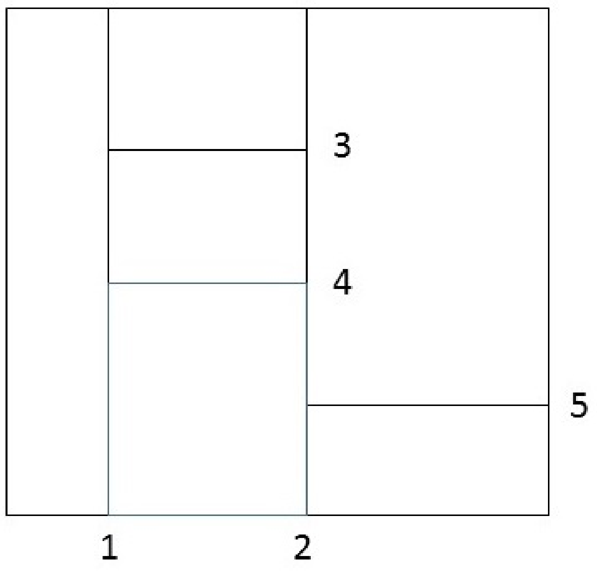

Figure 2.1: Guillotine-Cuttable Packing.

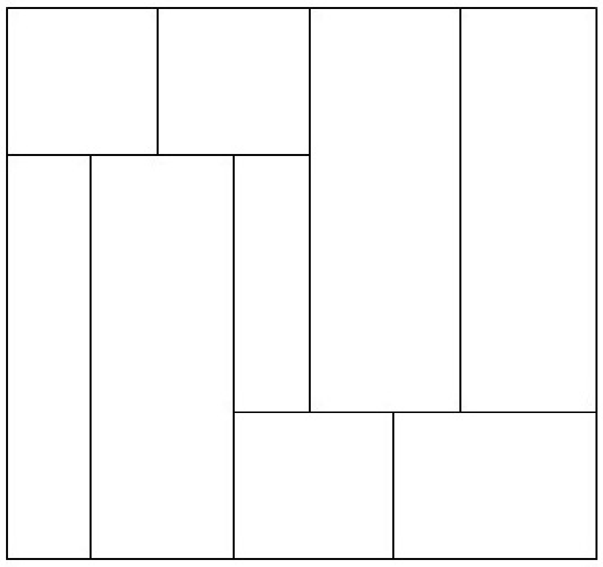

Figure 2.2: Non Guillotine-Cuttable Packing.

Amossen and Pisinger [1] refer to robot packing as a subset of guillotine packings. Robots used for packing boxes in the industry are equipped with a mechanical hand to lift the boxes. To avoid collision with already packed boxes, the packed boxes cannot be in front of, to the right of, or above the destination of the boxes the robot is currently placing. Each guillotine packing can be translated into a robot packing. In the three-dimensional case, by first placing items in the bottom, rear, and left, a feasible robot packing can be generated.

2.3 Heuristic Approaches

Because of the slow running time of exact methods, many heuristic approaches are developed to solve complex problems. In this section, we review several such approaches, including random search, prototype column generation, extreme point-based, finite enumeration based, and tabu search based heuristics.

Bischoff [2] describes a heuristic based on random search with scoring rules to find solutions that takes limited load bearing strength into account. The scoring represents a weighted sum of the area utilization of the loading surface and a proxy measure of the ability of the new layer to take on additional weights. Therefore, the score obtained takes into account how well the additional layer would fill the space available as well as how it would affect further placements. The weight of scores given to the two components must be specified by the user. In the sample data, all boxes are assumed to have equal density, hence the weight of boxes depends on their volume. The sizes of the samples range from 100 items to well over 300.

Zhu et al. [21] propose a prototype column generation strategy for the multiple container loading problem. They first show that the pricing sub-problem of utilizing column generation technique is NP-hard. Then they discuss the benefit of using prototypes to approximate the pricing problem feasible solutions and substitute prototypes by feasible columns in later iterations. In the model, they consider vertical stability constraints in both the fully supported case and the partially supported case and formulated a set-covering problem. To generate the prototypes, they employ a two phase method. The first phase uses an iterative construction approach, and the second phase takes the result from the first phase and utilizes a hill-climbing algorithm to find the best prototype column. The algorithm is tested against the standard 700 data set generated by Bischoff and Ratcliff [3], which can be accessed at http://people.brunel.ac.uk/ mastjjb/jeb/info.html.

Crainic et al. [6] proposed an extreme point-based heuristic for the three-dimensional bin packing problem. For any given packing, the extreme points, which are the corner points generated by placing an item in the bin, can be computed in polynomial time, and is independent of a particular packing problem. They first sort the items into clusters according to some rules on items height, base-area and volume. Then, an extreme point best fit decreasing heuristic is applied to the item clusters. The heuristic evaluates a merit function of each extreme point that can accommodate a new item. The heuristic is tested on the standard instances from Martello et al. [14] where each sample contains 20 to 100 items. The authors claim improvement over existing constructive based approaches through the use of a new definition for corner points and item placement rules.

Faina [9] utilizes a geometric model that reduces the general three-dimensional packing problem to a finite enumeration scheme. First a geometric procedure is introduced to generate a particular finite class of placings, then it is proved analytically that the set of all feasible placings will not provide a better solution. A simulated-annealing approach called zone$_{3D}$ is devised to search for an optimal solution.

Mohanty et al. [17] propose a heuristic based on fractional knapsack that maximizes the total value of items packed in a bin. Pimpawat and Chaiyaratana [18] present a co-operative and co-evolutionary based genetic algorithms.

Hifi et al. [12] uses a linear programming approach to solve the 3D-SBSBPP without employing any metaheuristics. For smaller item instances, they proposed an optimization model that chooses a subset of the entire item set that will best fit the active bin. For larger item instances, they proposed a greedy heuristic based on item starting order to select the items. In the method proposed, there are two phases, a selection phase and a placement phase. The placement phase finds the largest number of items that can fit in the current bin. Items already placed are removed from the item set, and the heuristic is repeated until all items are placed.

