Models and Solution Methods for the Pallet Loading Problem

by

Burak Can Yildiz

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Management Sciences

Waterloo, Ontario, Canada, 2018

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	José Fernando Oliveira
	Professor, Dept. of Industrial Engineering and Management
	University of Porto
Supervisor(s):	Samir Elhedhli
	Professor, Dept. of Management Sciences
	Fatma Gzara
	Assoc. Professor, Dept. of Management Sciences
Internal Member:	Sibel Alumur Alev
	Asst. Professor, Dept. of Management Sciences
Internal Member:	Houra Mahmoudzadeh
	Asst. Professor, Dept. of Management Sciences
Internal-External Member:	Ramadan El-Shatshat
	Lecturer, Dept. of Electrical and Computer Engineering

Abstract

The three-dimensional bin packing problem (3DBPP) seeks to find the minimum number of bins to pack a finite number of rectangular boxes. It has a wide array of applications, ranging from airline cargo transportation to warehousing. Its practical extension, the distributor's pallet loading problem (DPLP), requires the pallets to be stable, packable, and adhering to several industry requirements such as packing sequences and weight limits.

Despite being studied extensively in the optimization literature, the 3DBPP is still one of the most difficult problems to solve. Currently, medium to large size instances are only solved heuristically and remain out of reach of exact methods. This also applies to the DPLP, as the addition of practical constraints further complicates the proposed models. A recent survey identified the scarcity of exact solution methods that are capable of handling practical versions of the problem and the lack of a realistic benchmark data set as major research gaps.

In this thesis, firstly, we propose a novel formulation and an exact solution approach based on column generation for the 3DBPP, where the pricing subproblem is a two-dimensional layer generation problem. Layers are highly desirable in practical packings as they are easily packable and can accommodate important practical constraints such as item support, family groupings, isle friendliness, and load bearing. Being key to the success of the column generation approach, the pricing subproblem is solved optimally as well as heuristically, and is enhanced using item grouping, item replacement, layer reorganization, and layer spacing. We also embed the column generation approach within a branch-and-price framework. We conduct extensive computational experiments and compare against existing approaches. The proposed approach outperforms the best performing algorithm in the literature in most instances and succeeds to solve practical size instances in very reasonable computational times.

Secondly, we extend the column generation scheme to incorporate practical constraints set by the warehousing industry. We introduce a nonlinear layer spacing model to improve the stability of the planned pallets, which we then reformulate as an SOCP. In order to calculate the weight distribution within pallets, we introduce a new graph representation for placed items. Finally, we propose construction and improvement heuristics to tackle each practical constraint, such as vertical support, different item shapes, planogram sequencing, load bearing, and weight limits. We conduct extensive computational experiments to demonstrate the good performance of the proposed methodology, and provide results for future benchmarking. To the best of our knowledge, this is the first approach to fully solve the DPLP. Computational experiments show that the proposed approach succeeds in solving industry size instances in record computational times and achieves high quality solutions that account for all practical constraints.

Finally, we propose realistic benchmark instances by designing and training an instance generator using industry data. We apply clustering and curve fitting techniques to 342 industry instances with 166,406 items to obtain the distributions for item volumes, dimensions, and frequencies. We separate the instances into several classes and categories using k-clustering and generate multiple instances with different sizes. We, then, extend the generator to incorporate practical features such as weight, load capacity, shape, planogram sequencing, and reduced edge support.

Acknowledgements

First and foremost, I would like to express my utmost gratitude to my supervisors, Prof. Samir Elhedhli and Prof. Fatma Gzara, for their invaluable guidance and support throughout my degree. They helped me grow academically and professionally not just with the education and ideas that they provided, but also with their financial support, bottomless understanding, constant encouragement, and with their admirable character and personality. They were absolutely the best supervisors I could have worked with and I could not have asked for anything more.

Secondly, I would like to thank the members of my thesis committee: Prof. Sibel Alumur, Prof. Houra Mahmoudzadeh, Ramadan El-Shathat, and Prof. José Fernando Oliveira. Their suggestions and ideas helped improve this work immensely. In addition, I would like to extend my gratitude to Prof. Hossein Abouee Mehrizi and Prof. Leonardo Simon for their valuable input during my comprehensive examination.

I am also extremely lucky for being a member of the Waterloo Analytics and Optimization Lab (WanOpt), again thanks for Samir Elhedhli and Fatma Gzara. The time I spent among my colleagues at the lab, discussing new ideas and helping each other, proved invaluable in shaping who I am today and my academic work. For this reason, I am honoured to have worked with Da Lu, Ahmed Saif, Ugur Yildiz, Cynthia Waltho, Paulo de Carvalho, and Daniel Ulch. I was also able to work on two different industry projects during my time at the lab, second of which motivated the work in this thesis. Therefore, I am humbled and grateful for the opportunity that I was provided.

I would like to extend my gratitude towards the Management Sciences department and the University of Waterloo as a whole, for the educational, financial, and administrative support that they granted.

Last but not least, I would like to thank my family for their infinite support and love. My parents, Zeynep and Kadir Yildiz, raised me and made me into who I am today, and there are absolutely no words to do justice to the effort they spent. To them, I will eternally be indebted.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

1DBPP one-dimensional bin packing problem 2, 12

2DBPP two-dimensional bin packing problem 2, 11, 14, 19

3DBPP three-dimensional bin packing problem xiii, 1–5, 7–11, 13, 14, 18, 25, 28, 29, 31, 42, 44, 46, 56, 72, 94, 95

CLP container loading problem 7, 15

DPLP distributor's pallet loading problem xii, 3–5, 7, 18, 72, 74, 90, 94, 115–118

GRASP greedy randomized adaptive search procedure 13

LCGA layer-based column generation algorithm xii, 46–48, 50, 51, 55, 61, 62, 73, 90, 94, 104–107

ODPP open dimension packing problem 2, 21, 24, 31

RPM relative positioning model 14

SOCP second-order cone programing 6, 79, 91, 93, 94

SPP strip packing problem 19

List of Symbols

Sets		
$\mathcal{I}$	=	Set of items
$\mathcal{CI}$	=	Set of superitems and items
$\mathcal{IG}$	=	Set of item groups
$\mathcal{L}$	=	Set of layers
$\mathcal{B}$	=	Set of bins
Parameters		
W	=	Width of a layer/bin
D	=	Depth of a layer/bin
H	=	Height of a bin
w_i	=	Width of item $i \in \mathcal{I}$
d_i	=	Depth of item $i \in \mathcal{I}$
h_i	=	Height of item $i \in \mathcal{I}$
v_i	=	Volume of item $i \in \mathcal{I}$
f_{si}	=	1 if superitem $s \in \mathcal{CI}$ includes item $i \in \mathcal{I}$, 0 otherwise
Variables		
z_{il}	=	1 if item $i \in \mathcal{I}$ is placed in layer $l \in \mathcal{L}$, 0 otherwise
x_{ij}	=	1 if item $i \in \mathcal{I}$ precedes item $j \in \mathcal{I}$ along the width dimension, 0 otherwise
y_{ij}	=	1 if item $i \in \mathcal{I}$ precedes item $j \in \mathcal{I}$ along the depth dimension, 0 otherwise
α_l	=	1 if layer $l \in \mathcal{L}$ is selected, 0 otherwise
u_{lb}	=	1 if layer $l \in \mathcal{L}$ is placed in bin $b \in \mathcal{B}$, 0 otherwise
t_b	=	1 if bin $b \in \mathcal{B}$ is used, 0 otherwise
c_i^1	=	Bottom-left coordinate of item $i \in \mathcal{I}$ in the width dimension
c_i^2	=	Bottom-left coordinate of item $i \in \mathcal{I}$ in the depth dimension
o_l	=	Height of layer $l \in \mathcal{L}$
a^1	=	The minimum distance between every pair of items in a layer along the width dimension
a^2	=	The minimum distance between every pair of items in a layer along the depth dimension
s_{ij1}	=	The amount of width overlap between items i, j in consecutive layers
s_{ij2}	=	The amount of depth overlap between items i, j in consecutive layers

Table 1: Sets, parameters, and variables for the mathematical models.

Chapter 1

Introduction

The 3DBPP seeks to pack small rectangular boxes into the minimum possible number of larger boxes, called bins. The problem is frequently encountered in logistics and supply chain operations. Pallets and containers, being the most common platform for shipping, are essentially three-dimensional packings with side constraints. Pallet loading is a major component of any warehouse operation where thousands of items are packed in industry-size pallets on a daily basis. The current research is inspired by an industry project for a global warehousing and logistics company. Building optimized three-dimensional pallets is a major bottleneck in its highly-automated warehouse sorting, retrieval, and palletizing systems. Its automated warehouse has the capability to build 50 to 100-item pallets in 2 minutes. Being done dynamically, palletization must be optimized both for the number of bins and for the quality of the packings in less than 2 minutes.

1.1 The Three Dimensional Bin Packing Problem

Three dimensional bin packing problems come in three variants: the knapsack loading problem, the ODPP, and the 3DBPP (Martello et al.,2000). The knapsack loading problem focuses on packing a subset of rectangular boxes into one or more containers such that the total value of the packed items is maximized (Gehring et al., 1990, Pisinger, 2002, Wang et al., 2008). The ODPP aims to pack a set of rectangular boxes into a container with limited width and depth, and unlimited height, such that the height of the top item is minimized (George and Robinson, 1980, Bischoff and Marriott, 1990, Bortfeldt and Mack, 2007). Finally, the 3DBPP considers packing a set of items into a minimum number of fixed-sized containers (bins). A typology of packing problems is provided by Wäscher et al. (2007).

The 3DBPP is NP-Hard (Martello et al., 2000). It is easy to see that a three-dimensional bin packing problem reduces to a one-dimensional bin packing problem when two dimensions (e.g., height and depth) are equal to the bin dimensions for all items. This makes the 1DBPP a special case of the 3DBPP. As 1DBPP is already known to be NP-Hard, the 3DBPP is NP-Hard as well (Martello et al., 2000).

Although 3DBPP is a direct extension of the 1DBPP and the 2DBPP, the third dimension introduces substantial complications such as vertical support, bin stability, and load bearing (Bortfeldt and Wäscher, 2013). The problem has been the subject of extensive research for at least two decades with a wealth of solution approaches, mostly based on placement heuristics and metaheuristics, but with timid progress on exact methods. In fact, the proposed exact methods cannot handle industry-size instances, and the heuristics do not consider most of the practical constraints. Moreover, a recent survey by Zhao et al. (2016), focusing on solution methodologies and their comparative performance on benchmark data sets, identified three main research gaps: the lack of approaches for multiple container size problems, the inadequate handling of real-world practical constraints such as bin stability and vertical support, and the absence of realistic benchmark data sets.

1.2 The Distributor's Pallet Loading Problem

Warehousing and delivery costs are two dominating factors in logistics. Based on the 2013 State of Logistics Report by the Council of Supply Chain Management Professionals (CSCMP), annual transportation and warehousing costs were \$836 billion and \$434 billion in the U.S., respectively (Kearney, 2014). Together, they represent 95.4% of total logistics costs. Furthermore, 77.4% of the transportation costs were trucking costs, which are directly correlated with the number of trucks used to carry goods. Therefore, minimizing the number of trucks used is critical in reducing the operating costs in a supply chain.

Goods are generally transported using pallets and containers, which are planned and packed in automated warehouses and distribution centers. Thus, the 3DBPP and its practical counterpart, the DPLP, are of great importance, as they minimize the number of pallets, and therefore trucks used for transportation. The DPLP deals mostly with heterogenous items and is sometimes referred to as the mixed-case pallet optimization problem. It is closely related to the Container Loading Problem. The Manufacturer's Pallet Loading Problem, on the other hand, considers homogenous items. A modern automated warehouse is expected to pack thousands of items into hundreds of industry-sized pallets every day. Owing to the dynamic and fast paced nature of the packaging operation, a solution approach to the DPLP has to plan a pallet every 2 minutes. Additionally, these pallets have to be stable and light enough for transportation.

Being an extension to the 3DBPP, the DPLP is further complicated by the addition of practical constraints such as vertical support, load bearing, planogram sequencing, and bin weight limits. Current methodologies that account for these practical constraints cannot solve even the smallest of industry instances in reasonable times. Furthermore, none of the approaches devised to tackle the entirety of the DPLP is fast enough to be used in industry. This research is highly motivated by the lack of fast and scalable solution methodologies for the DPLP.

1.3 Contribution and Outline of the Thesis

In this thesis, we study the 3DBPP and its practical extension, the DPLP. Motivated by the need to provide an efficient solution methodology that can handle industry-size problems, we provide a novel formulation based on layers and exploit it to provide a branch-and-price solution framework. We extend the approach to handle practical constraint such as bin stability, and provide a benchmark data set that is trained on industrial data. The use of layers has numerous advantages related to stability and support. It also enables the use of a column generation approach where the subproblem is a two-dimensional bin packing problem. Finally, we tackle the entirety of the DPLP, incorporating the full range of the practical constraints in a scalable and fast solution methodology.

The thesis is organized as follows. In Chapter 2, we review the literature on the 3DBPP, focusing on lower bounds, exact methods, and heuristic approaches. We, then, review the literature that considers practical constraints, namely bin stability, item orientations, vertical support, load bearing, weight distribution and limit, and multiple container shapes and sizes.

In Chapter 3, we discuss the main contributions of the thesis: a novel formulation and a column generation framework based on layers. We explicitly model layers and devise branch-and-price and column generation methodologies. Being key to the success of the approach, we focus on the layering subproblem, solving it both exactly to achieve tight lower bounds and heuristically for quick warm-starting. To construct feasible solutions, we propose strategies for layer selection and bin construction as well as a simple placement heuristic to pack remaining items. Second, we address one of the most important practical constraints which is item support and highlight its relationship with the proposed layering approach and the construction heuristic we adopt. Increasing the density of layers implicitly provides bins with high stability and vertical support (Zhao et al., 2016). Therefore, we maximize layer density through item groupings that we call superitems and prioritize denser layers when bins are constructed. We also propose an optimization model to evenly distribute items in layers for better spacing and increased support. To the best of our knowledge, this is the first work to propose a rigorous layer-based column generation approach for 3DBPP that is capable of handling industry-size instances, addresses practical constraints, and outperforms previous approaches. The results show a clear superiority in terms of optimal number of bins used and solution times as well as bin stability and vertical support.

In Chapter 4, we analyze a large set of industrial data and propose a framework to generate realistic instances for the 3DBPP and the DPLP that we hope will be used as a basis for future benchmarking. The analysis first identifies item types based on their dimensional proportions, volumes, and frequency of occurrence, and uses these features to construct statistical distributions and parameters that are utilized to generate instances that resemble real-life palletization problems. It, then, uses practical item features such as weight, load bearing capacity, shape, reduced support, and sequence number in a more detailed analysis to extend the instance generator to provide a data set for the DPLP.

We extend the proposed methodology for the 3DBPP in Chapter 5 to explicitly account for all practical constraints set by the industry. We apply both exact and heuristic and post-processing to generated layers to improve the chances of feasible placements, and use an extreme point placement method to deal with items that cannot be placed in layers. We propose a SOCP approach to maximize spacing within the layer under consideration while ensuring enough overlap with the layer beneath it to satisfy support. In addition, we use a graph based breadth-first search algorithm to verify the load bearing feasibility of a packing in a fast manner. We propose algorithms for incorporating items with different shapes, and several layer sorting rules to improve the quality of the resulting pallets. We, also, provide the results of the extensive computational tests that we conducted to underline the effectiveness of the proposed solution methodology. On average, we are able to provide compact and stable solutions well within the industry time standard of two minutes. Moreover, these solutions satisfy all known practical constraints. Finally, we provide concluding remarks in Chapter 6 and discuss possible future research directions.

Chapter 2

Literature Review

Although the 3DBPP may be modeled and solved as a MIP, exact solution methodologies are scarce. The literature offers a variety of heuristics instead. Furthermore, studies that consider even a subset of the practical constraints are even fewer. This is largely due to the increasing complexity of the problem with each additional practical constraint. To the best of our knowledge, there is no methodology in the literature that considers the entirety of the DPLP that can also solve realistically large instances. In what follows, we review the best performing heuristic and exact methods, bounding schemes, and commonly used benchmark instance sets for the 3DBPP. We also review the limited number of publications on practical constraints. For recent reviews on the 3DBPP and the CLP, we refer to the work of Zhao et al. (2016) and Bortfeldt and Wäscher (2013).

2.1 Lower Bounds

Bounding schemes are important to accurately test the performance of a proposed solution method. Stricter lower bounds would allow a more precise look into how close a solution is to the optimal. There are three main lower bounds proposed in the literature for the 3DBPP.

The first of these bounds is called the continuous lower bound (L_0) (Martello et al., 2000). This bound is applied to all bin packing problems, independent from how many dimensions they have. L_0 is basically defined as the number of bins required to pack all items, if there is no empty space in any of the bins. For the three-dimensional case, it is calculated using the following equation:

$$L_0 = \frac{\sum\limits_{i \in \mathcal{I}} w_i d_i h_i}{WDH}$$

when the set of items are defined using $\mathcal{I}$, item dimensions are shown using w_i, d_i, and h_i ($\forall i \in \mathcal{I}$), and bin dimensions are given with W, D, and H for width, depth, and height, respectively.

Martello et al. (2000) argue that this bound would perform well if the item sizes are small with regards to the bin size. However, they continue that if there are large items present (e.g., $w_i > W/2$), L_0 would not provide a strict enough bound. They proved that the worst case performance of this bound is $\frac{1}{8}$. This means that it may be as low as one eighth of the optimal value. In light of this, they provide two more lower bounds for the 3DBPP, referred to as L_1 and L_2.

L_1 is obtained using a reduction to the one-dimensional case. Let

$$\mathcal{I}^{WH} = \left\{ i \in \mathcal{I} : w_i > \frac{W}{2} \text{ and } h_i > \frac{H}{2} \right\},$$

and define

$$\begin{aligned} L_1^{WH} &= \left| \left\{ i \in \mathcal{I}^{WH} : d_i > \frac{D}{2} \right\} \right| + \\ &\max_{1 \le p \le D/2} \left\{ \left\lceil \frac{\sum_{i \in \mathcal{I}_s(p)} d_i - \left(|\mathcal{I}_l(p)| \, D - \sum_{i \in \mathcal{I}_l(p)} d_i \right)}{D} \right\rceil, \left\lceil \frac{|\mathcal{I}_s(p)| - \sum_{i \in \mathcal{I}_l(p)} \left\lfloor \frac{D - d_i}{p} \right\rfloor}{\left\lfloor \frac{D}{p} \right\rfloor} \right\rceil \right\}, \\ \mathcal{I}_l(p) &= \left\{ i \in \mathcal{I}^{WH} : D - p \ge d_i > \frac{D}{2} \right\}, \\ \mathcal{I}_s(p) &= \left\{ i \in \mathcal{I}^{WH} : \frac{D}{2} \ge d_i \ge p \right\}. \end{aligned}$$

L_1^{WH} is obtained by considering the boxes that have more than half of the bin dimensions in the width and height dimensions. Such items can only be packed into a bin one behind another, this reduces the problem into a single dimension. Martello et al. (2000) proved that L_1^{WH} is a valid lower bound for the 3DBPP. Moreover, L_1^{WD} and L_1^{DH}, which are calculated by interchanging different dimensions, are also valid lower bounds. L_1 is calculated using the following equation:

$$L_1 = \max \left\{ L_1^{WH}, L_1^{WD}, L_1^{DH} \right\}.$$

They provided another lower bound by taking three item dimensions into account.

Using any pair of integers (p,q) such that $1 \leq p \leq W/2$ and $1 \leq q \leq H/2$, define

$$\begin{aligned}
\mathcal{K}_v(p,q) &= \{i \in \mathcal{I} : w_i > W - p \text{ and } h_i > H - q\}, \\
\mathcal{K}_l(p,q) &= \left\{i \in \mathcal{I} \setminus \mathcal{K}_v(p,q) : w_i > \frac{W}{2} \text{ and } h_i > \frac{H}{2}\right\}, \\
\mathcal{K}_s(p,q) &= \{i \in \mathcal{I} \setminus (\mathcal{K}_v(p,q) \cup \mathcal{K}_l(p,q)) : w_i \geq p \text{ and } h_i \geq q\},
\end{aligned}$$

and let

$$\begin{aligned}
L_2^{WH}(p,q) = & L_1^{WH} + \\
& \max\left\{0, \left\lceil \frac{\sum_{i \in \mathcal{K}_l(p,q) \cup \mathcal{K}_s(p,q)} v_i - \left(DL_1^{WH} - \sum_{i \in \mathcal{K}_v(p,q)} d_i\right) WH}{B} \right\rceil \right\},
\end{aligned}$$

where v_i is the volume of item $i \in \mathcal{I}$ and B is the volume of a bin, and

$$L_2^{WH} = \max_{1 \leq p \leq W/2; 1 \leq q \leq H/2} \left\{L_2^{WH}(p,q)\right\}.$$

L_2 starts from L_1 and takes the remainder of the boxes into account. They proved that a valid lower bound for the 3DBPP is given by

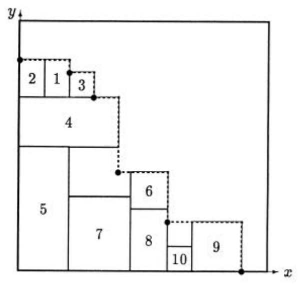
$$L_2 = \max\left\{L_2^{WH}, L_2^{WD}, L_2^{DH}\right\},$$

where L_2^{WD} and L_2^{DH} are calculated by interchanging different dimensions. Lastly, they proved that L_2 dominates both L_0 and L_1, and can be calculated in $O(n^2)$ time.

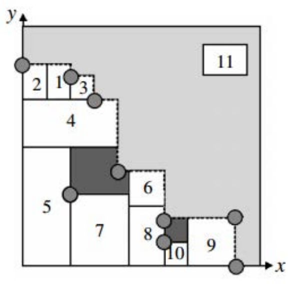
2.2 Heuristic Methods

The 3DBPP literature includes several well-performing heuristic approaches that are capable of solving larger and more realistic instances compared to exact methodologies. They are generally of two types: placement point methods and metaheuristics. Baker et al. (1980) were the first to introduce a placement point method for the 2DBPP, called bottom-left placement. This method works by placing a sorted list of items into a bin such that their bottom-left corner coincides with the lowest possible position. Martello et al. (2000) extended this methodology using corner points. A corner point is the result of the intersection between the three planes formed by the right, back, and top sides of items already placed in a bin. The bin is divided in two parts using a staircase that separates already placed items from the rest of the available space. They also introduced benchmark instances that were used in most subsequent work. Crainic et al. (2008) extended this approach by introducing the concept of extreme points. The latter result from the projections of the right, back, and top of items already placed in a bin. This method does not use an envelope to separate placed items from the rest of the bin as in the corner point approach. Using projections increases the number of candidate points for placement. Figures 2.1a and 2.1b show a 2D representation of the corner point and the extreme point methods, respectively.

Based on the extreme point method, Crainic et al. (2008) provided first-fit decreasing and best-fit decreasing heuristics, both of which use item sorting rules based on height-volume, volume-height, etc. After the items are sorted, the first-fit decreasing heuristic places the next item in the first open bin with a feasible extreme point placement. If there are no feasible placements, a new bin is opened. The best-fit decreasing heuristic on the other hand, evaluates extreme point item pairs based on a merit function and places them at the best point available. Zhu and Lim (2012) proposed a greedy look-ahead tree



(a) Corner point.



(b) Extreme point.

Figure 2.1: 2D representation of placement methods.

search algorithm. It starts by generating blocks of items and places them at the corner of the bin. The residual space created provides new corners that are, in turn, used to place new blocks. At each node of the search tree, they select a limited number of best placements and create child nodes by placing a new block at each. Zhu et al. (2012) presented a construction heuristic based on the extreme point and space defragmentation methods. Items are placed at extreme points and space is consolidated by pushing items out to the edges. They further improve the solution by moving items from a bin to another using space defragmentation. Faroe et al. (2003) proposed a Guided Local Search heuristic where the neighborhood is defined by either moving an item in any orthogonal direction or moving it to the same location in another bin while allowing overlapping. To find a feasible placement of items for a bin, they minimize the overlap between items, and guide the solution by penalizing consecutive overlaps.

Two Tabu Search heuristics were proposed for the problem. The first is due to Lodi et al. (2002). It starts by packing one item per bin. The neighborhood is defined by picking one bin and placing all its items into other bins. Items are placed in layers in two steps. A height-first and an area-first strategies are used in steps one and two, respectively. At the end of each step, a 1DBPP is solved to pack the layers into the bins. The second Tabu

Search heuristic is TS^2PACK due to Crainic et al. (2009), which also has two main steps. The first step determines the set of items to be packed into each bin, while the second step determines the position of each item in the bins. They use the extreme point based first-fit descending heuristic to find an initial solution. In the first step, the neighborhood is constructed by either swapping items from different bins or moving an item to another bin, where the bin dimensions are relaxed. In the second step, they find a feasible packing in a bin by swapping the relative positions of items.

Parreño et al. (2010) presented a hybrid heuristic that combines variable neighborhood descent and the GRASP. It starts by constructing an initial solution using the maximal space algorithm, which was for the container loading problem, based on GRASP. It then uses four improvement procedures (neighborhoods) in a variable neighborhood descent scheme by testing a move in each neighborhood. Wu et al. (2010) presented a genetic algorithm that is based on the relative positioning MIP. The chromosomes represent the order of items to be packed and their orientations, which are initially assigned by fixing an item sequence and randomly determining orientations. They use single point crossover to keep the order of the items relatively stable, a sequential and a random repair scheme to fix the chromosomes, and two different mutation schemes.

Zhu et al. (2012) proposed a column generation approach where each column is a certain arrangement of items in a single bin. The columns are generated by solving a single container loading subproblem. Since the subproblems are difficult, they solve a single container knapsack problem to generate approximate columns.

Out of all these heuristic methodologies, Space Defragmentation of Zhu et al. (2012) is found to perform the best with practical computational time limitations. Table 3.3 in Section 3.3 summarizes the performance of the most competitive heuristics for 3DBPP.

2.3 Exact Methods

Due to the complexity of the 3DBPP, the literature offers few exact solution methods. Chen et al. (1995) provided the first MIP formulation for 3DBPP based on the relative positions of items, which we refer to as the RPM in the rest of this paper. Their work is later extended by Wu et al. (2010) to allow for item orientations and by Junqueira et al. (2012) and Paquay et al. (2016) to accommodate practical constraints. Hifi et al. (2010) provided several lower bounds to RPM to decrease the solution time. They are based on the LP relaxation and valid inequalities. Junqueira et al. (2012) accounted for cargo stability and load bearing. They provided a framework to generate groups of items, which we expand on in this paper. Paquay et al. (2016) extended RPM by accounting for load bearing, item orientations, container shapes, weight distribution, and stability. Both Junqueira et al. (2012) and Paquay et al. (2016) solve their models using a commercial solver.

Martello et al. (2000) presented an enumerative two step tree search algorithm based on the corner-point placement method. Their work is extended by Martello et al. (2007) who provided an improved solution algorithm called "Algorithm 864". This algorithm determines which items to be packed into which bin at each node of the tree, ignoring placement. The feasibility of the formed bins is verified using constraint programming. A similar two-step approach was provided by Fekete et al. (2007) for higher dimensional bin packing problems (e.g., 2DBPP, 3DBPP). The main difference is in the way the feasibility of a bin is tested. They use an enumerative solution approach based on isomorphic packing classes.

A comparison of the aforementioned methods is provided in Table 2.1. Note that Junqueira et al. (2012), Paquay et al. (2016), and Fekete et al. (2007) generated and used

Methodology	Max. nb. of items	Avg. optimality gap	% of instances solved to opt.
Martello et al. (2007)	50	-	89
Fekete et al. (2007)	80	-	72
Hifi et al. (2010)	90	21.27	-
Junqueira et al. (2012)	100	1.77	50
Paquay et al. (2016)	27	8.37	54

Table 2.1: Performance of the exact methods.

their own instances, rather than the Martello et al. (2000) benchmark instances. There are two main issues with the proposed approaches. The exact methods are computationally slow. Solving a medium size problem to optimality is still computationally very challenging. Heuristics on the other hand consider only a limited number of options or disregard basic practical constraints such as bin stability and item support. We believe the methodology we propose is a serious attempt at using exact approaches, namely, column generation to solve large problem instances while being able to accommodate practical constraints such as item support.

2.4 Practical Constraints

The first practical constraint that was considered in the literature is bin stability, which refers to the resilience that pallets should have when subject to acceleration or movement during transportation. This is also called load balancing. Davies and Bischoff (1999), Bortfeldt and Gehring (2001), and Eley (2002) incorporated this constraint into their proposed methodologies. In their approach, they tried to align the center of mass of the pallets to the geometric centre of the container. More recent work in this field is provided by Trivella and Pisinger (2016) and Ramos et al. (2018). Trivella and Pisinger (2016) provided a mathematical model of the CLP with load balancing constraints and used a multi-level local search heuristic to solve the problem. In the first level, they explore the transitive orientations of the feasible packing graphs. In the second, they change the structure of these graphs. Finally, in the third, they exchange items between weakly balanced bins. They also provided lower bounds for the problem. They showed that their method can solve instances with up to 200 items in a few seconds. Ramos et al. (2018) further examined the problem in terms of placing the center of mass of the pallets based on specific vehicles used. They proposed a multi-population biased random-key genetic algorithm, where they designed a new fitness function that takes static stability and load balance into account. They also provided an instance generator that includes item weights on top of item dimensions. However, they based their weight values on a simple beta function, without an in-depth analysis of real-life data.

Ceschia and Schaerf (2013) worked on a multi-drop multi-container loading problem with several practical constraints, such as item orientations, load bearing, and different drop-off locations for the packed items. They use local search metaheuristics (based on tabu search and simulated annealing), where the search space is a sequence of boxes to be packed, rather than their placements. After they obtain a sequence, they place the items following a greedy heuristic that exploits the remaining space and the presence of identical items, while taking the practical constraints into account. They tested their method on real-life instances as well as benchmark instances, and found that it outperforms most other heuristics in terms of volume use and provides solutions in a few minutes of computation time.

Related to the previous work, Junqueira et al. (2012) provided the first MIP model that considers weight distribution and load bearing and tries to tackle the load balancing constraints. They tested their model on randomly generated instances. However, since their model is highly complex, it can only find solutions to instances with up to 100 items, in a few hours of computational time.

A more in depth MIP model is presented by Paquay et al. (2016) that includes constraints such as load bearing, weight distribution, item orientations, and different container shapes. Their work was derived from a real-life air cargo transportation application. Their model captures the most amount of practical constraints in the literature by far. However, since it is an exceptionally complex mathematical model, they were only able to use it to solve instances with up to 13 items to optimality, using branch-and-bound. They further extended this work in Paquay et al. (2017) by providing three mathheuristics: Relax-and-Fix, Insert-and-Fix and Fractional Relax-and-Fix. These heuristics are based on the decomposition of the original problem into smaller subproblems. They tested their approaches on instances with up to 100 items and a computational time limit of one hour and provided comparable results to the branch-and-bound approach in a fraction of the time.

Finally, Toffolo et al. (2017) introduced a heuristic decomposition method to solve the multiple container-size loading problem introduced by the Renault Challenge (Clautiaux et al., 2015). Their method takes advantage of the structure included in the challenge. They first convert the 3D items into 2D and place them into stacks, and then use these stacks to construct bins, hence decomposing the problem by grouping items together. They generate stacks by solving a 2D placement problem. Their approach is similar to what is presented in this thesis, albeit without the help of exact mathematical approaches such as column generation.

Looking at the previous work, the lack of research that considers the full problem, with all of the practical constraints that are used in the industry, is apparent. Most of the work on practical constraints either simplify or do not consider a subset of the requirements. The limited number of approaches that consider the full problem cannot provide solutions in industry time limits. This is expected, since the 3DBPP with no other constraints is already NP-Hard and the practical constraints add even more complexity to the problem. However, in this thesis, we show that all of the practical constraints can be tackled in acceptable computational time with the help of efficient decomposition techniques and smart algorithms. Indeed, this is the first work in the literature that tackles the DPLP in its entirety. Additionally, we provide realistic benchmark instances with practical item attributes so that the future work on DPLP can easily provide comparisons.

Chapter 3

A Layer-based Column Generation Solution Approach

Layer building approaches for packing problems are introduced by George and Robinson (1980). Lodi et al. (2004) proposed layering based mathematical models for the SPP and the 2DBPP and introduced new combinatorial lower bounds that can be computed in O($nlogn$). Based on these models, Bettinelli et al. (2008) developed a branch-and-price for the SPP. The branching rule that they used is based on the initializing item for each layer. Finally, Cui et al. (2017) worked on a triple solution approach where they use column generation and a residual algorithm to obtain one-dimensional layers. These papers generally build layers with single type items, or do not generate layers with homogeneous height, undercutting the real-world applicability of the solutions. Additionally, they work on simpler two-dimensional problems, and do not allow for multiple items to be stacked vertically inside each layer. In our approach, however, we consider complex two-dimensional layers individually in a column generation procedure, allowing vertical stacking throughout each layer.

Consider a set of items $\mathcal{I}$, each with width w_i, depth d_i, height h_i, and an unlimited number of identical containers (bins) $b \in \mathcal{B}$ with width W, depth D, and height H. Let layer $l \in \mathcal{L}$ be a two-dimensional item arrangement where items are confined to the horizontal area of a bin and no items are on top of each other. Let continuous variables c_i^1 and c_i^2 represent the width and depth coordinates of the front, left, bottom corner of item i. Define binary variables x_{ij} and y_{ij} that take value 1 if item i precedes item j along the width and depth directions, respectively. Let binary variables z_{il} take value 1 if item i is in layer l. A layer l is defined by the following constraints:

$$x_{ij} + x_{ji} + y_{ij} + y_{ji} \geq z_{il} + z_{jl} - 1 \qquad i, j \in \mathcal{I}, i < j \tag{3.1}$$

$$x_{ij} + x_{ji} \leq 1 \qquad i, j \in \mathcal{I}, i < j \tag{3.2}$$

$$y_{ij} + y_{ji} \leq 1 \qquad i, j \in \mathcal{I}, i < j \tag{3.3}$$

$$c_i^1 + w_i \leq c_j^1 + W(1 - x_{ij}) \qquad i, j \in \mathcal{I}, i \neq j \tag{3.4}$$

$$c_i^2 + d_i \leq c_j^2 + D(1 - y_{ij}) \qquad i, j \in \mathcal{I}, i \neq j \tag{3.5}$$

$$0 \leq c_i^1 \leq W - w_i \qquad i \in \mathcal{I} \tag{3.6}$$

$$0 \leq c_i^2 \leq D - d_i \qquad i \in \mathcal{I} \tag{3.7}$$

$$z_{il} \in \{0, 1\} \qquad i \in \mathcal{I}, l \in \mathcal{L} \tag{3.8}$$

$$x_{ij}, y_{ij} \in \{0, 1\} \qquad i, j \in \mathcal{I}, i \neq j \tag{3.9}$$

Constraints (3.1) enforce at least one relative positioning relationship between each pair of items in a layer. Constraints (3.2) and (3.3) ensure that there is at most one spacial relationship between items i and j along each of the width and depth dimensions. Constraints (3.4) and (3.5) are the non-overlapping constraints. Constraints (3.6) and (3.7) ensure that items are placed within the boundaries of the bin. Constraints (3.9) define the domain of the variables x_{ij} and y_{ij}.