The two-level tabu search (TS2 pack) heuristic of Crainic et al. [7] attempts to separate the task of determining feasibility and optimality by making decisions at two levels. The first level heuristic deals with the optimality of the problem and the second level heuristic tries to find feasible packings for the items assigned to the bins. Both are tabu search based. To generate an initial solution, the Extreme-Point-First-Fit-Decreasing EP-FFD heuristic proposed by Crainic et al. [6] was utilized. The EP-FFD is based on the first-fit decreasing rule. Items are first sorted by non-increasing volume, then placed one by one into the extreme points. If a bin is filled, a new bin is used. Upon obtaining an initial solution, the TS2 pack heuristic will discard the bin with the worst fitness value, and the items in it are iteratively assigned to the bins with the best fitness value. During this procedure, the height (z-axis) constraints are relaxed, if the solution is feasible, it is regarded as the current best and the emptied bin is discarded. If the solution is not feasible, the first stage tabu search algorithm called ACC-TS is executed. This heuristic works on the items-to-bins assignment without forcing the bin size constraints. Instead, it penalizes infeasible packings that are larger than the bin size in the objective function. The inner heuristic IG-TS is used by ACC-TS to check the feasibility and optimize the packing in order to satisfy the bin size constraints. ACC-TS uses a local-search neighbourhood whose size and accuracy are dynamic by means of a k-chain-moves procedure. It also includes a diversification phase to explore new solution space. To check the feasibility of a packing, IG-TS uses the implicit representation given by the Interval Graph approach proposed by Fekete and Schepers [10, 11]. They defined 7 overlapping rules that can be used in IG-TS to alternate the spatial relationship between any two items. The algorithm stops when a packing that respects the bin dimensions is obtained or the maximum number of iterations is reached.

Chapter 3

The Model

In this chapter, we formally define the problem under study, provide a formulation based on mixed-integer programming, and propose several improvements.

3.1 Problem Definition

Mixed-case pallet packing problems refer to the stacking of items, which can be weakly or strongly heterogeneous, into pallets of fixed area. Possible objectives include minimizing the number of pallets used, minimizing the unused space of within pallets, and maximizing the total value of items packed. Constraints are mainly physical, related to non-overlapping and the dimensions of the pallets, and practical, related to the packing process and the nature of the items.

3.2 A Grid-based formulation for the Open Dimension Problem

To model the Open Dimension Problem (ODP), the container/pallet can be divided to a three-dimensional grid, which divides the space to many small cubes. Items can be placed only at the corners of the cubes, thereby limiting the search space. Therefore, the problem is tranformed to a special layout problem with height consideration. The model uses the relative positioning concept from Meller et al. [16]. In using this approach, the cube size setting is crucial to ensure fast execution and efficient stacking layout. If the cubes are too large, it will greatly reduce the search space and eliminate many good packing patterns. Conversely, if the cubes are too small, the search space is not reduced enough and finding a good packing pattern may take a long time. The objective is to minimize the highest point of all items in the bin, which should give us a tight packing by pushing all items toward the bottom of the bin. Next we define the parameters and the decision variables.

Indices:

- i, j: indices for items, $i, j \in N$.
- s: index for the x, y, or z directions.

Parameters:

- l_i^s: the length of item i along direction s.
- L^s: the length of the pallet along direction s.

Decision Variables:

- c_i^s: the coordinate of item i along direction s.
- $z_{ij}^s = \begin{cases} 1, & \text{if item } i \text{ precedes item } j \text{ along direction } s, \\ 0, & \text{otherwise.} \end{cases}$
- h: highest point of all items in the bin.

The formulation is:

$$
\begin{aligned}
\min \quad & h \\
\text{s.t.} \quad & \sum_{s=x}^{z} (z_{ij}^s + z_{ji}^s) \geq 1 \quad \forall j > i, \quad i, j \in N && (1) \\
& z_{ij}^s + z_{ji}^s \leq 1 \quad \forall j > i, \quad i, j \in N, \quad s \in \{x, y, z\} && (2) \\
& c_i^s + l_i^s \leq c_j^s + L^s(1 - z_{ij}^s) \quad \forall i \neq j, \quad i, j \in N, \quad s \in \{x, y, z\} && (3) \\
& 0 \leq c_i^s \leq L^s - l_i^s \quad \forall i, \quad i \in N, \quad s \in \{x, y, z\} && (4) \\
& c_i^z + l_i^z \leq h \quad \forall i, \quad i \in N && (5) \\
& z_{ji}^s \in \{0, 1\} \quad \forall i \neq j, \quad i, j \in N, \quad s \in \{x, y, z\} && (6) \\
& h, c_i^s \geq 0 \quad \forall i, \quad i \in N, \quad s \in \{x, y, z\} && (7)
\end{aligned}
$$

Constraints (1) state that there is at least one spatial relationship between any two items. Constraints (2) enforce that an item cannot precede and follow another item in direction s. Constraints (3) are the non-overlapping constraint given the relative positions determined by z_{ij}^s. Constraints (4) keep items within the pallets boundaries. Constraints (5) find the maximum height of the pallet. Constraints (6) and (7) are the standard binary and non-negativity constraints.

When testing the grid-based model, we found that some items maybe unsupported and are suspended in air. This is caused by the simplicity in the objective function, which only considers minimizing the overall height of the pallet, without considering the support needed by each item. To tackle this issue, two improved models are proposed next. The basic idea in the first improved model is to cut the pallet space into horizontal slices, where the filled space is increasing as we progress from the top slice to the bottom slice. Moreover, we include a scoring rule in the objective function so that unfilled space in bottom slices is penalized more than in top slices. The second improved model includes the newly added constraints from the first model as well as explicit relationships between relative position variables.

3.2.1 The Grid-Based Formulation with Slicing

First, let us introduce some additional parameters and decision variables, let n be the number of slices:

Additional Indices:

- k: index for slices, $k \in n$.

Additional Parameters:

- ϵ: a very small number.

Additional Decision Variables:

- $B_i^k = \begin{cases} 1, & \text{if part of item } i \text{ is contained in slice } k, \\ 0, & \text{otherwise.} \end{cases}$
- s_k: the ceiling height of slice k.
- S_k: total used space in slice k.