Since a layer is a two-dimensional arrangement of items and it is not affected by the specific bin that it is placed in, we consider an easier ODPP to generate them. Define continuous variable o_l to represent the height of layer $l \in \mathcal{L}$, and binary variable z_{il} which takes value 1 if item $i \in \mathcal{I}$ is included in layer $l \in \mathcal{L}$. Using the layer definition constraints, and variables o^l and z_{il}, the ODPP is formulated as:

$$[LODPP]: \min_{o,z,x,y,c} \sum_{l \in \mathcal{L}} o_l \tag{3.10}$$

$$\text{s.t.} \sum_{l \in \mathcal{L}} z_{il} = 1 \qquad i \in \mathcal{I} \tag{3.11}$$

$$o_l \geq h_i z_{il} \qquad i \in \mathcal{I}, l \in \mathcal{L} \tag{3.12}$$

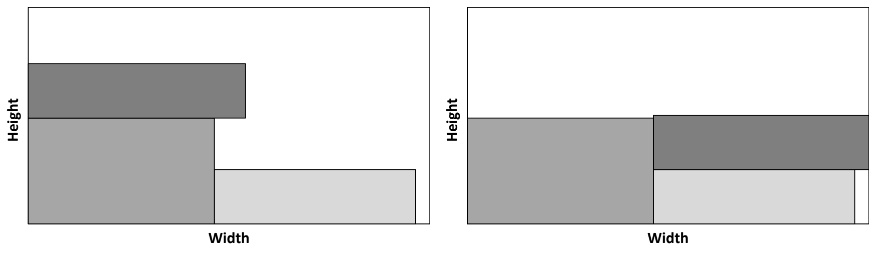
$$\sum_{i \in \mathcal{I}} w_i d_i z_{il} \leq WD \qquad l \in \mathcal{L} \tag{3.13}$$

$$(3.1) - (3.9)$$

$$o_l \geq 0 \qquad l \in \mathcal{L} \tag{3.14}$$

The goal of $[LODPP]$ is to build layers that cover every item in a way that the total height of such layers is minimized (3.10). Constraints (3.11) ensure that all items are covered. Constraints (3.12) determine the height of each layer $l \in \mathcal{L}$. Constraints (3.13) are valid inequalities which state that the total surface area of all items in a layer cannot exceed the surface area of the bin.

Even though this set-partitioning like formulation is prone for solution by decomposition, it does not allow for multiple stacked items within a layer. Consequently, it ignores certain arrangements of items, possibly resulting in sub-optimal solutions. This issue is



(a) Item placement without superitems. (b) Item placement with superitems.

Figure 3.1: Better utilization of the height dimension using superitems.

clearly visualized in Figure 3.1, where the more compact arrangement in (b) is not considered by $[LODPP]$ whereas the arrangement in (a) is. This is, however, done intentionally to reduce the complexity of the formulation and to push some of the considerations to the subproblem, when decomposition is used. This is where we use the concept of superitems. A superitem is a collection of individual items that are stacked together. A superitem is useful as it is a stable collection of items by design, and it provides support to items placed on top as single items do. Superitems are treated as new items and are added to set $\mathcal{I}$ to form set $\mathcal{CI}$, which includes both the original items and the superitems.

Let us define binary parameter f_{si} that takes value 1 if candidate item $s \in \mathcal{CI}$ includes item $i \in \mathcal{I}$, and modify the decision variable z_{il} to z_{sl}; i.e. z_{sl} takes value 1 if item $s \in \mathcal{CI}$ is included in layer $l \in \mathcal{L}$. The width and depth of superitems are denoted by w_s and d_s.

The extended formulation with superitems is:

$$
\begin{aligned}
[SIODPP]: \min_{o,z,x,y,c} \; & \sum_{l \in \mathcal{L}} o_l \\
\text{s.t.} \; & \sum_{l \in \mathcal{L}} \sum_{s \in \mathcal{CI}} f_{si} z_{sl} = 1 && i \in \mathcal{I} \\
& o_l \geq h_s z_{sl} && s \in \mathcal{CI}, l \in \mathcal{L} \\
& \sum_{s \in \mathcal{CI}} w_s d_s z_{sl} \leq WD && l \in \mathcal{L} \\
& x_{sj} + x_{js} + y_{sj} + y_{js} \geq z_{sl} + z_{jl} - 1 && j > s : s, j \in \mathcal{CI}, l \in \mathcal{L} \\
& x_{sj} + x_{js} \leq 1 && j > s : s, j \in \mathcal{CI} \\
& y_{sj} + y_{js} \leq 1 && j > s : s, j \in \mathcal{CI} \\
& c_s^1 + w_s \leq c_j^1 + W(1 - x_{sj}) && s \neq j : s, j \in \mathcal{CI} \\
& c_s^2 + d_s \leq c_j^2 + D(1 - y_{sj}) && s \neq j : s, j \in \mathcal{CI} \\
& 0 \leq c_s^1 \leq W - w_s && s \in \mathcal{CI} \\
& 0 \leq c_s^2 \leq D - d_s && s \in \mathcal{CI} \\
& x_{sj}, y_{sj} \in \{0, 1\} && s \neq j : s, j \in \mathcal{CI} \\
& z_{sl} \in \{0, 1\} && l \in \mathcal{L}, s \in \mathcal{CI} \\
& o_l \geq 0 && l \in \mathcal{L}
\end{aligned}
$$

Unless all possible superitem combinations are considered, the formulation does not guarantee solutions to the original problem. However, our computational experiments show that even with a limited number of superitems, the proposed approach outperforms all solution methodologies in the literature. Details on how to generate superitems for both theoretical and practical implementations are given in Section 3.2.

3.1 Solution by Column Generation

Although $[LODPP]$ models the ODPP using layers, it is a large MIP formulation with many binary variables. Testing on industry data, we found that it solves instances with up to 20 items to optimality within 5 minutes using a commercial solver. Without valid inequalities (3.13), the solution takes 30 minutes. However, instances with 50 items are not solvable within 24 hours. To remedy this, we adopt a column-generation framework. In fact, the formulations are developed with this in mind. Using the set $\mathcal{L}$ that contains all possible layers, continuous parameter $\overline{o}_l$ for the height of a layer $l \in \mathcal{L}$, binary parameter $\overline{z}_{sl}$ that takes value 1 if layer l includes candidate item s, and binary decision variable α_l that takes value 1 if layer l is selected, the layer based open dimension bin packing formulation is:

$$[LCODPP] : \min_{\alpha} \sum_{l \in \mathcal{L}} \alpha_l \overline{o}_l \tag{3.15}$$

$$\text{s.t.} \sum_{l \in \mathcal{L}} \sum_{s \in \mathcal{CI}} f_{si} \overline{z}_{sl} \alpha_l \geq 1 \qquad i \in \mathcal{I} \tag{3.16}$$

$$\alpha_l \in \{0, 1\} \qquad l \in \mathcal{L} \tag{3.17}$$

Using binary variables t_b that take value 1 if bin $b \in \mathcal{B}$ is used and u_{lb} that take value

1 if layer $l \in \mathcal{L}$ is assigned to bin $b \in \mathcal{B}$, the layer-based 3DBPP is:

$$[3DLCBPP] : \min_{\alpha,t,u} \sum_{b\in\mathcal{B}} t_b \tag{3.18}$$

$$\text{s.t.} \sum_{l\in\mathcal{L}} \sum_{s\in\mathcal{CI}} f_{si}\overline{z}_{il}\alpha_l \geq 1 \qquad i \in \mathcal{I} \tag{3.19}$$

$$\sum_{l\in\mathcal{L}} \overline{o}_l u_{lb} \leq H \qquad b \in \mathcal{B} \tag{3.20}$$

$$t_b \geq u_{lb} \qquad b \in \mathcal{B}, l \in \mathcal{L} \tag{3.21}$$

$$\alpha_l \leq u_{lb} \qquad b \in \mathcal{B}, l \in \mathcal{L} \tag{3.22}$$

$$\alpha_l, t_b, u_{lb} \in \{0,1\} \qquad l \in \mathcal{L}, b \in \mathcal{B} \tag{3.23}$$

Constraints (3.16) and (3.19) ensure that each item is covered. Constraints (3.20) ensure that the total height of the layers assigned to a bin does not exceed the height of the bin. Constraints (3.21) and (3.22) make sure that a bin is used and a layer is selected when a layer is assigned to a bin. The main differences between $[LCODPP]$ and $[SIODPP]$ are that items are pre-assigned to layers and their placements are pre-determined.

The optimal layers for $[LCODPP]$ and $[3DLCBPP]$ may be different. However, the difference between the two is expected to be minimal. In principle, $[LCODPP]$ provides a lower bound to $[3DLCBPP]$. It can be used to produce a feasible solution by introducing separator sheets to $[LCODPP]$ at height intervals of H. This way, the open dimension is separated into bins with height H. Adding the separator constraints to $[LCODPP]$ should yield a good feasible solution to $[3DLCBPP]$ since decreasing the total layer height also decreases the number of "bins". Because $[LCODPP]$ is a much easier problem to solve, we use it for column generation.

The linear relaxation of $[LCODPP]$ is the Dantzig-Wolfe master problem. When defined on a subset of layers $\mathcal{L}'$, we obtain the restricted master problem $[RMP]$.

$$
\begin{aligned}
[RMP] : \min_{\alpha} & \sum_{l \in \mathcal{L}'} \alpha_l \bar{o}_l \\
\text{s.t.} & \sum_{l \in \mathcal{L}'} \sum_{s \in \mathcal{CI}} f_{si} \bar{z}_{sl} \alpha_l \geq 1 && i \in \mathcal{I} \\
& \alpha_l \geq 0 && l \in \mathcal{L}'
\end{aligned}
$$

Let λ_i, $i \in \mathcal{I}$ be the dual variables corresponding to constraints (3.16). The reduced cost of a new layer l is $o_l - \sum_{i \in \mathcal{I}} \sum_{s \in \mathcal{CI}} \lambda_i f_{si} z_{sl}$. The pricing subproblem to generate new layers (columns) is:

$$
\begin{aligned}
[SP] : \min_{o,z,x,y,c} \quad & o_l - \sum_{i \in \mathcal{I}} \sum_{s \in \mathcal{CI}} \lambda_i f_{si} z_{sl} && (3.24) \\
\text{s.t.} \quad & (3.1) - (3.9) \\
& \sum_{s \in \mathcal{SI}} w_s d_s z_{sl} \leq WD
\end{aligned}
$$

In $[SP]$, only items with $\lambda_i \geq 0$ are considered, since it is a minimization problem and thus items with $\lambda_i < 0$ will not be included in the optimal solution. New columns (o^l, z_{sl}) are added to $[RMP]$ as long as they have a negative reduced cost. If all such columns are included in $[RMP]$, a lower bound to $[LCODPP]$ is obtained.

3.1.1 Solution of the Pricing Subproblem

The subproblem $[SP]$ takes less than one minute to solve using CPLEX when the number of items considered in a layer is 20 or less. As the number of items increases, the solution time increases significantly. For example, when $|\mathcal{I}| > 100$, CPLEX takes more than one hour of CPU time. Since $[SP]$ is solved repeatedly in column generation, such times are not acceptable. To overcome this, we suggest to solve a relaxed problem where placement is ignored and we use the property that the objective function of $[SP]$ is only a function of the items in the layer (variable z_{sl}) and not on the placement (variables c_s^1 and c_s^2) to suggest an iterative algorithm to solve $[SP]$.

The algorithm first ignores the placement constraints (3.1)-(3.9) and finds a set of items such that the reduced cost is minimized. If the reduced cost is nonnegative, the column generation algorithm terminates. Else, a set of items $\mathcal{S}$ is found such that the reduced cost is negative. The algorithm then checks if there is a feasible placement of $\mathcal{S}$ in a layer, by solving $[SP]$ for the set $\mathcal{S}$ instead of $\mathcal{I}$ until a feasible solution is found. If such a feasible placement exists, the algorithm terminates and the column is added to the master problem. Else, a feasibility constraint $\sum\limits_{s \in S} z_{sl} \leq |S| - 1$ is added to the relaxed subproblem, and the iterative algorithm continues. Because of the valid inequality (3.13), $|\mathcal{S}|$ is expected to be much smaller than $|\mathcal{I}|$, which makes the feasibility check fast regardless of the size of $\mathcal{I}$. Based on our experiments, even for problems with $|\mathcal{I}| > 100$, the iterative algorithm succeeds in finding good layers within a maximum of one minute.

To improve the solution time, $[SP]$ could also be solved heuristically so that multiple columns can be added to $[RMP]$ at each iteration of column generation, reducing the number of iteration needed for termination. There are numerous well-performing 2D packing algorithms in the literature. In this work, we use the placement algorithm called Maxrects implemented by (Jylänki, 2010) since it is readily available as a C++ library and was found to perform well in our experiments.

Maxrects uses 5 different objectives in object placement. We generate layers using all of these objectives. Since the reduced cost of a layer l is $o^l - \sum\limits_{i \in \mathcal{I}_l} \sum\limits_{s \in \mathcal{CI}} \lambda_i f_{si} z_{sl}$, items with higher dual variables are more likely to minimize the reduced cost. Therefore, we sort the items in decreasing order of λ_i and use Maxrects to place them. Experimentally, a layer is generated in less than 0.0001 seconds on average for instances with up to 3000 items.

3.1.2 Branch-and-Price and Column Generation Frameworks

We introduce both a layer-based column generation algorithm (LCGA) and a branch-and-price method to obtain solutions to 3DBPP. In this subsection, we explain both of these approaches.

To solve $[LODPP]$ to proven optimality, we incorporate the column generation in a branch-and-price scheme. We use the Ryan and Foster (1981) branching rule where two items are either forced to belong to the same layer or to different layers. To find the constraints to branch on, we scan the coefficient matrix of [RMP] to identify two items m and n, and two layers (columns) where m and n are both in the first layer, but only one is in the second layer. The columns with the highest α_l values are picked first. In the left child node, m and n are forced to be together, while in the right only one is (see Vance et al. (1994) for more details). Since the left child reduces the size of the model, it provides an easier problem to solve. Therefore, we use a depth-first search through the left child when traversing the branch-and-bound tree. Note that instead of adding these rules as constraints to the master problem, we simply remove the columns that do not satisfy them from the respective child node.

These restrictions can easily be applied to the column generation subproblem. To generate columns in the left child node, we add a constraint in the form of $\sum_{s \in \mathcal{CI}} f_{sm} z_{sl} = \sum_{s \in \mathcal{CI}} f_{sn} z_{sl}$. For the right child, the constraint to be added is $\sum_{s \in \mathcal{CI}} (f_{sm} + f_{sn}) z_{sl} \leq 1$, where we remove the super items that include both items m and n from $\mathcal{CI}$ to reduce the problem size.

To begin the column generation process with a warm start, we first generate a subset of layers using Maxrects with randomly generated dual variables. Also $|\mathcal{I}|$ single item layers are included to ensure feasibility. As proven by Vance et al. (1994), it is not always necessary to solve the root node $[RMP]$ to optimality to obtain a lower bound on $[LCODPP]$. Where v_B is the optimal value of $[RMP]$ over the current set of columns and $\overline{c}_{min}$ is the reduced cost of the column given by the optimal solution to $[SP]$, column generation in the root node is terminated when $\lceil v_B \rceil = \lceil v_B/(1 - c_{min}) \rceil$. Let v_{LB} be equal to the final v_B value obtained from the column generation process.

We also suggest a column generation heuristic to construct feasible solutions for large instances of 3DBPP. The column generation algorithm terminates when no columns with negative reduced cost can be generated, or when the objective value of $[RMP]$ is not improved in the last 20 iterations. After layer generation, layers with less than 50% density $\left(\frac{\sum_{i \in \mathcal{I}_l} w_i d_i}{WD}\right)$ are discarded, since their use would lead to unstable bins. The heuristic selects layers ordered in descending order of density, until all items are covered. Algorithm 1 showcases this heuristic, where $\overline{\mathcal{L}}$ and $\mathcal{SL}$ refer to the sets of generated and selected layers, respectively. The parameter de_l is the density of layer l and it is calculated using the expression $\frac{\sum_{s \in \mathcal{CI}_l} w_s d_s h_s}{WDo^l}$. Layers with higher densities are prioritized, since they implicitly improve bin stability. Based on our tests, many of the layers with high density have common items. To be able to select more of these layers, we allow each item to be covered at maximum 3 times. We also let each selected layer to have a maximum of 3 items that are covered using the previously selected layers. To ensure that each item is covered only once, we keep the items that are covered multiple times only in the layer with the highest α_l value, and remove them from the rest of the selected layers. We then place the remaining items in an empty layer using the Maxrects heuristic. Since the removal of items creates empty space, we also try to place items that are not covered yet in this layer to further improve item coverage and increase layer density. This approach is further elaborated on in Algorithm 2, where $\mathcal{SL}$ refers to the set of selected layers.

Algorithm 1 Layer Selection Algorithm

```
1: procedure INITIALIZATION
2:     Set $\mathcal{SL} = \varnothing$.
3: end procedure
4: procedure SELECT–LAYERS($\overline{\mathcal{L}}$)
5:     Sort layers in $\mathcal{SL}$ in descending order by decision variables $\alpha_l$.
6:     for each layer $l \in \overline{\mathcal{L}}$ do
7:         for each item $i$ in layer $l$ do
8:             if item $i$ is already covered then
9:                 Go to next layer.
10:            end if
11:        end for
12:        Set $\mathcal{SL} \leftarrow \mathcal{SL} \cup l$.
13:    end for
14: end procedure
```

Algorithm 2 Layer Reorganization Algorithm

```
1: procedure REPLACE–ITEMS($\mathcal{SL}$)
2:     for each layer $l \in \mathcal{SL}$ do
3:         $\mathcal{SL} \leftarrow \mathcal{SL} \setminus l$.
4:         for each item $i$ in $l$ do
5:             if $i$ is covered before then
6:                 Remove $i$ from $l$.
7:             end if
8:         end for
9:         Clear layer $l$ and place the remaining items in it using Maxrects.
10:        Sort items not in $l$ by $\lambda_i$ in descending order.
11:        Use Maxrects to put the sorted items to $l$.
12:        $\mathcal{SL} \leftarrow \mathcal{SL} \cup l$.
13:    end for
14: end procedure
```

3.1.3 Bin Construction Heuristic

Once a solution to ODPP is found, whether exactly using branch-and-price or heuristically through column generation, the selected layers are used to construct a solution to the 3DBPP. Since denser layers provide more support, the layers are first sorted in decreasing density. Then, L_2 number of bins are opened and the layers are placed sequentially, starting from the lowest possible placement in any bin. This way the layers with higher density are distributed among the open bins, increasing bin stability. If a layer cannot fit in any bin, a new bin is opened. The bin construction procedure is explained in Algorithm 3, where $\mathcal{B}$ refers to the set of constructed bins and $\mathcal{SL}$ represents the set of selected layers.

Parameters h_b and h_l are the total height of the layers in the bin b and the height of layer l, respectively. H is the maximum height of a bin.