The grid-based formulation with pallet space slicing is:

$$
\begin{aligned}
\text{min} \quad & h - \sum_{k=1}^{n} \frac{(n-k)S_k}{L^x L^y} \\
\text{s.t.} \quad & (1),(2),(3),(4),(5),(6),(7) \text{ and} \\
& -c_i^z + (s_k - l_i^z)B_i^k \leq 0 \quad \forall i,k, \quad i \in N, \quad k \in \{1..n\} && (8) \\
& c_i^z + L_z B_i^k \leq s_k - \epsilon + L_z \quad \forall i,k, \quad i \in N, \quad k \in \{1..n\} && (9) \\
& -S_k + S_{k+1} \leq 0 \quad \forall k, \quad k \in \{1..n-1\} && (10) \\
& \sum_{i=1}^{I} B_i^k (l_i^x l_i^y) = S_k \quad \forall k, \quad k \in \{1..n\} && (11) \\
& \sum_{k=1}^{n} B_i^k = l_i^z \quad \forall i, \quad i \in N && (12) \\
& B_i^k \in \{0,1\} \quad \forall i \neq j, \quad i,j \in N, \quad s \in \{x,y,z\} && (13)
\end{aligned}
$$

The objective function seeks to minimize the overall height of the pallet and at the same time eliminate unfilled space in the bottom slices by giving a higher score to filled space in bottom slices than top slices. The filled space are reduced by a factor of $L^x L^y$ to ensure the score is measured with the same unit as h in the objective function. Constraints (8) and (9) are used to determine if all or part of an item i is contained in slice k. Constraints (10) ensure the lower slices have a greater filled space than upper slices. Constraints (11) calculate the filled space in each slice k. Because each slice has unit height, constraints (12) state each item i is contained in exactly l_i^z slices.

3.2.2 Grid-Based Formulation with Slicing and Relative Position Transitivity

In this model, additional cuts are added to model the fact that if item i is above item j, and j is above l, then item i is above item l. These constraints are:

$$z_{ij}^s + z_{jl}^s - z_{il}^s \leq 1 \quad \forall i, j, l, \ \ where\, i \neq j \neq l, \ \ i, j, l \in N, \ \ \forall s, \ \ s \in \{x, y, z\} \tag{14}$$

The comparison between the three models is presented in Chapter 5.

Chapter 4

Heuristic Approaches

The direct solution of the model presented in the previous chapter take a long time to solve for practical problems. In this chapter, we resort to heuristics to find good solutions with shorter run times.

4.1 The Stacking Heuristic

This heuristic attempts to solve the problem sequentially by solving the grid-based formulation with slicing for small subsets of items. The main idea can be described as the following:

(1) select a small subset of the items to be packed,

(2) solve the grid-based formulation with slicing to optimality or within a time limit,

(3) fix the position of the previously selected items,

(4) Pick a new subset and solve the model with the previous items fixed,

(5) repeat until all items are packed.

The success of this approach greatly depends on the choice of the item subsets. Generally speaking, we want to save the smaller items for later iterations so that they can be used to fill in the fragmented spaces created by the larger items. However, we also do not want to pack all the large items first, because that will create too many fragmented spaces. An optimal method to split the items into small subsets is yet to be determined.

The objective function needs to be changed so that items that are closer to the origin of the x-y plane will get a higher score. This change should eliminate some of the fragmented spaces since all items are being pushed toward the origin. Figure 4.1 provides an illustration of the heuristic approach. The example used has 80 objects. Each item subset contains 16 objects, therefore a total of 5 iterations are performed. Each sub-figure demonstrates the resulting packing generated after each iteration.

The model could be enhanced by trying to eliminate fragmented space. This is acheved by pusing all items towards the origins and minimizing the z^x_{ji} and z^y_{ji} coordinates of items.

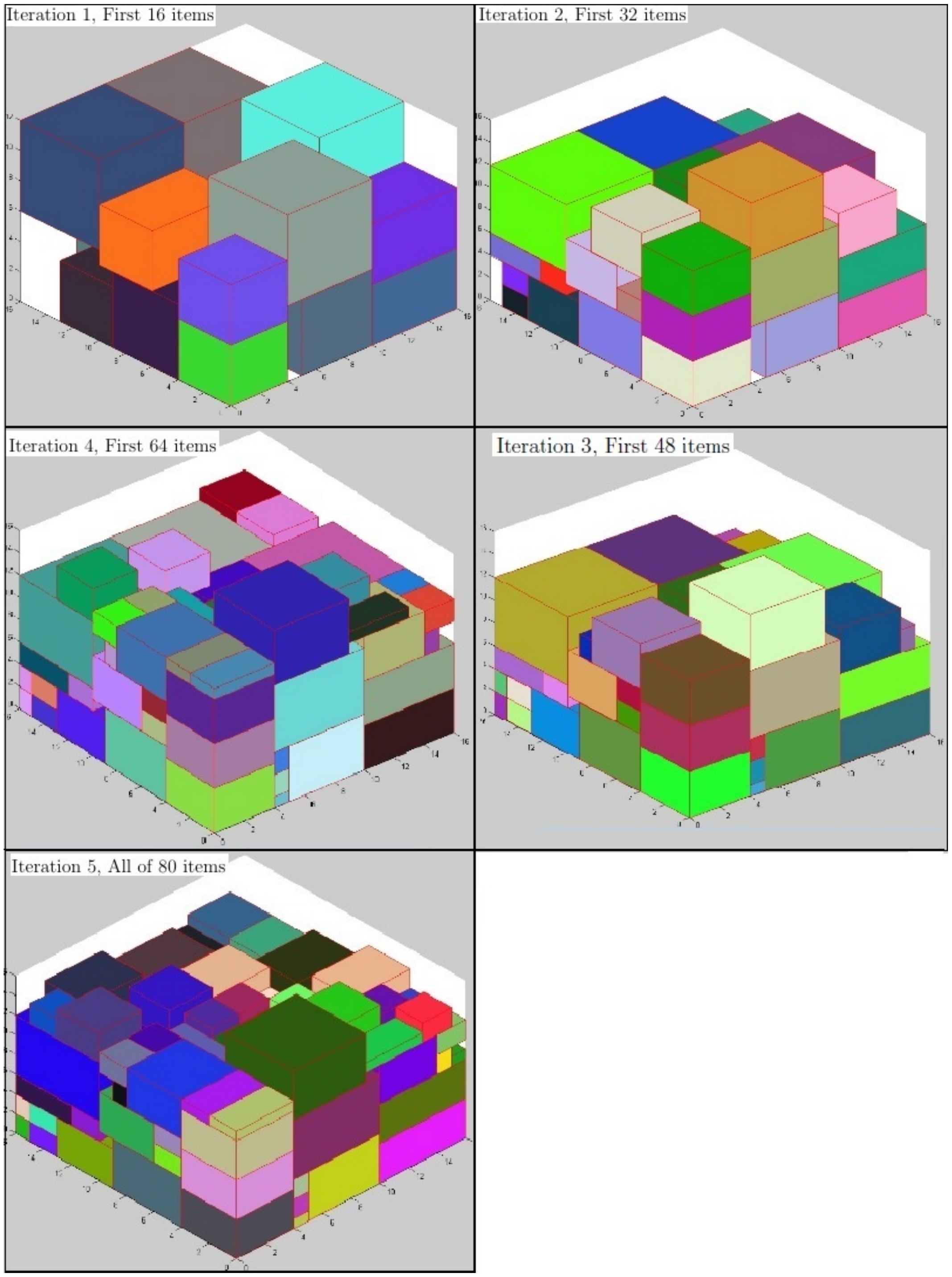
Iteration 1, First 16 items
Iteration 2, First 32 items
Iteration 4, First 64 items
Iteration 3, First 48 items
Iteration 5, All of 80 items