Algorithm 3 Bin Construction Algorithm

```
1: procedure INITIALIZATION
2:     Set $|\mathcal{B}| = L_2$.
3: end procedure
4: procedure CONSTRUCT–BINS($\mathcal{SL}$)
5:     Sort layers in $\mathcal{SL}$ in descending order by density.
6:     for each layer $l \in \mathcal{SL}$ do
7:         for each bin $b \in \mathcal{B}$ do
8:             if $h_b + h_l \leq H$ then
9:                 Place $l$ in $b$.
10:            end if
11:        end for
12:        if $l$ cannot be placed in any bin $b \in \mathcal{B}$ then
13:            Open a new empty bin $n$ and place $l$ in $n$.
14:            Set $\mathcal{B} \leftarrow \mathcal{B} \cup n$.
15:        end if
16:    end for
17: end procedure
```

Unfortunately, not all items are always covered by layers. There are two main reasons for this. Layers with less than 50% density are discarded and some items may have heights that are too different from the rest and cannot be grouped in a layer in a stable manner. Therefore, we build the majority of bins using layers and place the remaining items on top in an S-shape (see Figure 3.2), while alternating between short and tall items. Algorithm

4 describes this approach, where $\mathcal{N}$ refers to the set of items that are not covered. The parameter *itr* is the iteration count. To keep the overall height constant throughout the top sections of the bins, we alternate the placement of items in descending and ascending orders of height in each consecutive iteration. This reduces the negative impact on vertical stability. If the majority (e.g., 70%) of the bin is filled with layers and the remaining items occupy only a small portion at the top, the constructed bin would be stable. Minor instabilities can be fixed through shrink wrapping (i.e., horizontally compressing the pallet by wrapping stretch film around it), which is standard in the industry.

Algorithm 4 The S-Shaped Placement Algorithm

```
 1: procedure PLACE–ITEMS($\mathcal{N}$)
 2:     Sort $\mathcal{N}$ in descending order by width and depth.
 3:     for each bin $b \in \mathcal{B}$ do
 4:         $itr \leftarrow 1$.
 5:         while $\mathcal{N} \neq \varnothing$ do
 6:             if $itr \equiv 1 \pmod 2$ then
 7:                 Sort $\mathcal{N}$ in descending order by height.
 8:             else
 9:                 Sort $\mathcal{N}$ in ascending order by height.
10:             end if
11:             Place the items in an S-Shape.
12:             if No other item can be fit in current layer then
13:                 Go up a layer, and set $itr \leftarrow itr + 1$.
14:             end if
15:             if No other item can be fit in current bin then
16:                 Go to next bin.
17:             end if
18:         end while
19:     end for
20: end procedure
```

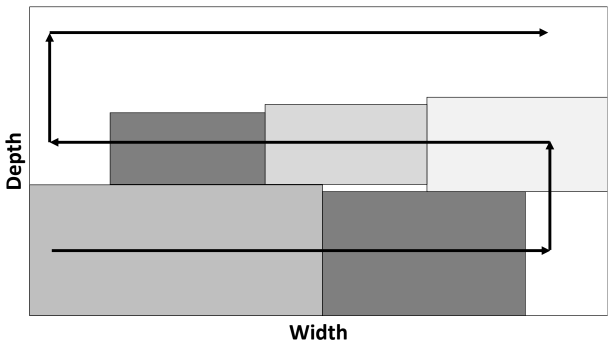


Figure 3.2: S-Shaped placement.

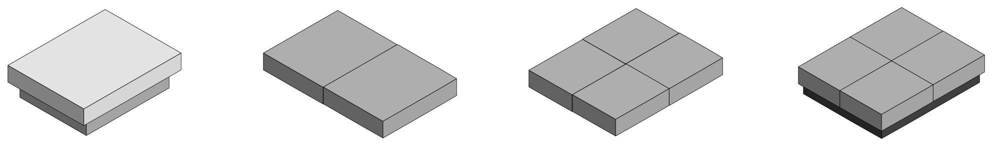
3.2 Layer and Bin Improvement Strategies

To construct dense and stable layers, we propose three enhancement strategies. The first strategy is based on the formation of superitems. The second is a post processing strategy through item replacement. The third is layer reorganization and spacing strategy.

For stability, industry rules require a minimum percentage of the surface area of an item to be supported by the item(s) underneath it. As suggested by our industry partner, we used 70%. To satisfy this, superitems are formed by stacking a large item on top of an almost equal item (see Figure 3.3a). Additionally, superitems are formed by placing identical items horizontally as in Figures 3.3b and 3.3c. Figure 3.3d shows a mixed formation. Larger items are placed on top of smaller items to increase the potential support surface. In our implementation, we use 2 and 4-item horizontal and 2-item vertical formations. While increasing the number of items per formation provides many more potential placements, the computational effort increases significantly. Algorithm 5 describes the superitem generation in more detail, where $\mathcal{SI}$ refers to the set of superitems while $\mathcal{LI}$ refers the set of items from which superitems can be created. In line 20, $|i|$ is the number of items currently in superitem i and max_items is the maximum number of items that can be stacked on top of one another for a superitem. In our experiments, we used the value of 4 for this parameter. At the end of the algorithm, $\mathcal{LI}$ includes all the items and superitems that can be used to generate layers.

Algorithm 5 Superitem Generation Algorithm

```
1:  procedure INITIALIZATION
2:      Set $\mathcal{SI} = \varnothing$.
3:      Set $\mathcal{LI} = \varnothing$.
4:  end procedure
5:  procedure GENERATE SUPER–ITEMS($\mathcal{I}$)
6:      for each set $I_s$ of items with identical dimensions do
7:          if $|I_s| > 1$ then
8:              Create all possible superitems with 2 items by putting items side-by-side.
9:          end if
10:         if $|I_s| > 3$ then
11:             Create all possible superitems with 4 items by putting items side-by-side.
12:         end if
13:         Add the created superitems to $\mathcal{SI}$.
14:     end for
15:     Set $\mathcal{LI} \leftarrow \mathcal{I} \cup \mathcal{SI}$.
16:     Sort each item $i \in \mathcal{LI}$ in ascending order by width, depth, and height.
17:     for each item $i \in \mathcal{LI}$ do
18:         for each item $j > i \in \mathcal{LI}$ do
19:             if $0.7 w_j d_j \leq w_i d_i$ then
20:                 if ($i \in \mathcal{SI}$ and $|i| + 1 \leq max\_items$) OR $i \notin \mathcal{SI}$ then
21:                     Create superitem $k$ by stacking item $j$ over item $i$ such that
22:                     $x_i = \frac{w_j - w_i}{2}$ and $y_i = \frac{d_j - d_i}{2}$.
23:                     Set $\mathcal{LI} \leftarrow \mathcal{LI} \cup k$.
24:                     Set $\mathcal{SI} \leftarrow \mathcal{SI} \cup k$.
25:                 end if
26:             end if
27:         end for
28:     end for
29: end procedure
```



(a) Vertical. (b) 2-item horizontal. (c) 4-item horizontal. (d) Mixed.

Figure 3.3: Superitem formations.

The item replacement strategy is a simple procedure that allows the generation of additional layers with minimal computational effort. The replacement is limited to items that satisfy the minimum industry overlap in surface area with the item being replaced. The approach is described in Algorithm 6, where $\mathcal{NL}$ refers to the set of new layers obtained by replacing an item in a pre-existing layer and $\mathcal{L}$ represents the set of generated layers.

Algorithm 6 Item Replacement Algorithm

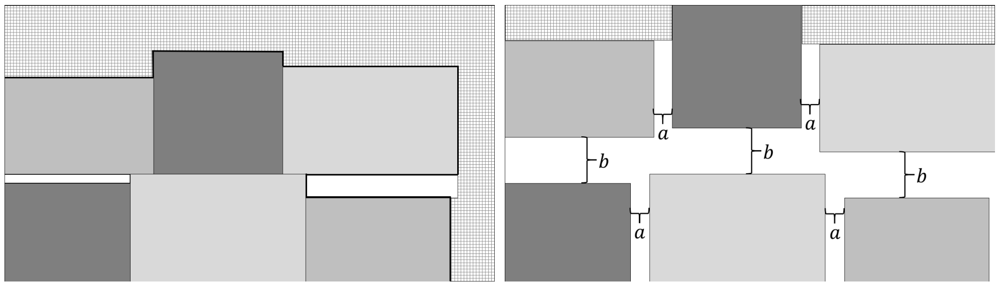
```
procedure INITIALIZATION
    Set $\mathcal{NL} = \varnothing$.
end procedure
procedure REPLACE–ITEMS($\mathcal{I}, \mathcal{L}$)
    for each layer $l \in \mathcal{L}$ do
        for each item $i$ in layer $l$ do
            for each item $j$ not in layer $l$ do
                if $0.7 w_i d_i \leq w_j d_j \leq w_i d_i$ then
                    Replace $i$ with $j$ to create layer $k$.
                    Set $\mathcal{NL} \leftarrow \mathcal{NL} \cup k$.
                end if
            end for
        end for
    end for
    Set $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{NL}$.
end procedure
```

Spacing aims at consolidating dead space to increase density and avoid towers. The marked area in Figure 3.4a shows the wasted space in a bin if spacing is not applied. Therefore we introduce an approach to space items evenly in a layer. One way to space items is to fix the relative positioning variables in constraints (3.1)-(3.9) and solve the following mathematical model:

$$
\begin{aligned}
[SM] : \max_{a,c} \;& a^1 + a^2 && \\
\text{s.t. } & c_i^1 + w_i \leq c_j^1 + W(1 - \overline{x}_{ij}) && i \neq j, i, j \in \mathcal{I} \\
& c_i^2 + d_i \leq c_j^2 + D(1 - \overline{y}_{ij}) && i \neq j, i, j \in \mathcal{I} \\
& c_i^1 \leq W - w_i && i \in \mathcal{I} \\
& c_i^2 \leq D - d_i && i \in \mathcal{I} \\
& a^1 \leq c_j^1 - (c_i^1 + w_i) && i \neq j, \overline{x}_{ij} = 1, i, j \in \mathcal{I} \qquad (3.25) \\
& a^2 \leq c_j^2 - (c_i^2 + d_i) && i \neq j, \overline{y}_{ij} = 1, i, j \in \mathcal{I} \qquad (3.26) \\
& c_i^1, c_i^2 \geq 0 && i \in \mathcal{I}
\end{aligned}
$$

where we try to maximize the continuous variables a^1 and a^2 that denote the minimum distance between each pair of items in the width and depth dimensions, respectively. Constraints (3.25) and (3.26) determine the value of the variables a^1 and a^2. $\overline{x}_{ij}$ and $\overline{y}_{ij}$ are the binary parameters obtained from the relative positions of the items in the layer.

This model, however, is not effective as empty spaces are not necessarily distributed fully throughout a layer (Figure 3.4). Alternatively, we propose a segmented approach. When an item moves in one direction (width or depth), it may impact other items in that direction. Let a segment S_i be the set of items that may be impacted by moving item i in some direction. For the width dimension, the complement to the set $\{j \in \mathcal{S} : (c_j^2 >$



(a) Before spacing. (b) After spacing.

Figure 3.4: Limited item distribution and wasted 2D space with $[SM]$.

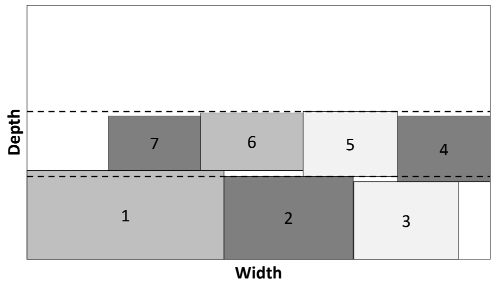


Figure 3.5: Visual representation of the segments.

$c_i^2 + d_i) \cup (c_j^2 + d_j < c_i^2)\}$ defines segment S_i. Figure 3.5 illustrates this concept where the spacing of item 5 in the width dimension can only be affected by items 1, 4, 6, and 7. Note that we only consider unique segments, and we only use the distances between the items that directly follow one another. Additionally, a segment is discarded if it is a subset of another segment.

Using S^w and S^d as the set of segments for the width and depth dimensions, and continuous decision variables a_s^1 and a_s^2 that denote the minimum distance between the items in segment s along the width and depth dimensions, respectively, the layer spacing formulation is:

$$\begin{aligned}
\max_{a,c} \quad & \sum_{s \in \mathcal{S}^w} a_s^1 + \sum_{s \in \mathcal{S}^d} a_s^2 & \\
\text{s.t.} \quad & c_i^1 + w_i \leq c_j^1 + W(1 - \bar{x}_{ij}) & i \neq j, i, j \in \mathcal{I} \\
& c_i^2 + d_i \leq c_j^2 + D(1 - \bar{y}_{ij}) & i \neq j, i, j \in \mathcal{I} \\
& 0 \leq c_i^1 \leq W - w_i & i \in \mathcal{I} \\
& 0 \leq c_i^2 \leq D - d_i & i \in \mathcal{I} \\
& a_s^1 \leq c_j^1 - (c_i^1 + w_i) & i \neq j, \bar{x}_{ij} = 1, i, j \in \mathcal{I}, s \in \mathcal{S}^w \\
& a_s^2 \leq c_j^2 - (c_i^2 + d_i) & i \neq j, \bar{y}_{ij} = 1, i, j \in \mathcal{I}, s \in \mathcal{S}^d \\
& c_i^1, c_i^2 \geq 0 & i \in \mathcal{I}
\end{aligned}$$

where the objective is to maximize the minimum distance between the items in each segment. Since segment definitions in one dimension may change with spacing in the other dimension, we decompose this model and apply spacing to each dimension sequentially. The segmented spacing formulation in the width dimension is:

$$\begin{aligned}
\max_{a,c} \quad & \sum_{s \in \mathcal{S}_w} a_s^1 & \\
\text{s.t.} \quad & c_i^1 + w_i \leq c_j^1 + W(1 - \bar{x}_{ij}) & i \neq j, i, j \in \mathcal{I} \\
& 0 \leq c_i^1 \leq W - w_i & i \in \mathcal{I} \\
& a_s^1 \leq c_j^1 - (c_i^1 + w_i) & i \neq j, \bar{x}_{ij} = 1, i, j \in \mathcal{I}, s \in \mathcal{S}^w \\
& c_i^1 \geq 0 & i \in \mathcal{I}
\end{aligned}$$

After this model is solved and the set S^d is updated, a similar model is used to space in the depth dimension.

3.2.1 Practical Requirements

In practice, three-dimensional bin packing has to satisfy important practical constraints. In distributor's pallet loading and container loading problems, item support, bin stability, and load bearing are particularly important. The proposed layering approach is suitable to deal with some of these constraints, either explicitly or implicitly, when the layers are being formed and selected to create feasible solutions.

To ensure item support, we divide the set of items $\mathcal{CI}$ in (3.1)-(3.9) into item groups $\mathcal{IG}$ that only contain items of the same height. We allow a tolerance of 5 mm, which is an industry standard. It is important to note that an item may be in multiple item groups $g \in \mathcal{IG}$ based on its height. Another important practical constraint is family grouping. Items in the same family group (e.g., beverages) should preferably be packed close to each other and in the same bin, which may also be handled in the definition of $\mathcal{IG}$. Finally, load bearing, an important practical constraint, may be accommodated by limiting the sets $\mathcal{IG}$ to only contain items with similar load bearing capabilities. To ensure bin stability, layers are placed in the bins in decreasing order of load bearing capability. This approach decomposes the layer generation subproblem into multiple smaller subproblems, one for each group, which results in a considerable decrease in solution time. Finally, we restrict the number of sets $\mathcal{IG}$ by eliminating groups with $\sum_{i \in g, g \in \mathcal{IG}} w_i d_i \leq 0.5WD$, since they do not lead to dense layers. The grouping approach is explained in Algorithm 7.

Using the column generation and branch-and-price frameworks, pre- and post-processing approaches, and the bin construction heuristic, the overall solution methodology for 3DBPP is summarized in Figure 3.6. The item grouping pre-processing for tackling practical constraints is examined in more detail in Chapter 5.

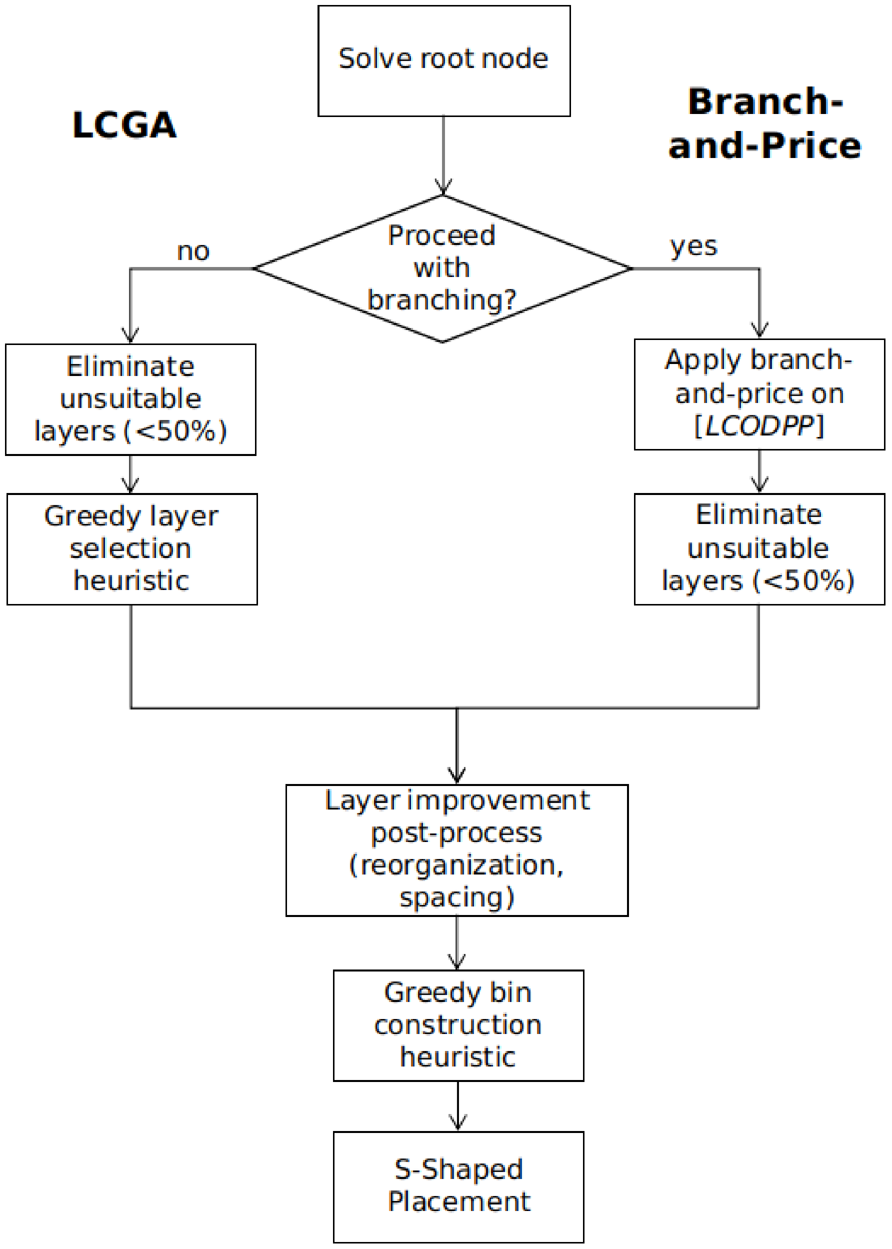

Solve root node
LCGA
Branch-and-Price
Proceed with branching?
no
yes
Eliminate unsuitable layers (<50%)
Greedy layer selection heuristic
Apply branch-and-price on [*LCODPP*]
Eliminate unsuitable layers (<50%)
Layer improvement post-process (reorganization, spacing)
Greedy bin construction heuristic
S-Shaped Placement

Figure 3.6: The overall solution methodology for 3DBPP.