Figure 4.1: Illustrations of using the Stacking Heuristic to pack 80 items in 5 iterations.

4.2 A Layered Heuristic for SBSBPP

So far we have only considered building a single pallet. However, in reality we must respect the height limit of the pallets. We introduce a divider item to the previous models. This involves the use of "divider" items at specific positions along the z-axis. Figure 4.2 illustrates the idea by placing thin slices along the vertical axis. After the dividers are created to separate all the pallets, we are ready to place all the items from the bottom up using the stacking heuristic.

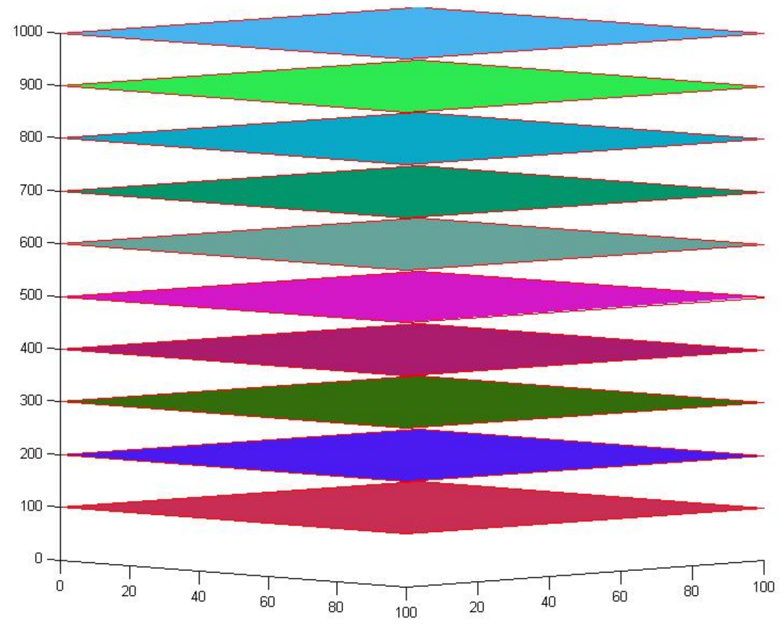

Figure 4.2: An illustration of the Layered Heuristic using 10 Dividers.

Chapter 5

Numerical Testing

The code was implemented in Matlab, with Cplex as the solver. The code was run on a computer with an Intel Core i7 4770 cpu and 8 gigabyte of ram.

5.1 Comparison and Visualization of the Formulations in Chapter 3

All three models presented in Chapter 3 are tested on a sample problem with 30 items and a pallet of size $6 \times 6 \times 15$. The items vary in sizes, including $1 \times 1 \times 1$, $2 \times 1 \times 1$, $1 \times 2 \times 2$, $2 \times 2 \times 1$, $2 \times 1 \times 2$, $2 \times 2 \times 2$, $3 \times 2 \times 2$, $2 \times 2 \times 3$, $3 \times 3 \times 2$, $3 \times 3 \times 3$, $4 \times 2 \times 2$. A computational time limit of 200 seconds is used for the Cplex solver. The packing patterns generated using the grid-based formulation are illustrated in figures 5.1 and 5.2. It is obvious that although the overall height is minimized, the entire packing is not very compact, and several items are fully or half suspended in the air.

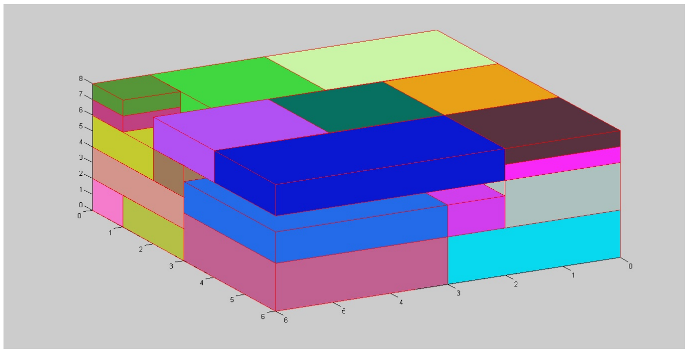

Figure 5.1: Packing generated by the grid-based formulation; front 45 degree view.

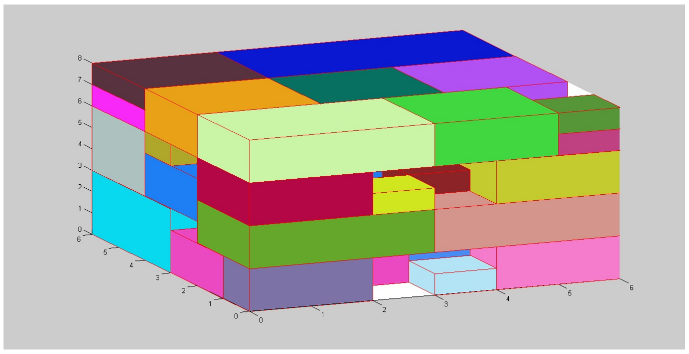

Figure 5.2: Packing generated by the grid-based formulation; rear 45 degree view.

Figure 5.3 and 5.4 display the packing obtained using the grid-based formulation with slicing. The packing still has the same height as in Figure 5.1 and 5.2. However, the unfilled space is eliminated completely in the bottom five slices. There is only one item not being fully supported.

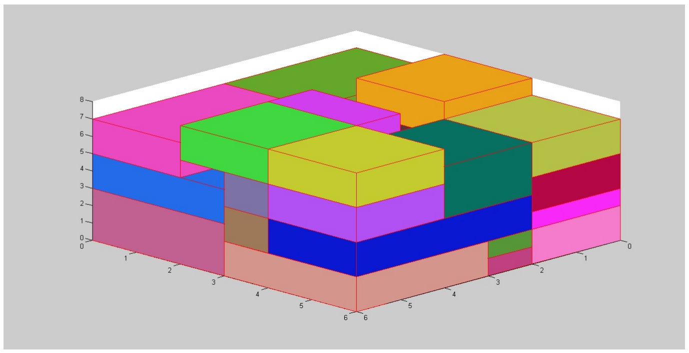

Figure 5.3: Packing generated by the grid-based formulation with slicing; front 45 degree view.

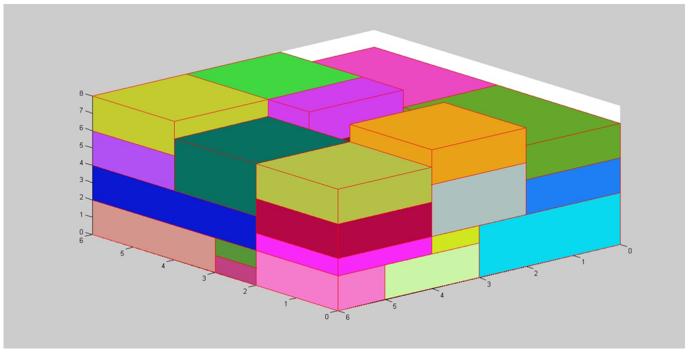

Figure 5.4: Packing generated by the grid-based formulation with slicing; back 45 degree view.

One disadvantage of the first improved formulation is the increased computational time. Whereas the original model takes less than 20 minutes to find the last solution, the improved model takes more than 3 hours.

The effect of adding the transitivity constraints is counter intuitive. It takes longer to find a good solution. The increase in run time comes from the large size of the added constraints, which is the order of the number of items cubed. One advantage is that Cplex takes only few iterations to find a feasible solution. Figure 5.5 displays the resulting packing.

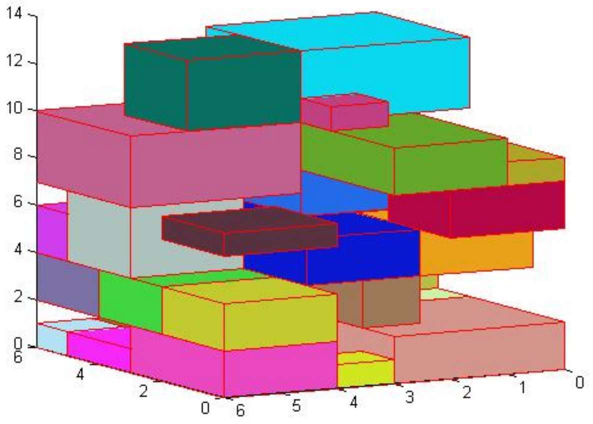

Figure 5.5: Packing generated by the grid-based formulation with slicing and transitivity constraints.

5.2 Testing the Layered Heuristic

In this section, we test the layered heuristic of section 4.3 on the standard instances from Martello et al. [14]. A total of 9 different item classes are generated, with each item class containing 10 instances. For the first five classes, the bin dimensions are $Width(W) = Height(H) = Depth(D) = 100$ and five types of items are uniformly randomly generated with their sizes in different intervals. The items are generated with w_j, h_j, d_j as the width, height and depth of each item j, as follows:

Type 1: $w_j \in [1, \frac{1}{2}W], h_j \in [\frac{2}{3}H, H], d_j \in [\frac{2}{3}D, D]$,

Type 2: $w_j \in [\frac{2}{3}W, W], h_j \in [1, \frac{1}{2}H], d_j \in [\frac{2}{3}D, D]$,

Type 3: $w_j \in [\frac{2}{3}W, W], h_j \in [\frac{2}{3}H, H], d_j \in [1, \frac{1}{2}D]$,

Type 4: $w_j \in [\frac{1}{2}W, W], h_j \in [\frac{1}{2}H, H], d_j \in [\frac{1}{2}D, D]$,

Type 5: $w_j \in [1, \frac{1}{2}W], h_j \in [1, \frac{1}{2}H], d_j \in [1, \frac{1}{2}D]$,

Item classes k ($k = 1, 2, 3, 4, 5$) are then generated by giving type k 60% probability and other types 10% probability each.

For item classes 6 to 8, the items are generated as follows:

Class 6: w_j, h_j, d_j uniformly distributed in $[1, 10]$,

Class 7: w_j, h_j, d_j uniformly distributed in$[1, 35]$,

Class 8: w_j, h_j, d_j uniformly distributed in $[1, 100]$, and

Class 9 consists of diffcult full bin solutions at optimality with an optimal solution of 3 bins. The items are generated by cutting the bins into smaller parts. Bin 1 and 2 are cut into $\lfloor \frac{n}{3} \rfloor$ items each, and Bin 3 is cut into $n - \lfloor \frac{n}{3} \rfloor$ items, where n is the number of items.

The Layered heuristic is set up so that each iteration considers 10 items. For an instance of 100 items, 10 iterations were needed. The results are summarized in Table 5.1. The columns "Average", "Min", and "Max" represent the average, minimum, and maximum number of bins required by each of the item classes, respectively. Each item class contains

10 instances. Given that there are 100 items, it is clear that each bin contains on average 4 to 5 items except for item class 6 and 7. However, in real life scenarios each pallet should contain about 50 to 100 items. Therefore we increased the volume of the bins by a factor of 10 and re-run the tests to see how the algorithm performs in a more realistic setting. The results are summarized in Table 5.2.

	10-item subsets		
	Average	Min	Max
Class 1	29.4	26	32
Class 2	28.3	25	31
Class 3	28.8	25	33
Class 4	59.2	53	66
Class 5	17.9	13	22
Class 6	1	1	1
Class 7	2	2	2
Class 8	22.4	17	32
Class 9	5.3	5	6

Table 5.1: Performance of the Layered Heuristic: 10-item subsets, 1000 seconds, bin dimension $100 \times 100 \times 100$.