Algorithm 7 Item Grouping Algorithm

```
1: procedure Initialization
2:     Set $\mathcal{IG} = \varnothing$.
3: end procedure
4: procedure GroupItems($\mathcal{CI}$)
5:     Sort items in $\mathcal{CI}$ in descending order by height.
6:     Create an item group $g_1$ for the first item in $\mathcal{CI}$, and set $\mathcal{IG} \leftarrow \mathcal{IG} \cup g_1$.
7:     for each item $i > 1 \in \mathcal{CI}$ do
8:         if $h_i \neq h_{i-1}$ then
9:             Create an item group $g_i$ for item $i$, and set $\mathcal{IG} \leftarrow \mathcal{IG} \cup g_i$.
10:        end if
11:    end for
12:    for each item group $g \in \mathcal{IG}$ do
13:        for each item $i \in \mathcal{CI}$ do
14:            if $h_g - h_i \leq 5$ then
15:                Set $g \leftarrow g \cup i$.
16:            end if
17:        end for
18:    end for
19:    for each item group $g \in \mathcal{IG}$ do
20:        if $|g| = 1$ then
21:            Set $\mathcal{IG} \leftarrow \mathcal{IG} \setminus g$.
22:        else if $\sum_{i \in g} w_i d_i \leq 0.5WD$ then
23:            Set $\mathcal{IG} \leftarrow \mathcal{IG} \setminus g$.
24:        end if
25:    end for
26: end procedure
```

3.3 Computational Experiments

We perform extensive computational testing of the proposed solution approach and compare it to the best solution methodologies. We start by describing standard benchmark instances from the literature and then provide a new and realistic benchmark data set. We compare our approach with the best performing heuristic and exact approaches from the literature on the standard instances, and provide a comparison to Algorithm 864 of Martello et al. (2007) on the new instances. Testing is conducted on a computer with Intel i7-5500U CPU, with 2.40 Ghz and 16 GB on Windows 8.1. The mathematical models are solved using CPLEX Concert technology with CPLEX version 12.6.1. The algorithms are coded in C++ using Visual Studio 2012 IDE.

There is only one standard 3DBPP data set based on the random instance generator of Martello et al. (2000). The set has 9 classes with 50, 100, 150 and 200 items. We generate 360 instances, 10 for each class and item combination. The random instance generator uses the same seed, so all generated instances are guaranteed to be the same as those used in the literature.

3.3.1 Comparison of the Proposed Methodologies

The Branch-and-Price methodology and the Layer-Based Column Generation Algorithm (LCGA) are tested and compared on 50-item instances from Martello et al. (2000). Table 3.1 displays the average number of bins used, the average number of nodes searched, and the average CPU time in seconds. The results show that the branch-and-price method finds optimal solutions within a maximum of 465.3 seconds. Additionally, it finds the optimal fairly close to the root node, which shows us that the gap at the root node is small. On the other hand, LCGA finds good solutions in less than one second. As branch-and-price is not expected to handle large instances, we focus on LCGA for the rest of the experiments.

	BnP			LCGA		
Class	Avg Bin	Avg Nodes	Avg CPU (s)	Avg Bin	Gap (%)	Avg CPU (s)
1	13	5.7	378.4	13.2	1.5	0.56
2	13.1	4.2	349.2	13.3	1.5	0.68
3	12.5	4.8	416.7	12.8	2.4	0.41
4	29.2	3.9	363	29.4	0.7	0.27
5	7.9	3.4	395.9	8.4	6.3	0.43
6	9.5	4.1	387.6	9.8	3.2	0.79
7	7.1	5	448.7	7.4	4.2	0.82
8	8.9	3.3	465.3	9.2	3.4	0.77
9	3.8	3.1	418.5	4	5.3	0.91

Table 3.1: Results for Branch-and-Price.

3.3.2 Comparison of LCGA to the State-of-the-art

To assess the efficacy of the proposed layer based column generation approach, we compare to the best performing algorithms in the literature. Zhao et al. (2016) give a comparison of 10 algorithms (Table 3.2) based on the total number of bins used for classes 1 and 4-8 of Martello et al. (2000). Some classes are omitted as some approaches do not test on them. The comparison is given in Table 3.3. The column IbB-2004 gives the best known lower bound due to Boschetti (2004). The bold results are the best solutions in terms of number of bins used for each class and instance size. LCGA outperforms the best algorithm in the literature in 13 out of 24 class-item size combinations and in all classes except 6 and 7. This is likely because classes 6 and 7 have relatively small bin size and large item dimensions, which is far from being realistic and does not favor layering since only few items can fit in one bin. Even though the improvements on the average number of bins used may not seem too large, it is important to note that the overall optimality gap is 2.8%. The fastest methodology in Table 3.3 is Space Defragmentation by Zhu et al. (2012) with a time limit of 30 seconds per instance. In comparison, LCGA uses only 3.17 seconds on average.

Year	Paper	Code	Name
2000	Martello et al. (2000)	MPV-2000	Algorithm 864
2000	Martello et al. (2000)	MPV-2000-BS	Algorithm 864
2000	Martello et al. (2000)	MPV-2000-Spack	Algorithm 864
2002	Lodi et al. (2002)	L2002-HA	Tabu Search
2002	Lodi et al. (2002)	L2002-TS	Tabu Search
2003	Faroe et al. (2003)	F-2003	Guided Local Search
2008	Crainic et al. (2008)	C-2008	Extreme Point Placement Algorithm
2009	Crainic et al. (2009)	C-2009	TS^2PACK
2012	Zhu et al. (2012)	SD	Space Defragmentation
2004	Boschetti (2004)	lbB-2004	Lower Bound

Table 3.2: List of papers used for comparison in Table 3.3.

In Table 3.4, we compare to Algorithm 864 (MPV-2000) on 100 item instances with standard and larger bin sizes. The larger bin sizes have approximately 10 times the total volume compared to the standard sizes. Increasing bin dimensions reduces the item to bin volume ratio and leads to more realistic data sets. All of the solutions reported for Algorithm 864 are obtained using the provided C code [27] with a CPU time limit of 1000 seconds. In Table 3.4, LB gives the average lower bound (L_2) over 10 instances and #Better provides the number of instances in which LCGA outperforms Algorithm 864. The columns Min, Max, Avg and CPU report the minimum, maximum, and average number of bins used and the average CPU time for both approaches, respectively.

Class	Bin dimensions	Item no.	MPV-2000	MPV2000-BS	MPV2000-Spack	L2002-HA	L2002-TS	F-2003	C-2008	C-2009	SD	LCGA	IbB-2004
1	100x100	50	13.6	13.5	15.3	13.9	13.4	13.4	13.7	13.4	-	**13.2**	12.9
		100	27.3	29.5	27.4	27.6	26.6	26.7	27.2	26.7	-	**26.1**	25.6
		150	38.2	38	40.4	38.1	**36.7**	37	37.7	37	-	**36.7**	35.8
		200	52.3	52.3	55.6	52.7	51.2	51.2	51.9	51.1	-	**50.7**	49.7
Class total			131.4	133.3	138.7	132.3	127.9	128.3	130.5	128.2	127.4	**126.7**	124
4	100x100	50	**29.4**	**29.4**	29.8	**29.4**	**29.4**	**29.4**	**29.4**	**29.4**	-	**29.4**	29
		100	59.1	59	60	59	59	59	59	**58.9**	-	59	58.5
		150	87.2	87.3	87.9	86.9	**86.8**	**86.8**	**86.8**	**86.8**	-	**86.8**	86.4
		200	119.5	119.3	120.3	119	118.8	119	118.8	118.8	-	**118.6**	118.3
Class total			295.2	295	298	294.3	294	294.2	294	293.9	294	**293.8**	292.2
5	100x100	50	9.2	9.1	10.2	8.5	8.4	**8.3**	8.4	**8.3**	-	8.4	7.6
		100	17.5	17	17.6	15.1	15	15.1	15.1	15.2	-	**14.6**	14
		150	24	23.7	24	21.4	20.4	20.2	21	20.1	-	**19.8**	18.8
		200	31.8	31.7	31.7	28.6	27.6	27.2	28.1	27.4	-	**26.9**	26
Class total			82.5	81.5	83.5	73.6	71.4	70.8	72.6	71	70.3	**69.7**	66.4
6	10x10	50	**9.8**	11	11.2	10.5	9.9	**9.8**	10.1	**9.8**	-	**9.8**	9.4
		100	19.4	22.3	24.5	20	19.1	19.1	19.6	19.1	-	**18.9**	18.4
		150	29.6	32.4	35	30.6	29.4	29.4	29.9	**29.2**	-	29.5	28.5
		200	38.2	40.8	42.3	39.1	37.7	37.7	38.5	37.7	-	**37.4**	36.7
Class total			97	106.5	113	100.2	96.1	96	98.1	95.8	**95.5**	95.6	93
7	40x40	50	8.2	8.2	9.3	8	7.5	**7.4**	7.5	**7.4**	-	8.1	6.8
		100	15.3	13.9	15.3	13.3	12.5	**12.3**	13.2	**12.3**	-	12.8	11.5
		150	19.7	18.1	20.1	17.2	16.1	**15.8**	17	**15.8**	-	16.5	14.4
		200	28.1	28	28.7	25.2	23.9	**23.5**	25.1	**23.5**	-	24.3	22.7
Class total			71.3	68.2	73.4	63.7	60	59	62.8	59	**58.4**	61.7	55.4
8	100x100	50	10.1	9.9	11.3	9.9	9.3	9.2	9.4	9.2	-	**9**	8.7
		100	20.2	20.2	21.7	19.9	18.9	18.9	19.5	**18.8**	-	18.9	18.4
		150	27.3	26.8	28.3	25.7	24.1	23.9	25.2	23.9	-	**23.8**	22.5
		200	34.9	34	35	31.6	30.3	29.9	31.3	30	-	**29.2**	28.2
Class total			92.5	90.9	96.3	87.1	82.6	81.9	85.4	81.9	81.3	**80.9**	77.8
Total			769.9	775.4	802.9	751.2	732	730.2	743.4	729.8	**726.9**	728.4	708.8

Table 3.3: Detailed comparison with literature on standard instances.

				LCGA				Algorithm 864			
				Nb. of Bins				Nb. of Bins			
Class	Size (WxDxH)	LB	#Better	Min	Max	Avg	CPU (s)	Min	Max	Avg	CPU (s)
1	100x100x100	25.2	8	24	28	**26.8**	1.9	25	30	27.5	1000
	215x215x215	2	10	3	4	**3.2**	6.05	4	5	4.9	1000
2	100x100x100	24.2	7	23	27	**25.9**	1.97	24	29	26.4	1000
	215x215x215	2	8	2	4	**3.4**	6.47	4	5	4.7	1000
3	100x100x100	24.7	7	23	30	**27.1**	2.05	26	33	29	1000
	215x215x215	2	10	3	4	**3.3**	6.25	4	5	4.9	1000
4	100x100x100	57.8	2	52	65	**59.1**	1.48	52	66	59.2	1000
	215x215x215	3.8	10	4	5	**4.5**	2.68	6	7	6.5	1000
5	100x100x100	13.2	5	12	19	**15.3**	2.28	13	21	17.3	1000
	215x215x215	1.8	10	2	3	**2.2**	1.37	3	4	3.5	1000
6	10x10x10	17.5	6	15	24	**18.2**	2.22	17	21	19.4	1000
	22x22x22	2	3	3	4	**3.1**	0.95	3	4	3.4	1000
7	40x40x40	11	4	11	15	**13.8**	1.93	12	17	15.3	1000
	86x86x86	1.2	4	2	3	**2.3**	0.83	2	4	2.7	1000
8	100x100x100	17.8	5	17	28	**18.9**	3.03	18	28	19.9	1000
	215x215x215	2	10	2	3	**2.6**	3.18	3	4	3.9	1000
9	100x100x100	3	10	4	4	**4**	2.05	5	8	6.7	1000
	215x215x215	1	10	1	1	**1**	0.9	2	4	2.4	1000
	Average	11.78	7.17			**13.04**	2.64			14.07	1000

Table 3.4: Comparison of Algorithm 864 and LCGA with different bin size.

LCGA outperforms Algorithm 864 on all instances based on the average number of bins used. LCGA uses 44.14% less bins than Algorithm 864 for large bins and 5.55% for smaller ones. This proves the efficacy of LCGA to solve realistic instances. Additionally, there was no instance where Algorithm 864 uses less bins than LCGA. LCGA is nearly 500 and 300 times faster for small and large bin sizes, respectively.

3.3.3 Vertical Support

One of the benefits of the layering approach is its ability to handle item support relatively easily. Once a layer is formed, the areas requiring support as well as those that may provide support are known. We exploit such information to account for support explicitly and provide preliminary results. For that, we modify the bin construction (Algorithm 3), the item spacing, and the S-Shaped placement algorithms.

In Algorithm 3, items are placed based on their density. We only check if items are parts of previous layers. To ensure support, we conduct a second check. A layer is placed only if items have at least 70% of their bottom surface covered by the layer underneath. If not, items are spaced to remedy this. For that, a nonlinear mathematical program is used to maximize the overlap between the current layer and the one below it. The objective maximizes the overlap in the width and depth directions.

As the S-shaped placement approach used previously cannot guarantee support, we adopt an extreme point approach that uses a merit function to decide on the placement. The merit function evaluates extreme points and favors items that are placed lower in the bin. Even though the approach is only applied to the top 30% of the bin, all extreme points and empty spaces are considered. Table 3.5 displays preliminary results on two of the generated instances. The columns report the instance details, CPU time, support percentage, number of items with more than 70% support, number of items that have all 4 corners supported, and number of bins used, respectively, with and without accounting for vertical support. The bins are displayed in Figures 3.7 and 3.8.

Class	Items	Instance	CPU (s)	Support (%)	Supported	4-corner	Bins
With support							
1	100	4	2.87	94.52	100	84	1
1	1000	5	299.29	95.17	1000	872	8
Without support							
1	100	4	1.75	87.57	72	51	1
1	1000	5	240.99	92.76	826	634	8

Table 3.5: Vertical support results.

According to Table 3.5, every item is 100% supported with an 85.6% of the items having additional 4-corner support, on average. While there is a slight increase in computational time, the number of bins used did not increase. In an industrial setting minimizing the number of bins is the ultimate measure. Therefore, being able to achieve stability without increasing the number of bins is remarkable. The results are very promising and clearly demonstrate that the layering approach can easily handle a variety of critical practical constraints. The modifications and additions required to tackle vertical support are described in detail in Chapter 5.

3.4 Conclusion

Motivated by a pressing need from the automated warehousing industry, we studied the three-dimensional bin packing problem and its practical counterpart, the distributor's pallet loading problem. Unlike previous models in the literature, we proposed a new formulation that inherently lends itself to branch-and-price and column generation methodologies. The resulting subproblem is a two-dimensional layer generation problem that is solved optimally as well as heuristically. Generated layers are further enhanced using a variety of

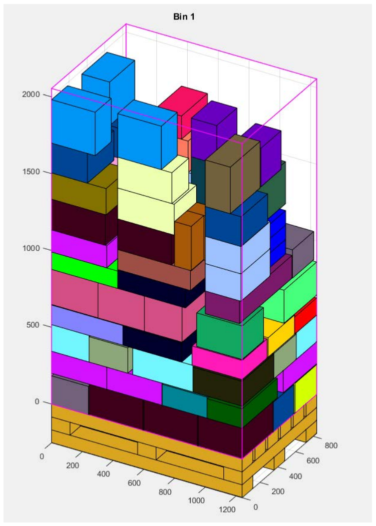


Figure 3.7: Vertical support solution for class 1, 100 items, instance 4.