After enlarging the bins by a factor of 10, the impact of going from 10-item subsets to 20-item subsets is compared in Table 5.2. The former setting needs 10 iterations to place all the items whereas the latter setting requires 5 iterations. By considering 20 items per iteration, we expect to find better solutions, which is clear from the Table 5.2, as on average 0.08 less bins are used. However, we should keep in mind that as the size of item subset doubles, the run-time required to find an optimal solution would increase. The computational time limit given to both settings are the same, which is 1000 seconds.

	10-item subsets			20-item subsets		
	Average	Min	Max	Average	Min	Max
Class 1	3.3	3	4	3	3	3
Class 2	3.1	3	4	3	3	3
Class 3	3	3	3	3	3	3
Class 4	5.1	5	6	5	4	6
Class 5	2	2	2	2	2	2
Class 6	1	1	1	1	1	1
Class 7	1	1	1	1	1	1
Class 8	2.3	2	3	2.2	2	3
Class 9	1.3	1	3	1.2	1	3
Overall	2.46			2.38		

Table 5.2: Performance of the Layered Heuristic: 10-items subsets V.S 20-item subsets, 1000 seconds, bin dimension $215 \times 215 \times 215$.

	10-item subsets			10-item subsets with ordering		
	Average	Min	Max	Average	Min	Max
Class 1	3.3	3	4	3.3	3	4
Class 2	3.1	3	4	3	3	3
Class 3	3	3	3	3	3	3
Class 4	5.1	5	6	4.8	4	5
Class 5	2	2	2	2	2	2
Class 6	1	1	1	1	1	1
Class 7	1	1	1	1	1	1
Class 8	2.3	2	3	2.2	2	3
Class 9	1.3	1	3	1.3	1	3
Overall	2.46			2.4		

Table 5.3: Performance of the Layered Heuristic: 10-item subsets V.S. 10-item subsets with ordering, 1000 seconds, bin dimension $215 \times 215 \times 215$.

Next we test the effect of ordering items. Intuitively, it is preferred to place larger items first so that smaller items can fill the spaces created between the larger items. Therefore the items are sorted in descending order of their volume. This is the same item-selection strategy as in Hifi et al. [12]. We gave each setting a run-time of 1000 seconds over and the result is summarized in Table 5.3. Ordering the items produced better solutions for classes 2, 4, and 8.

10-item subsets	500 seconds			1000 seconds			2000 seconds		
	Average	Min	Max	Average	Min	Max	Average	Min	Max
Class 1	3.1	3	4	3.3	3	4	3.1	3	4
Class 2	3.1	3	4	3	3	3	3	3	3
Class 3	3	3	3	3	3	3	3	3	3
Class 4	4.7	4	5	4.8	4	5	4.8	4	5
Class 5	2	2	2	2	2	2	2	2	2
Class 6	1	1	1	1	1	1	1	1	1
Class 7	1	1	1	1	1	1	1	1	1
Class 8	2.1	2	3	2.2	2	3	2.1	2	3
Class 9	1.3	1	2	1.3	1	3	1.2	1	2
Overall	2.37			2.4			2.36		

Table 5.4: Performance of the Layered Heuristic: 10-item subsets, comparison of running time of 500s, 1000s, 2000s, bin dimension $215 \times 215 \times 215$.

Different running-time settings are compared in Table 5.4. The items are sorted in descending order of their volumes. Using 10-item subsets, the obtained results are very interesting. First, the 500-second setting outperformed the 1000-second setting, and the 2000-second setting outperformed the 500-second setting by a small margin. This is due to the nature of the heuristic.

In Table 5.5, we test on 20-item subsets and see that the solution improves as running time increases.

20-item subsets	500 seconds			1000 seconds			2000 seconds		
	Average	Min	Max	Average	Min	Max	Average	Min	Max
Class 1	3.2	3	4	3	3	3	3	3	3
Class 2	3.2	3	4	3	3	3	3	3	3
Class 3	3.2	3	4	3	3	3	3	3	3
Class 4	4.9	4	5	5	4	6	4.9	4	5
Class 5	2	2	2	2	2	2	2	2	2
Class 6	1	1	1	1	1	1	1	1	1
Class 7	1	1	1	1	1	1	1	1	1
Class 8	2.4	2	3	2.2	2	3	2.1	2	3
Class 9	1.2	1	2	1.2	1	3	1.2	1	2
Overall	2.46			2.38			2.36		

Table 5.5: Performance of the Layered Heuristic: 20-item subsets, comparison of running time of 500s, 1000s, 2000s, bin dimension $215 \times 215 \times 215$.

5.2.1 Comparing the Layered Heuristic to Algorithm 864 by Martello et al. [15]

Martello *et al.* provide an implementation of their algorithm, Algorithm 864, available at Professor Pisinger's project webpage. In this section, we test the layered heuristic against Algorithm 864 for bins of dimension $100 \times 100 \times 100$ and $215 \times 215 \times 215$, given a computational time of 1000 seconds. The results are summarized in Table 5.6. Algorithm 864 outperformed the layered heuristic by an average of 0.58 bins.

	Layered Heuristic			Algorithm 864		
	Average	Min	Max	Average	Min	Max
Class 1	29.4	26	32	27.5	25	30
Class 2	28.3	25	31	26.4	24	29
Class 3	28.8	25	33	29	26	33
Class 4	59.2	53	66	59.2	52	66
Class 5	17.9	13	22	17.3	13	21
Class 6	1	1	1	1	1	1
Class 7	2	2	2	2.1	2	3
Class 8	22.4	17	32	19.9	18	28
Class 9	5.3	5	6	6.7	5	8
Overall	21.59			21.01		

Table 5.6: Layered Heuristic V.S. Algorithm 864, 1000 seconds, bin dimension $100 \times 100 \times 100$.

Next, we compare both algorithms using enlarged bins with dimension $215 \times 215 \times 215$. The results are summarized in Table 5.7. In this case, the layered heuristic does better and saves an average of 1.26 bins over Algorithm 864. From the two summary tables, it is clear that the layered heuristic is more efficient when the average number of items per bin is larger. In this section, we have showed that using the standard testing instances, the

Layered Heuristic compares well to algorithm 864.

	Layered Heuristic			Algorithm 864		
	Average	Min	Max	Average	Min	Max
Class 1	3	3	3	4.9	4	5
Class 2	3	3	3	4.7	4	5
Class 3	3	3	3	4.9	4	5
Class 4	5	4	6	6.5	6	7
Class 5	2	2	2	3.5	3	4
Class 6	1	1	1	1	1	1
Class 7	1	1	1	1	1	1
Class 8	2.2	2	3	3.9	3	4
Class 9	1.2	1	3	2.4	2	4
Overall	2.38			3.64		

Table 5.7: Layered Heuristic V.S. Algorithm 864, 1000 seconds, bin dimension $215 \times 215 \times 215$.

Chapter 6

Conclusion

In this thesis, the three dimensional bin packing problem is formulated, and solved. We start with an integer programming formulation for the Open Dimension Problem (ODP) where only one bin is assumed to hold all items, and then provide an enhancement to the model through the slicing approach.

Given the long computational times, the model is only able to pack small number of items. We use this idea in a heuristic approach where small subsets of items are solved sequentially. In the first iteration, we find the optimal placement of the first group of items. In the second iteration, the position of the items in the first subset is fixed and the second subset is placed. This process is repeated until all items are packed.

Finally, to use the solution of the ODP model to solve the SBSBPP, we introduce the Layered Heuristic, where we put very thin dividers of height 0.01 in the single bin at standard bin heights, so that the single bin will later be divided into several standardsized bins. This Layered Heuristic is compared to the approach of Martello *et al* using the standard test instance generator. It was found to perform well when the expected number of items per bin is large, which is the case for practical problems.

APPENDICES

Appendix A

Detailed Numerical Results

The numbered columns represent the index of each instance in a given class of items. The "Subtotal" column gives the total bins used for each item class. Each row gives the bins used to pack each item instance for a given class.

Layering	1	2	3	4	5	6	7	8	9	10	Subtotal
Class 1	28	31	32	29	32	26	28	30	27	31	294
Class 2	27	29	30	28	31	28	25	26	28	31	283
Class 3	25	30	33	29	31	26	26	28	29	31	288
Class 4	53	55	62	56	62	66	59	61	58	60	592
Class 5	16	20	22	18	21	13	14	18	19	18	179
Class 6	1	1	1	1	1	1	1	1	1	1	10
Class 7	2	2	2	2	2	2	2	2	2	2	20
Class 8	20	22	22	21	17	23	19	24	24	32	224
Class 9	6	5	5	5	6	6	5	5	5	5	53
										Total	1943
Pisinger	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Subtotal**
Class 1	26	29	30	27	28	25	25	29	26	30	275
Class 2	25	26	27	25	29	28	24	24	27	29	264
Class 3	27	32	33	30	30	26	26	27	28	31	290
Class 4	52	55	62	56	62	66	59	61	58	61	592
Class 5	16	18	21	18	21	13	14	17	15	20	173
Class 6	1	1	1	1	1	1	1	1	1	1	10
Class 7	2	2	2	3	2	2	2	2	2	2	21
Class 8	18	18	19	19	18	20	18	20	21	28	199
Class 9	7	6	7	7	6	7	5	8	7	7	67
										Total	1891
Lower Bound	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Subtotal**
Type 1	22	28	27	26	25	24	23	26	23	27	251
Type 2	22	24	26	24	26	26	21	22	23	27	241
Type 3	23	24	27	24	27	25	22	24	23	28	247
Type 4	51	51	61	56	61	64	57	60	57	58	576
Type 5	12	15	15	13	15	10	10	13	12	14	129
Type 6	1	1	1	1	1	1	1	1	1	1	10
Type 7	1	1	1	1	1	1	1	1	1	1	10
Type 8	18	17	17	16	13	18	14	17	18	28	176
Type 9	3	3	3	3	3	3	3	3	3	3	30
										Total	1670

Table A.1: Layered Heuristic V.S. Algorithm 864, 1000 seconds, bin dimension $100 \times 100 \times 100$, detailed results.

Stacking	1	2	3	4	5	6	7	8	9	10	Subtotal
Class 1	3	4	4	3	3	3	3	3	3	4	33
Class 2	3	3	3	3	3	3	3	3	3	4	31
Class 3	3	3	3	3	3	3	3	3	3	3	30
Class 4	5	5	6	5	5	5	5	5	5	5	51
Class 5	2	2	2	2	2	2	2	2	2	2	20
Class 6	1	1	1	1	1	1	1	1	1	1	10
Class 7	1	1	1	1	1	1	1	1	1	1	10
Class 8	2	2	3	2	2	2	3	2	2	3	23
Class 9	3	1	1	2	1	1	1	1	1	1	13
										Total	221
Pisinger	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Subtotal**
Class 1	5	5	5	5	5	5	4	5	5	5	49
Class 2	4	5	5	5	5	5	5	4	5	4	47
Class 3	5	5	5	5	5	5	4	5	5	5	49
Class 4	6	6	7	6	6	7	6	7	7	7	65
Class 5	4	3	3	4	4	3	3	3	4	4	35
Class 6	1	1	1	1	1	1	1	1	1	1	10
Class 7	1	1	1	1	1	1	1	1	1	1	10
Class 8	4	4	4	4	3	4	4	4	4	4	39
Class 9	4	3	2	2	2	3	2	2	2	2	24
										Total	328
Lower Bound	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Subtotal**
Class 1	2	3	3	3	3	2	2	2	2	3	25
Class 2	2	2	2	2	2	2	2	2	2	3	21
Class 3	2	2	3	2	3	2	2	2	2	3	23
Class 4	3	3	4	4	4	4	4	4	4	4	38
Class 5	2	2	2	2	2	1	1	2	2	2	18
Class 6	1	1	1	1	1	1	1	1	1	1	10
Class 7	1	1	1	1	1	1	1	1	1	1	10
Class 8	2	2	2	2	2	2	2	2	2	2	20
Class 9	3	1	1	1	1	1	1	1	1	1	12
										Total	177

Table A.2: Layered Heuristic V.S. Algorithm 864, 1000 seconds, bin dimension $215 \times 215 \times 215$, detailed results.