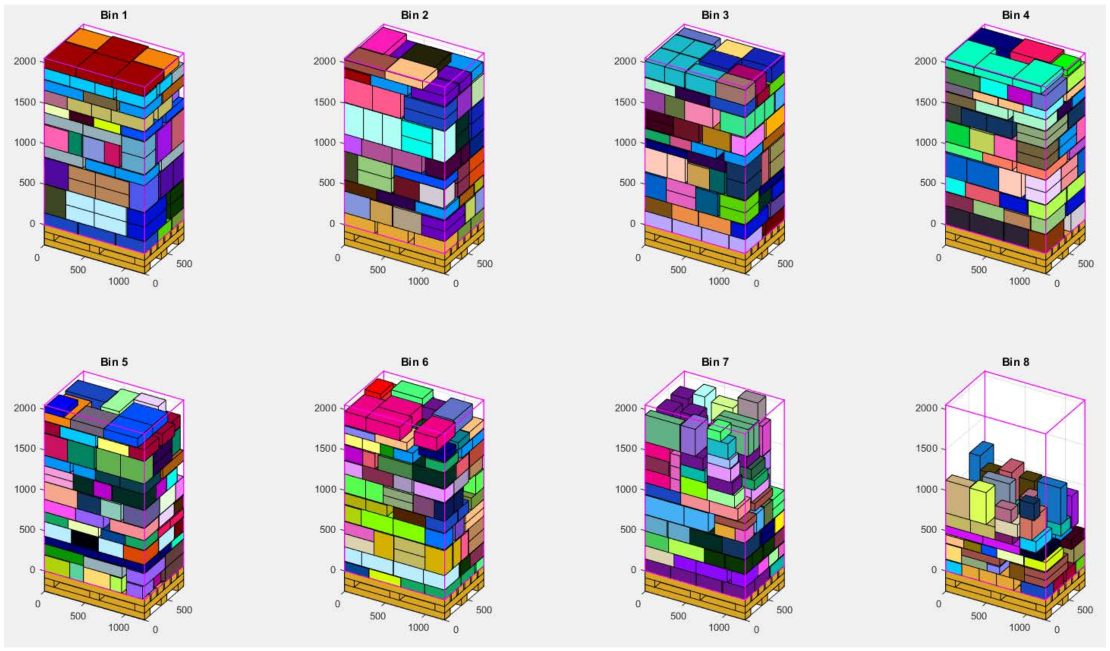


Figure 3.8: Vertical support solution for class 1, 1000 items, instance 5.

strategies such as item grouping, item replacement, layer reorganization, and layer spacing.

Generating layers and using them to construct bins/pallets has numerous advantages both at the packing stage in terms of stability and at the unpacking stage for ease of shelving. In addition, layering can accommodate some of the more difficult practical constraints such as item support and bin stability. We developed the analysis on how to account for support and provided results where every item is fully supported. Even when support is not accounted for explicitly, the layering approach constructed bins where support is guaranteed for the majority of items.

We conducted extensive numerical testing using standard benchmark instances and compared against all previous approaches. The proposed approach found better solutions compared to the best performing approach in the literature in most of the standard benchmark instances both in terms of number of bins used and bin stability within significantly shorter computational times. It was only outperformed by the Space Defragmentation when the instance did not inherently have layers.

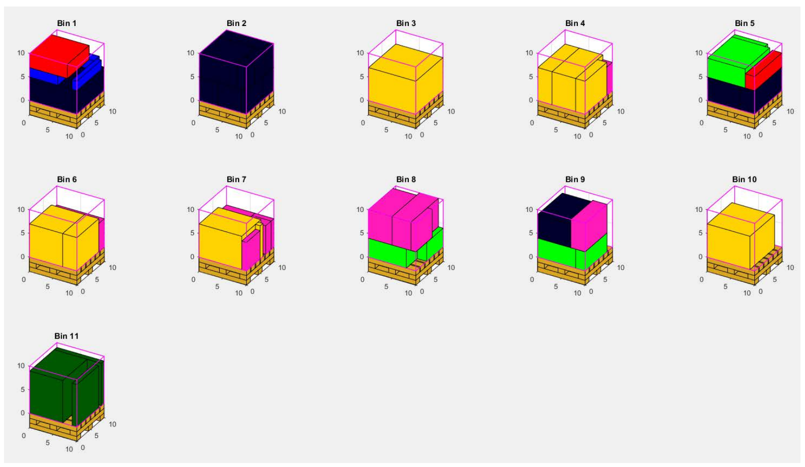
Chapter 4

Generating Realistic Benchmak Instances

A major gap in the three-dimensional packing literature is the lack of realistic benchmark instances. Most of the published work uses the instance generator of Martello et al. (2000). However, solutions to these instances do not represent the packing problems encountered in the industry as items are relatively large compared to the bin. To remedy this, we designed an instance generator that is trained using industrial data. This chapter discusses the details of the instance generator and provides solutions obtained using LCGA.

4.1 Training on Basic Item Features

We obtained 342 real-life data sets from an industrial partner, with number of items ranging from 50 to 3000. All of these sets use bin dimensions of $W = 1200$ mm, $D = 800$ mm, $H = 2055$ mm.


(a) A Martello et al. (2007) instance.

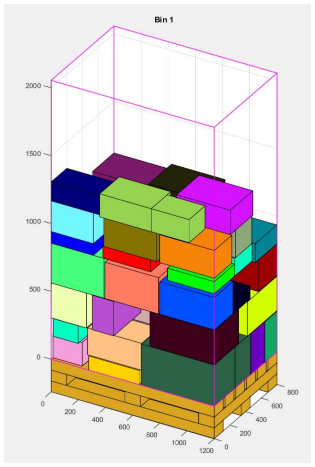
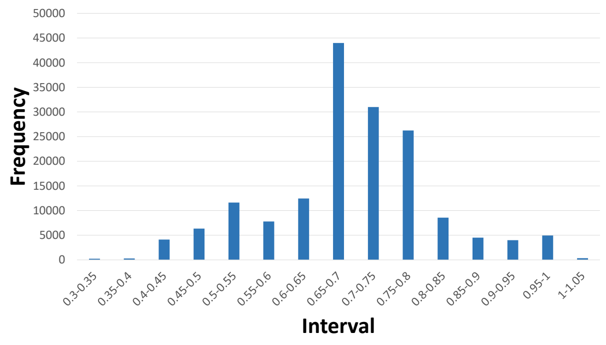

(b) A real-life instance.

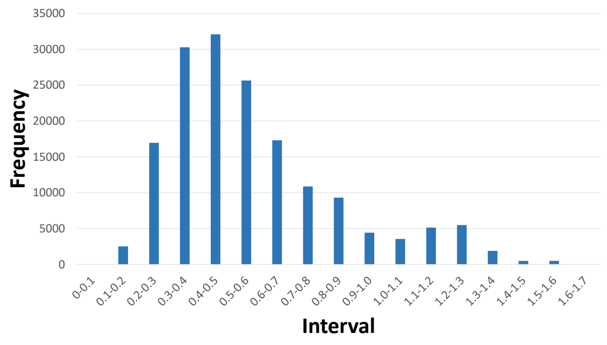
Figure 4.1: Comparison between a standard benchmark and a real-life instance.

When comparing the number of items per bin and the ratio of item to bin dimensions in the industry data set to those from Martello et al. (2007) instance generator, it is obvious that the latter are far from being realistic (Figure 4.1). The main issue is that items are large relative to the bin. Based on the L_2 lower bound, the average number of items per bin is 4.31 in the Martello et al. (2000) instances, while it is at least 90 items per bin for the industry data set. The same was found by Zhu et al. (2012), leading them to use a secondary real-life data set for testing. To best of our knowledge, there is no 3DBPP instance generator that produces realistic instances. To remedy this, we propose a new instance generator.

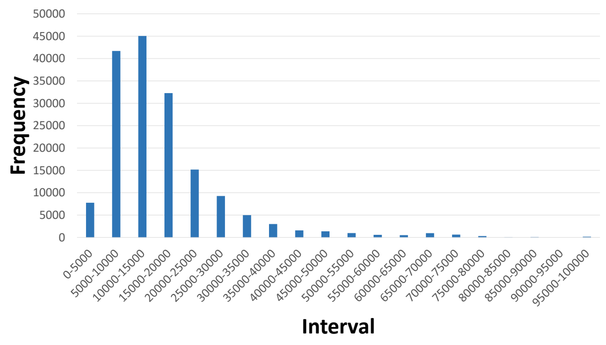
The most important packing-related characteristics of an item are: the ratio of item depth and height to its width, its volume, and its frequency of occurrence. The first two determine the dimensions of an item. The reason for considering the proportions and the volume of an item instead of its dimensions is to construct items with different sizes whose dimensions are properly scaled to each other.



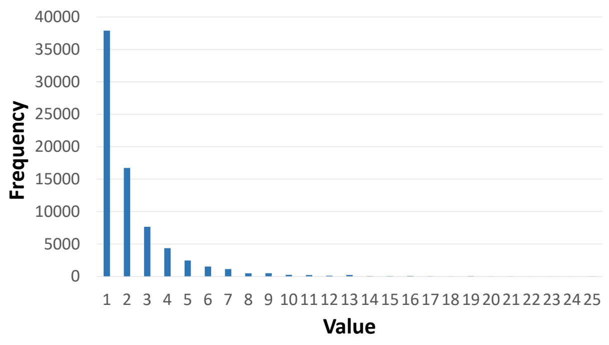
(a) Depth to width (d/w) ratio.



(b) Height to width (h/w) ratio.



(c) Volume.



(d) Frequency of occurrence.

Figure 4.2: Distribution of item characteristics.

The item characteristics are determined based on the industry data we had access to. A total of 166,406 items with 73,978 distinct items are used to construct distribution functions that form the basis of the proposed generator. Figure 4.2 shows histograms for each item characteristic. A distribution fitting package in R, called "fitdistrplus" (Delignette-Muller and Dutang, 2015), is used. It provides the fitting and comparison of different probability distribution functions with goodness-of-fit tests such as Kolmogorov-Smirnov, Cramer von Mises, and log-likelihood. We tested fitting Normal, Lognormal, Gamma, Weibull, and Beta functions to each characteristic. Based on these tests, the best probability distribution functions and their parameters are given in Table 4.1.

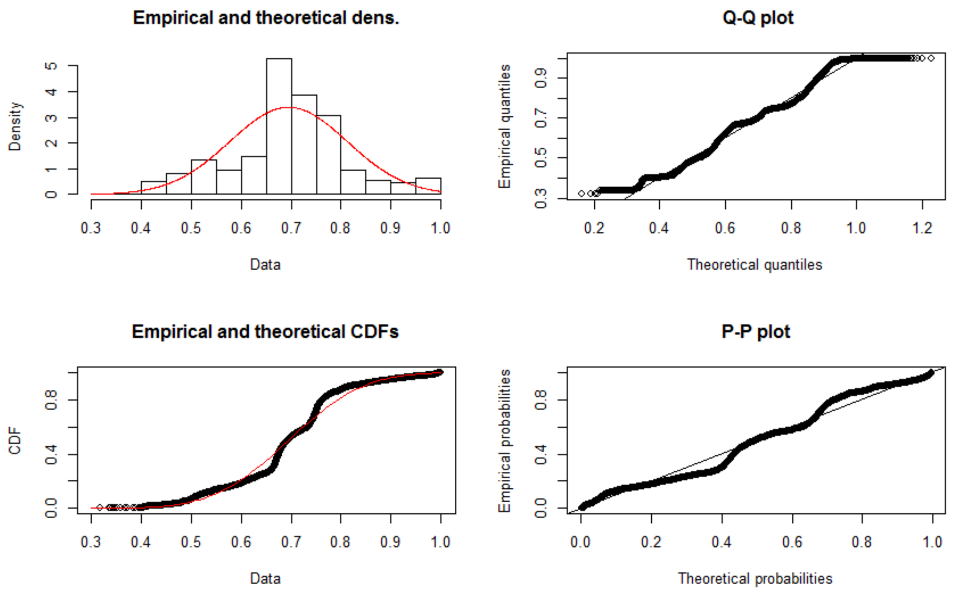


Figure 4.3: Distribution fitting results for depth to width (d/w) ratios.

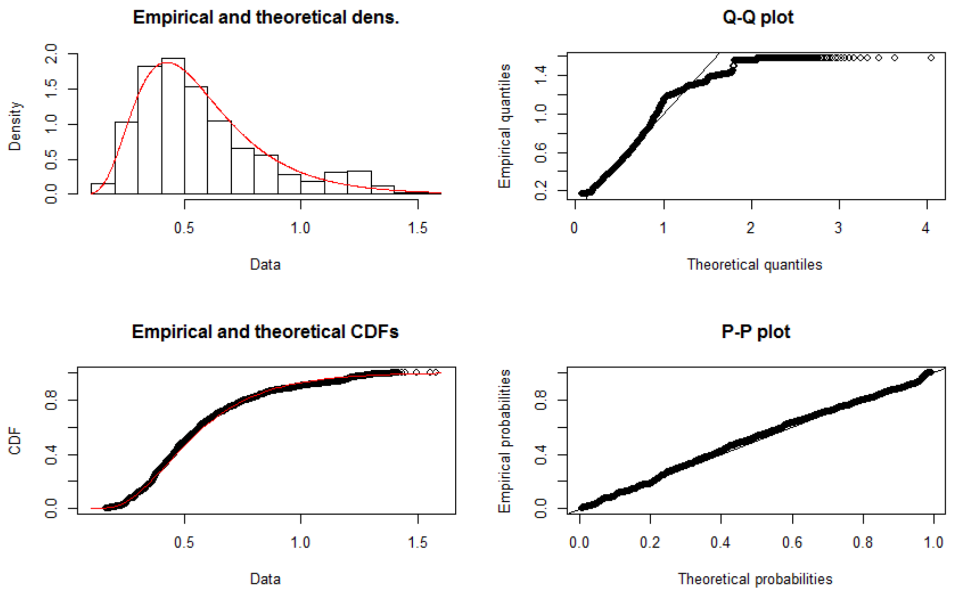


Figure 4.4: Distribution fitting results for height to width (h/w) ratios.

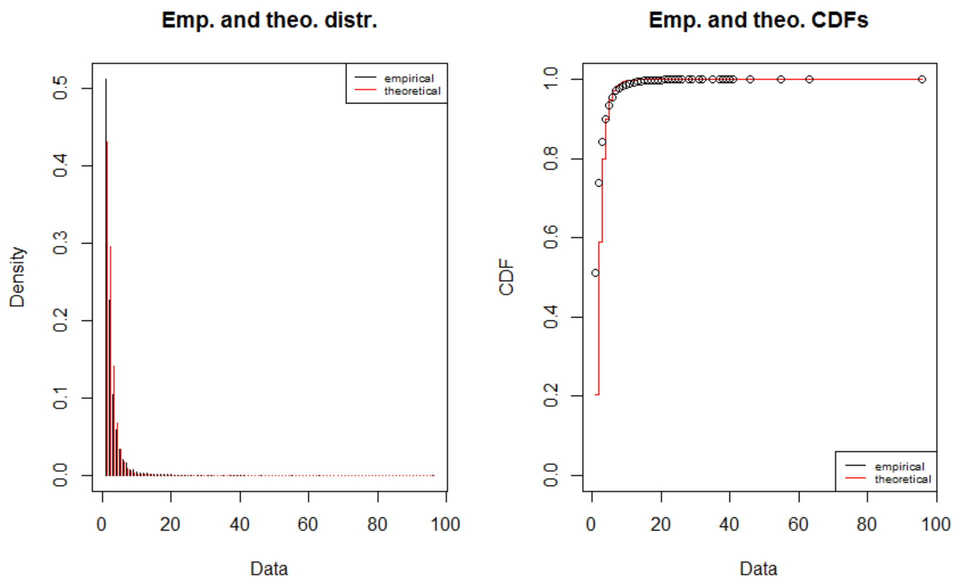
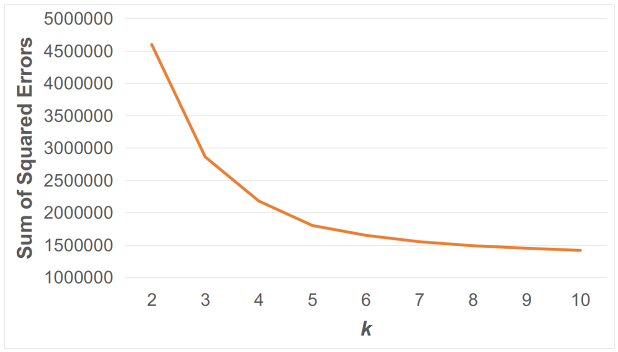


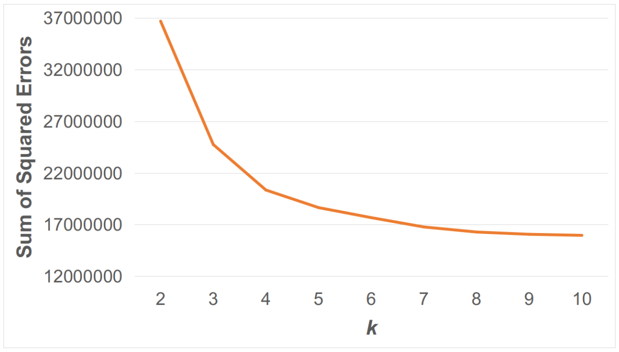
Figure 4.5: Distribution fitting results for item frequency of occurrence.

Characteristic	Distribution	Parameters
Depth to width (d/w) ratio	Normal	(0.695, 0.118)
Height to width (h/w) ratio	Lognormal	(-0.654, 0.453)
Frequency of occurrence	Lognormal	(0.544, 0.658)

Table 4.1: Distribution functions and parameters for item characteristics.



(a) k-clustering results for volumes.



(b) k-clustering results for classes.

Figure 4.6: Sum of squared error plots for k-clustering.

For the volume characteristic (v), we applied k-means clustering to assess its distribution. We tested the k values $[2, 10]$, and provide the plot of the sum of squared errors in Figure 4.6a. The analysis separates the 166,406 items into five categories, as using more categories does not decrease the error significantly. Category 1 has 72,037 items and $v \in [2.72, 12.04]$, category 2 has 55,436 items and $v \in [12.05, 20.23]$, category 3 has 26,254 items and $v \in [20.28, 32.42]$, category 4 has 9,304 items and $v \in [32.44, 54.08]$, and category 5 has 3,376 items and $v \in [54.31, 100.21]$ (in dm^3). Furthermore, we calculated the distribution of item categories for each industry instance and applied k-clustering (see Figure 4.6b). Based on the analysis, we determined that the instances can be separated into four classes according to the percentage of items of each category they include. We, then, calculated the average percentages for each item category for each class (cluster) and report them in Table 4.2.

The instance generator uses the determined parameters to generate instances with different sizes. We provide the C++ code of the generator, using a default seed, at www.wanopt.uwaterloo.ca/projects/3dbppRealisticInstanceGen/ for use by the academic community. We use the generator to create 5 instances of size 50, 100, 150, 200, 500, 1000,

	Percentage of items				
Class	Category 1	Category 2	Category 3	Category 4	Category 5
Class 1	28.48%	58.75%	12.67%	0.1%	0%
Class 2	33.08%	32.36%	23.34%	7.94%	3.28%
Class 3	66.88%	24.75%	5.7%	2.6%	0.08%
Class 4	78.58%	13.16%	6.33%	1.78%	0.15%

Table 4.2: Percentage of each category of item in each instance class.