10-item subset	1	2	3	4	5	6	7	8	9	10	Subtotal
Class 1	3	4	4	3	3	3	3	3	3	4	33
Class 2	3	3	3	3	3	3	3	3	3	4	31
Class 3	3	3	3	3	3	3	3	3	3	3	30
Class 4	5	5	6	5	5	5	5	5	5	5	51
Class 5	2	2	2	2	2	2	2	2	2	2	20
Class 6	1	1	1	1	1	1	1	1	1	1	10
Class 7	1	1	1	1	1	1	1	1	1	1	10
Class 8	2	2	3	2	2	2	3	2	2	3	23
Class 9	3	1	1	2	1	1	1	1	1	1	13
										Total	221
20-item subset	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Subtotal**
Class 1	3	3	3	3	3	3	3	3	3	3	30
Class 2	3	3	3	3	3	3	3	3	3	3	30
Class 3	3	3	3	3	3	3	3	3	3	3	30
Class 4	4	5	5	5	5	6	5	5	5	5	50
Class 5	2	2	2	2	2	2	2	2	2	2	20
Class 6	1	1	1	1	1	1	1	1	1	1	10
Class 7	1	1	1	1	1	1	1	1	1	1	10
Class 8	2	2	3	2	2	2	2	2	2	3	22
Class 9	3	1	1	1	1	1	1	1	1	1	12
										Total	214

Table A.3: Performance of the Layered Heuristic: 10-item subset V.S 20-item subset, bin dimension $215 \times 215 \times 215$, detailed results.

10 per group	1	2	3	4	5	6	7	8	9	10	Subtotal
Class 1	3	4	4	3	3	3	3	3	3	4	33
Class 2	3	3	3	3	3	3	3	3	3	4	31
Class 3	3	3	3	3	3	3	3	3	3	3	30
Class 4	5	5	6	5	5	5	5	5	5	5	51
Class 5	2	2	2	2	2	2	2	2	2	2	20
Class 6	1	1	1	1	1	1	1	1	1	1	10
Class 7	1	1	1	1	1	1	1	1	1	1	10
Class 8	2	2	3	2	2	2	3	2	2	3	23
Class 9	3	1	1	2	1	1	1	1	1	1	13
										Total	221
10 per Group with Ordering	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Subtotal**
Class 1	3	3	4	3	4	3	3	3	3	4	33
Class 2	3	3	3	3	3	3	3	3	3	3	30
Class 3	3	3	3	3	3	3	3	3	3	3	30
Class 4	4	4	5	5	5	5	5	5	5	5	48
Class 5	2	2	2	2	2	2	2	2	2	2	20
Class 6	1	1	1	1	1	1	1	1	1	1	10
Class 7	1	1	1	1	1	1	1	1	1	1	10
Class 8	2	2	3	2	2	2	2	2	2	3	22
Class 9	3	1	1	1	1	1	1	1	1	2	13
										Total	216

Table A.4: Performance of the Layered Heuristic: 10-item subset V.S 10-item with ordering, bin dimension $215 \times 215 \times 215$, detailed results.

References

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

[2] EE Bischoff. Three-dimensional packing of items with limited load bearing strength. *European Journal of Operational Research*, 168(3):952–966, 2006.

[3] EE. Bischoff and MSW Ratcliff. Issues in the development of approaches to container loading. *Omega*, 23(4):377–390, 1995.

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

[5] CS Chen, S. Lee, and QS Shen. An analytical model for the container loading problem. *European Journal of Operational Research*, 80(1):68–76, 1995.

[6] T. Crainic, G. Perboli, and R. Tadei. Extreme point-based heuristics for three-dimensional bin packing. *Informs Journal on computing*, 20(3):368–384, 2008.

[7] TG. Crainic, G. Perboli, and R. Tadei. Ts 2 pack: A two-level tabu search for the

three-dimensional bin packing problem. *European Journal of Operational Research*, 195(3):744–760, 2009.

[8] E. Den Boef, J. Korst, S. Martello, D. Pisinger, and D. Vigo. Erratum to the three-dimensional bin packing problem: robot-packable and orthogonal variants of packing problems. *Operations Research*, 53(4):735–736, 2005.

[9] L. Faina. A global optimization algorithm for the three-dimensional packing problem. *European Journal of Operational Research*, 126(2):340–354, 2000.

[10] S. Fekete and J. Schepers. A new exact algorithm for general orthogonal d-dimensional knapsack problems. In *AlgorithmsESA'97*, pages 144–156. Springer, 1997.

[11] S. Fekete and J. Schepers. A combinatorial characterization of higher-dimensional orthogonal packing. *Mathematics of Operations Research*, 29(2):353–368, 2004.

[12] M. Hifi, S. Negre, and L. Wu. Hybrid greedy heuristics based on linear programming for the three-dimensional single bin-size bin packing problem. *International Transactions in Operational Research*, 21(1):59–79, 2014.

[13] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241–252, 2002.

[14] S. Martello, D. Pisinger, and D. Vigo. The three-dimensional bin packing problem. *Operations Research*, 48(2):256–267, 2000.

[15] S. Martello, D. Pisinger, D. Vigo, E. Boef, and J. Korst. Algorithm 864: General

and robot-packable variants of the three-dimensional bin packing problem. *ACM Transactions on Mathematical Software (TOMS)*, 33(1):7, 2007.

[16] RD. Meller, V. Narayanan, and PH. Vance. Optimal facility layout design. *Operations Research Letters*, 23(3):117–127, 1998.

[17] B. Mohanty, K. Mathur, and N. Ivancic. Value considerations in three-dimensional packinga heuristic procedure using the fractional knapsack problem. *European Journal of Operational Research*, 74(1):143–151, 1994.

[18] C. Pimpawat and N. Chaiyaratana. Using a co-operative co-evolutionary genetic algorithm to solve a three-dimensional container loading problem. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 2, pages 1197–1204. IEEE, 2001.

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

[20] Y. Wu, W. Li, M. Goh, and R. de Souza. Three-dimensional bin packing problem with variable bin height. *European Journal of Operational Research*, 202(2):347–355, 2010.

[21] W. Zhu, W. Huang, and A. Lim. A prototype column generation strategy for the multiple container loading problem. *European Journal of Operational Research*, 223 (1):27–39, 2012.

[22] W. Zhu, Z. Zhang, W. Oon, and A. Lim. Space defragmentation for packing problems. *European Journal of Operational Research*, 222(3):452–463, 2012.