1500, and 2000 for each class, leading to 160 instances in total.

Finally, we applied Kolmogorov-Smirnov tests with the null hypothesis stating that the two sets of data follow the same distribution to verify the similarity of the generated instances to the industry ones. The Kolmogorov-Smirnov D values (supremums) related to the depth and height to width ratio, volume, and frequency of occurrence comparisons are 0.11, 0.03, 0.05, and 0.02, respectively. Moreover, the p-values corresponding to these tests are 0.07, 0.24, 0.17, and 0.15, respectively. According to the Kolmogorov-Smirnov test, the closer the value of D to zero, the more likely the samples are similar. Moreover, the null hypothesis that the two samples are similar cannot be rejected at the 0.05 significance level (p-values > 0.05). Therefore, we conclude that the generated instances sufficiently represent the industry data.

4.2 Results and Comparison on Generated Instances

In this section, we compare the performance of LCGA to the exact approach of Martello et al. (2007), referred to as Algorithm 864, on the generated instances. The purpose of the experiment is to demonstrate that LCGA is suited to tackle realistic problems with up to 2000 items. The code provided for Algorithm 864 is only able to solve instances with up to 100 items, due to a parameter they use for array sizes. We increased this parameter to accommodate larger problems. The results are reported in Table 4.3, where the class, number of items, average lower bound (L_2), average CPU time, average number of bins used, and average support percentage per item (Support %) are reported for LCGA and Algorithm 864, respectively. The support percentage refers to an item's base area that is supported by items underneath it. This metric is important in assessing bin stability, and is a requirement in practice. The results are averaged over 5 instances.

The results show that LCGA clearly outperforms Algorithm 864 in terms of number of bins, support and CPU time. LCGA requires less bins and achieves an overall reduction of about 10%. Algorithm 864 reaches the 1 hour time limit in 133 out of the 160 instances, while LCGA spends a maximum of 25 minutes. Please note that Algorithm 864 reaches the computational time limit in less than 5 instances in some of the class and instance size pairs (e.g., 4 out of 5 instances for Class 2, $|I| = 100$). The increase in percentage support is remarkable. LCGA provides 74.14% more vertical support than Algorithm 864. The latter is only able to provide the 70% industry minimum in only 27 out of the 160 instances, while LCGA achieves this in all 160. Considering that 70% support per item is the industry requirement, LCGA can be directly used for real-life palletization problems.

Finally, in Tables A.1-A.4 in Appendix A, we provide detailed computational results for all instances. The tables report the class, number of items, instance number, lower bound, the number of bins used, CPU time, average support percentage per item, and the percentage of items with at least 70% support for each instance.

4.3 Extending the Instance Generator

There are additional item features that characterize practical distributor's pallet loading problems. To include these, the instance generator is extended by adding five practical

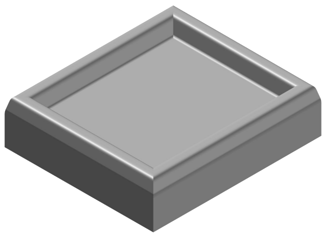
			LCGA					Algorithm 864				
				# Bins					# Bins			
Class	Items	L_2	CPU (s)	Min	Max	Avg	Support (%)	CPU (s)	Min	Max	Avg	Support (%)
1	50	1	0.24	1	1	**1**	86.70	0.10	1	1	**1**	34.13
	100	1	1.38	1	1	**1**	87.77	3600	2	2	**2**	41.14
	150	2	15.50	2	2	**2**	89.90	3600	2	2	**2**	36.37
	200	2	37.46	2	2	**2**	91.40	3600	3	3	3	41.14
	500	4	120.31	5	5	**5**	91.98	3600	5	5	**5**	50.78
	1000	7.2	218.05	8	9	**8.8**	92.67	3600	10	10	10	63.52
	1500	11	439.27	12	13	**12.4**	93.26	3600	13	14	13.6	68.49
	2000	14	888.66	16	16	**16**	92.97	3600	17	18	17.2	73.95
2	50	1	0.19	1	1	**1**	89.50	0.51	1	1	**1**	34.61
	100	1	1.60	1	1	**1**	86.84	2880	1	2	1.8	48.57
	150	1.2	15.44	2	2	**2**	92.00	3600	2	2	**2**	38.19
	200	2	55.29	2	2	**2**	91.61	3600	2	3	2.8	48.82
	500	4	144.37	5	5	**5**	91.76	3600	5	5	**5**	60.74
	1000	7	244.66	9	9	**9**	92.43	3600	9	10	9.2	72.04
	1500	11	585.42	12	12	**12**	92.97	3600	13	13	13	78.20
	2000	14	1035.09	16	16	**16**	92.86	3600	17	18	17.4	73.00
3	50	1	0.17	1	1	**1**	88.89	0.18	1	1	**1**	31.78
	100	1	1.50	1	1	**1**	87.46	2880	1	2	1.8	45.35
	150	1	21.00	1	2	**1.8**	89.35	3600	2	2	2	30.36
	200	2	70.49	2	2	**2**	91.32	3600	2	2	**2**	33.32
	500	3	344.60	4	4	**4**	91.76	3600	5	5	5	46.10
	1000	6	424.41	7	7	**7**	92.12	3600	8	8	8	58.69
	1500	9	884.64	10	11	**10.6**	92.33	3600	11	12	11.6	61.79
	2000	12	1208.16	14	14	**14**	92.50	3600	14	15	14.8	74.91
4	50	1	0.13	1	1	**1**	88.80	0.67	1	1	**1**	32.72
	100	1	1.77	1	1	**1**	87.88	720	1	2	1.2	38.93
	150	1	21.67	1	2	**1.2**	89.70	2880	2	2	2	35.37
	200	1.8	75.14	2	2	**2**	90.33	3600	2	2	**2**	50.11
	500	3	252.94	4	4	**4**	91.46	3600	4	5	4.2	47.83
	1000	6	536.38	7	7	**7**	91.17	3600	7	8	7.6	66.16
	1500	8	944.40	10	10	**10**	91.71	3600	10	11	10.4	83.65
	2000	11	1462.29	12	13	**12.8**	92.20	3600	14	14	14	67.82
Average		4.73	314.14			**5.52**	90.8	2992.55			6.08	52.14

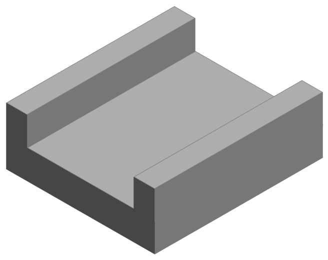
Table 4.3: Comparison of Algorithm 864 and LCGA.

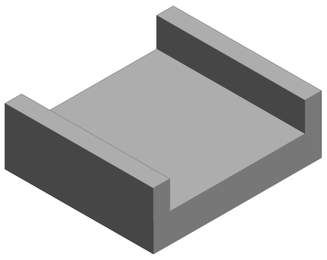
features. To this end, we further analyzed the industry data and obtained probability distribution functions for each feature. The industry data that we used spans 153 separate instances and includes 60,663 items, 6,387 of which are distinct. Finally, we validated the generated instances, comparing them to the industry data.

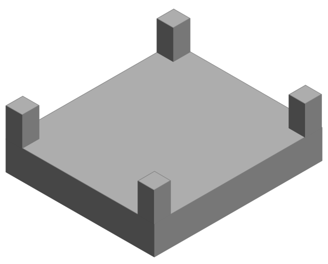
There are five practical item features that are considered in most industry applications: weight, load capacity, support surface shape, edge reduction (along width and depth dimensions), and sequence number. The weight of an item is straightforward, and is given in grams. Load capacity is defined as the maximum weight that can be carried per unit surface area of an item (in kg/m^2). To calculate the total maximum weight that an item can carry (i.e., load bearing limit), we multiple the load capacity by the surface area of the item. Items can have six different support surface shapes: the standard full support (i.e., the item is a rectangular box), along all edges, along only short or long edges, at corners, and reduced support. Figure 4.7 shows all support surface shapes except the standard full support case. In the industry data we use, edge and corner support surfaces have a thickness of 27 mm. Furthermore, the surface of a full support item may be reduced in width and depth dimensions (i.e., the item may not provide support along its edges). This is called width and depth edge reduction and its value may vary from item to item. Lastly, each item has a sequence number that depicts its arrival order at the packing area.

For the basic instance generator, we determined that generating item volumes and width, depth, and height ratios was a better approach instead of generating each dimension separately. This is done to capture the correlation between item dimensions and volumes. We follow a similar approach for the practical features. To determine how each feature is related to the others, we applied correlation analysis to the data. Table 4.4 shows the correlation matrix. Note that we added the item densities (weight/volume) into the analysis, anticipating that it would also be correlated.


(a) All edges support.


(b) Long edges support.


(c) Short edges support.


(d) Corner support.

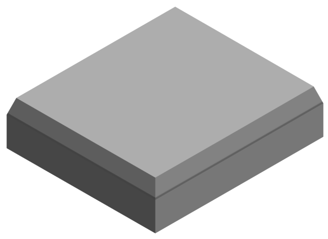
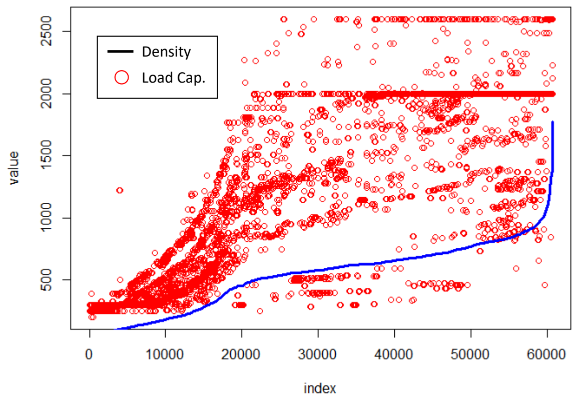

(e) Edge reduction.

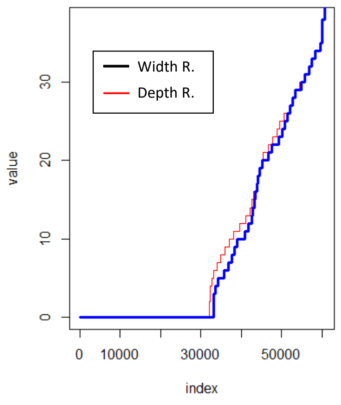
Figure 4.7: Item shapes.

	Volume	Weight	Density	Load Capacity	Width R.	Depth R.
Volume	1	0.51	-0.05	-0.07	-0.11	-0.1
Weight	-	1	0.72	0.47	0.37	0.37
Density	-	-	1	**0.75**	0.28	0.25
Load Capacity	-	-	-	1	0	-0.05
Width Reduction	-	-	-	-	1	**0.98**
Depth Reduction	-	-	-	-	-	1

Table 4.4: Correlation between the five practical item features.



(a) Density and load capacity.



(b) Width and depth edge reduction.

Figure 4.8: Relationship between item features.

According to Table 4.4, it is clear that there is a strong correlation between an item's density and its load capacity, as well as between the width and depth edge reduction values. Figures 4.8a and 4.8b further demonstrate these correlations. The remaining features are more or less independent, and can be analyzed independently.

Since the density of an item encompasses most its attributes (e.g., weight, volume, and load capacity), we focused on analyzing it first. Figure 4.9 depicts the sorted density values. Visually, it is clear that the density values can be separated into two distribution curves. The two curves have densities in $[31.76, 434.64)$ and $[434.64, 1771.11]$ g/dm^3, and include 31.32% and 68.68% of the items, respectively. We, then, used the $fitdistrplus$ package to determine the best probability distribution functions to fit these curves and report them in Table 4.5 along with the p-values. Both distribution functions have $p > 0.05$, validating their accuracy.

After item densities are determined, we can analyze weight and load capacity further.

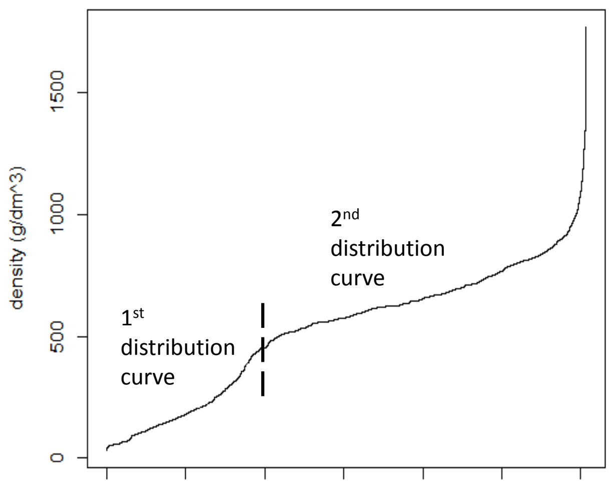


Figure 4.9: Separation of item density into curves.

Curve	Fit	Probability of appearing (%)	p-value
Distribution Curve 1	Gamma(3.211, 58.824)	31.32	0.08
Distribution Curve 2	Lognormal(6.502,0.208)	68.68	0.06

Table 4.5: Distribution functions for item density.

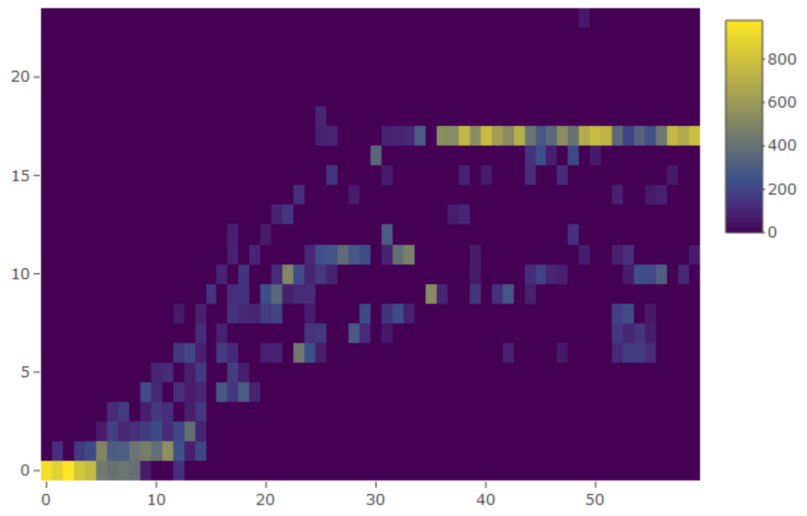
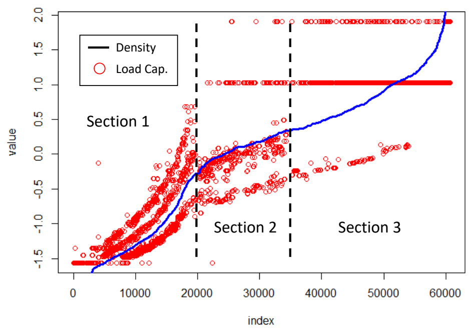


Figure 4.10: Load capacity heat map.

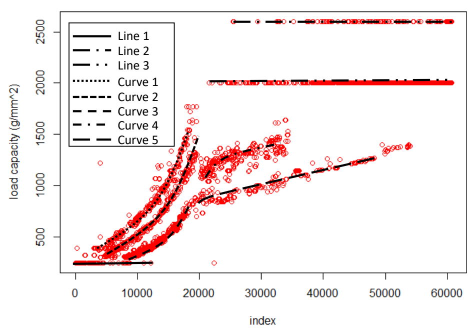
The connection between the density and the weight of an item is straightforward, especially considering that the basic instance generator already includes item volumes. The weight, in grams, is determined by multiplying volume by density. Note that *density* is given in g/dm^3 and volume in dm^3.

To better understand the relationship between load capacity and density, we cleaned the data in Figure 4.8a. We used a heat map (Figure 4.10) to determine where the data is concentrated. We identified the sections that have 10% or less concentration compared to the maximum (1000 data points) according to the heat map and removed them (see Figure 4.11a). Note that the plot has scaled data so that the relationship is further underlined. This cleaned plot includes 84.78% of the original data points, which gives a good representation of the overall data set.

Observing the plot in Figure 4.11b it is clear that the majority of the load capacity values follow several distinct curves and lines. Therefore, we separated the load ca-



(a) Load capacity cleaned and scaled.



(b) Load capacity trend lines.

Figure 4.11: Detailed load capacity charts.

pacity data following these trend lines and fitted curves/lines to each. In addition, we found that these trend lines can be separated into three sections, with respect to density. This way, we can calculate the probability that the load capacity of an item belonging to a certain trend line, based on its density. The sections 1, 2, and 3 have densities in $[31.76, 434.64)$,$[434.64, 617.56)$, and $[617.56, 1771.11]$, respectively. Table 4.6 shows the probability distribution function, the likelihood of an item belonging to each trend line, and the p-value for each section. Since none of the distribution functions have $p \leq 0.05$, the null hypothesis cannot be rejected, validating the accuracy of the fits. Figures C.1a-C.1e show the cumulative distribution function plots of the best fits.

Using these distributions and probabilities, the connection between the density and the other features are incorporated into the instance generator as follows:

Step 1. Generate the volume (dm^3) and dimensions (mm) of an item using the basic instance generator.

Step 2. Determine the density (g/dm^3) of the item following Table 4.5.

Trend line	Load capacity (kg/m^2)	Probability of appearing (%)	p-value
Section 1			
Line 1	250	16.84	
Curve 1	Gamma(5.698, 142.857)	24.33	0.31
Curve 2	Gamma(4.589,142.857)	27.48	0.52
Curve 3	Gamma(12.324,37.037)	31.35	0.19
Section 2			
Line 3	2000	5	
Curve 4	Gamma(127.49, 100)	65.13	0.14
Curve 5	Lognormal(6.944,0.143)	29.27	0.22
Section 3			
Line 2	2500	4.85	
Line 3	2000	81.02	
Curve 5	Lognormal(6.944,0.143)	14.13	0.22

Table 4.6: Load capacity curve fits.

Step 3. Calculate the weight (g) using the volume and density.

Step 4. Determine which load capacity section the item belongs to, with respect to density.

Step 5. Calculate the load capacity (kg/m^2) of the item following Table 4.6.

Step 6. Calculate the maximum weight capacity (g) by multiplying load capacity by width and depth.

The other correlated attributes are the width and depth edge reduction values. To analyze the relationship, we plotted them against each other, along with the width-depth difference (see Figures 4.8b and 4.12). According to the data, 52.97% of the items have width and depth edge reduction of 0 mm. The remaining 47.03% of the items have a width edge reduction that follows a Weibull distribution with shape 2.289 and rate 22.802. The difference between width and depth edge reduction for these items follow a Normal distribution with mean 1.229 and standard deviation 3.264. The p-values corresponding to

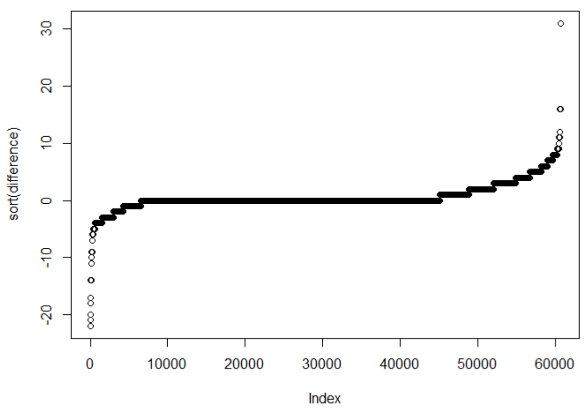


Figure 4.12: Width and depth edge reduction difference.

the distributions are 0.39 and 0.34, respectively. Using these two distributions, the width and depth edge reduction values are determined.

The remaining item features are not correlated and can be analyzed independently. For item shapes, we calculated the likelihood that an item having a certain shape and report them in Table 4.7. Finally, planogram sequence values are completely randomized, as there is no inherent connection between an item and its arrival order at the packing area. Further discussion on these features is given in Chapter 5 where we describe each practical constraint in detail.

Shape	Probability of appearing (%)
Full	87.19
Long Edge Support	0.06
Short Edge Support	0.05
All Edge Support	12.25
Corner Support	0.45

Table 4.7: Item shape probabilities.

The generator is used to create 5 instances for each class and size combination ($(|I| \in$

$\{50, 100, 150, 200, 500, 1000, 1500, 2000\}$), leading to 160 instances and 110,000 items in total. The generated instances are given in this format: width (mm), depth (mm), height (mm), weight (g), load capacity (kg/m^2), width edge reduction (mm), depth edge reduction (mm), support surface type, frequency of occurrence, and planogram sequence, respectively. A C++ implementation of the instance generator is provided at http://www.wanopt.uwaterloo.ca/ projects/3dbppPracticalInstanceGen/.

Finally, to further validate the generated data, we calculated the correlation matrix and report it in Table 4.8. The correlation values are very similar to the industry data. Furthermore, all of the reported p-values for the determined distribution functions are greater than 0.05. Based on these, the generated data seems to successfully represent the industry data.

	Volume	Weight	Density	Load Capacity	Width R.	Depth R.
Volume	1	0.52	-0.01	-0.01	0	0
Weight	-	1	0.8	0.67	0.03	0.03
Density	-	-	1	**0.84**	0.03	0.03
Load Capacity	-	-	-	1	0.03	0.03
Width Reduction	-	-	-	-	1	**0.85**
Depth Reduction	-	-	-	-	-	1

Table 4.8: Correlation matrix of the generated data.

4.4 Conclusion

The benchmark instances that the literature offers were found to be unrealistic, thus making it impossible to assess the true efficacy of any proposed solution methodology for 3DBPP and DPLP. To remedy this, we proposed a new instance generator whose parameters and distributions are determined based on industry data.

First, we used industry data to analyze item dimensions and volumes. We applied statistical techniques such as clustering and curve fitting to determine instance classes and distribution functions. We, then, generated 160 instances with different sizes and item types, and conducted extensive numerical testing to compare LCGA against Algorithm 864 of Martello et al. (2007). In the experiments, LCGA found better solutions on all instances, both in terms of number of bins used and bin stability, within significantly shorter computational times.

Finally, we extended the instance generator by further analyzing the item features, adding practical concepts such as weight, load capacity, different item shapes and support surfaces, and planogram sequences. To this end, we conducted correlation analysis as well as clustering, data cleaning, and curve fitting. As a result, we determined distribution functions and parameters for the new features and added them to the basic instance generator. Based on our validation tests, the generator is able to create instances that accurately capture the practical features.

Chapter 5

The Distributor's Pallet Loading Problem

As Zhao et al. (2016) points out, there is no solution methodology that considers all practical constraints in the distributor's pallet loading problem. In this chapter, we propose what we believe is the first approach that accounts for all practical constraints. It is based on a layer generation approach and is able to solve large and realistic instances within the industry time limit of 2 minutes. We first describe the practical constraints set out by the palletization industry, and then propose solution techniques for each.

5.1 Practical Constraints

The palletization industry requires four critical constraints for the DPLP: vertical item support, load bearing, planogram sequencing, and bin weight limit. In this section, we will explain each practical constraint in detail.

Vertical support: To build stable pallets, it is essential to provide sufficient vertical support for each item; i.e., sufficient bottom support to ensure the item is stable. Industry guidelines use two criteria: (1.) 70% of the bottom surface of an item or (2.) all four corners of an item are supported. Both guarantee that the center of gravity of the item has enough support. Vertical support should be considered both in terms of item dimensions and shape.

Load bearing: It is unreasonable to place heavy items on top of fragile ones (e.g., a case of beverages on top of a box of potato chips). The weight distribution of the items throughout a pallet should be managed extremely carefully so that no item is in risk of breaking or giving way, resulting in lost stability. Additionally, items should be placed with respect to their load bearing capabilities in a way that more of the volume of a bin is leveraged. For example, if an extremely fragile item is placed at the bottom of a pallet, not many items can be placed on top.

Planogram sequencing: In large automated warehouses, there is generally a buffer area prior to palletization where items for a specific order are gathered. Since the size of the buffer is limited, it is sometimes impossible to wait for all items in an order to arrive in order to start the palletization process. This may result in pallets being planned considering only a partial order. This is taken into account using planogram sequences.

Weight limit: This refers to limiting the weight of pallet for ease of handling.

5.1.1 Vertical Support

After a layer is formed, the areas it can provide and it requires support are known. The methodology that we use to tackle vertical support takes advantage of this information and places a layer on top of another, only if vertical support is guaranteed for all items.

When two layers are considered for placement, first, items in the top layer are spaced out in a way that ensures there is sufficient support received from the bottom layer. To this end, we use a mathematical model with an objective to minimize the width-depth overlap between the items in the bottom and top layers, while adding a constraint that guarantees support. This corresponds to solving a 2D placement problem where the values for all the relative positioning variables are known.

The two-dimensional overlap between items i and j is visually described in Figure 5.1.1. To calculate the overlapping area between the items, we determine the coordinates of the overlapping area, c^x_{ij}, c^y_{ij}, $c^x_{ij} + w_{ij}$, $c^y_{ij} + d_{ij}$ using the following equations:

$$c^x_{ij1} = \max\{c^x_i, c^x_j\}, \tag{5.1}$$

$$c^y_{ij1} = \max\{c^y_i, c^y_j\}, \tag{5.2}$$

$$c^x_{ij2} = \min\{c^x_i + w_i, c^x_j + w_j\}, \tag{5.3}$$

$$c^y_{ij2} = \min\{c^y_i + d_i, c^y_j + d_j\}. \tag{5.4}$$

Let I be the set of items, w_i and d_i be the width and depth of an item, and W and D be the width and depth of a pallet, respectively. Let $\overline{z}^x_{ij}$ and $\overline{z}^y_{ij}$ take value 1 if item i precedes item j in the width and depth dimensions, respectively. Note that these are all parameters obtained from the information included in a layer. Let c^x_i and c^y_i be the width and depth coordinates of the front bottom left corner of item i. Using $s_{ij1} = c^x_{ij2} - c^x_{ij1}$ and

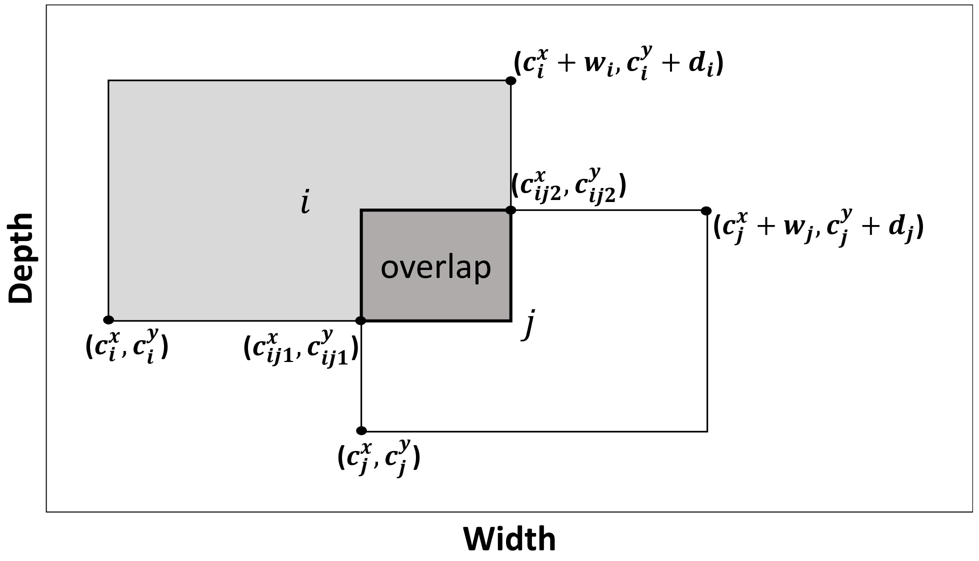


Figure 5.1: Two-dimensional overlap between two items.

$s_{ij2} = c_{ij2}^y - c_{ij1}^y$, the spacing model is:

$$[SM] : \min_{s,c} \sum_{i \in I_t, j \in I_b} s_{ij1} + s_{ij2} \tag{5.5}$$

$$\text{s.t.} \quad \sum_{j \in \mathcal{I}_b} s_{ij1} s_{ij2} \geq 0.7 w_i d_i \qquad i \in \mathcal{I}_t \tag{5.6}$$

$$c_i^x + w_i \leq c_j^x + W(1 - \overline{z}_{ij}^x) \qquad i \neq j : i, j \in \mathcal{I}_t \tag{5.7}$$

$$c_i^y + d_i \leq c_j^y + D(1 - \overline{z}_{ij}^y) \qquad i \neq j : i, j \in \mathcal{I}_t \tag{5.8}$$

$$0 \leq c_i^x \leq W - w_i \qquad i \in \mathcal{I}_t \tag{5.9}$$

$$0 \leq c_i^y \leq D - d_i \qquad i \in \mathcal{I}_t \tag{5.10}$$

where $\mathcal{I}_t$ and $\mathcal{I}_b$ are the set of items in the top and bottom layers, respectively. The objective of $[SM]$ (5.5) is to minimize the width-depth overlap between items in the bottom and top layers. Constraints (5.6) ensure that there is sufficient support for every item in the top layer. Since support is guaranteed, we seek to minimize the overlap between items so that they are as spaced out as possible, which results in more support surface for the items to be placed later on. Constraints (5.7) and (5.8) ensure the items in the top layer do not overlap in the width and depth dimensions. Constraints (5.9) and (5.10) ensure that the items reside within the boundaries of a pallet.

The objective (5.5) is nonlinear due to the multiplication and min/max operators. It can be partially linearized by defining variables x_{ij}^{max}, x_{ij}^{min}, y_{ij}^{max}, and y_{ij}^{min} for the nonlinear terms in (5.5), respectively. $[SM]$ becomes:

$$[SML] : \min_{s,c,x,y} \sum_{i \in I_t, j \in I_b} s_{ij1} + s_{ij2} \tag{5.11}$$

$$\text{s.t.} \quad (5.6) - (5.10)$$

$$x_{ij}^{max} \leq c_i^x + w_i \qquad i \in \mathcal{I}_t, j \in \mathcal{I}_b \tag{5.12}$$

$$x_{ij}^{max} \leq \bar{c}_j^x + w_j \qquad i \in \mathcal{I}_t, j \in \mathcal{I}_b \tag{5.13}$$

$$x_{ij}^{min} \geq c_i^x \qquad i \in \mathcal{I}_t, j \in \mathcal{I}_b \tag{5.14}$$

$$x_{ij}^{min} \geq \bar{c}_j^x \qquad i \in \mathcal{I}_t, j \in \mathcal{I}_b \tag{5.15}$$

$$y_{ij}^{max} \leq c_i^y + d_i \qquad i \in \mathcal{I}_t, j \in \mathcal{I}_b \tag{5.16}$$

$$y_{ij}^{max} \leq \bar{c}_j^y + d_j \qquad i \in \mathcal{I}_t, j \in \mathcal{I}_b \tag{5.17}$$

$$y_{ij}^{min} \geq c_i^y \qquad i \in \mathcal{I}_t, j \in \mathcal{I}_b \tag{5.18}$$

$$y_{ij}^{min} \geq \bar{c}_j^y \qquad i \in \mathcal{I}_t, j \in \mathcal{I}_b \tag{5.19}$$

$$s_{ij1} \geq x_{ij}^{max} - x_{ij}^{min} \qquad i \in \mathcal{I}_t, j \in \mathcal{I}_b \tag{5.20}$$

$$s_{ij2} \geq y_{ij}^{max} - y_{ij}^{min} \qquad i \in \mathcal{I}_t, j \in \mathcal{I}_b \tag{5.21}$$

$$s_{ij1}, s_{ij2}, x_{ij}^{min}, x_{ij}^{max}, y_{ij}^{min}, y_{ij}^{max} \geq 0 \qquad i \in \mathcal{I}_t, j \in \mathcal{I}_b \tag{5.22}$$

Note that (5.6) is still nonlinear. $[SML]$ can be modeled as an SOCP by noting that:

$$\begin{aligned} (s_{ij1} - s_{ij2})^2 &= s_{ij1}^2 + s_{ij2}^2 - 2s_{ij1}s_{ij2}, \\ \iff s_{ij1}s_{ij2} &= 0.25[(s_{ij1} + s_{ij2})^2 - (s_{ij1} - s_{ij2})^2], \end{aligned} \tag{5.23}$$

substituting $s_{ij1}s_{ij2}$ in (5.6) using (5.23) we obtain:

$$\begin{aligned} &\sum_{j\in\mathcal{I}_b}[(s_{ij1} + s_{ij2})^2 - (s_{ij1} - s_{ij2})^2] \geq 4 \times 0.7w_id_i && i \in \mathcal{I}_t, \\ \iff &\sum_{j\in\mathcal{I}_b}(s_{ij1} + s_{ij2})^2 \geq \sum_{j\in\mathcal{I}_b}(s_{ij1} - s_{ij2})^2 + 2.8w_id_i && i \in \mathcal{I}_t, \\ \iff &\sum_{j\in\mathcal{I}_b}(s_{ij1} + s_{ij2}) \geq \sqrt{\sum_{j\in\mathcal{I}_b}(s_{ij1} - s_{ij2})^2 + 2.8w_id_i} && i \in \mathcal{I}_t, \\ \iff &\sum_{j\in\mathcal{I}_b}(s_{ij1} + s_{ij2}) \geq \left\| \begin{bmatrix} \sum_{j\in\mathcal{I}_b}(s_{ij1} - s_{ij2}) \\ 1.4\sqrt{w_id_i} \end{bmatrix} \right\| && i \in \mathcal{I}_t. \end{aligned} \tag{5.24}$$

Replacing (5.6) with (5.24) in $[SML]$ and modifying the objective, we obtain:

$$\begin{aligned} [SML - SOCP] : \min_{s,c,x,y} &\sum_{i\in\mathcal{I}_t, j\in\mathcal{I}_b} (s_{ij1} + s_{ij2}) \\ \text{s.t.} \quad &(5.7) - (5.10), (5.12) - (5.22), (5.24) \end{aligned}$$

$[SML - SOCP]$ is easily solvable using a commercial solver such as CPLEX.

Using the spacing model, we tackle the vertical support using Algorithm 8. After layer placement is done, any remaining items are placed using an extreme point heuristic (Crainic et al., 2008) that favors lower positions in the bin, with respect to height, width, and depth, respectively. The vertical support tests are applied for any extreme point and item pair, and only feasible placements are used. It is important to note that we remove layers with non-homogeneous height (i.e., layers generated using the complete set of items $\mathcal{CI}$) from the set $\mathcal{L}$ before we start the solution approach for the practical constraints, since these layers cannot provide proper vertical support.

Algorithm 8 Vertical Support Algorithm

```
1: procedure INITIALIZATION
2:     $\mathcal{L}$ = Set of generated layers.
3: end procedure
4: procedure TACKLE–VERTICAL–SUPPORT($\mathcal{L}$)
5:     Sort layers in $\mathcal{L}$ in descending order by density.
6:     Open an empty bin $b$ and set $\mathcal{B} \leftarrow \mathcal{B} \cup b$.
7:     for each layer $l \in \mathcal{L}$ do
8:         for each bin $b \in \mathcal{B}$ do
9:             if $b$ is empty then
10:                Place $l$ in $b$.
11:            else
12:                Space $l$ using [$SML$].
13:                if $h_b + h_l \leq H$ AND every item $i \in \mathcal{I}_l$ is supported then
14:                    Place $l$ in $b$.
15:                end if
16:            end if
17:        end for
18:        for each remaining layer $k \in \mathcal{L}$ do
19:            for each item $i \in \mathcal{I}_k$ do
20:                if $i \in \mathcal{I}_l$ then
21:                    $\mathcal{L} \leftarrow \mathcal{L} \backslash k$
22:                end if
23:            end for
24:        end for
25:    end for
26: end procedure
```

5.1.2 Load Bearing

To account for load bearing, we first need to ensure that each layer has a nearly homogeneous distribution of load capacity. This means that fragile and hard items should not be parts of the same layer. This way, layers, being blocks of items with similar load bearing limits, can be placed on top of each other in decreasing order of overall carrying capacity. To this end, we further divide the item groups $\mathcal{IG}$. We first calculate the average load bearing limit of items, denoted by b_g, for group $g \in \mathcal{IG}$. For each group g, items with load bearing limits outside of b_g +/- a certain percentage are removed. Algorithm 9 describes this process more formally.

Algorithm 9 Practical Item Groupung Algorithm

```
 1: procedure INITIALIZATION
 2:     Create item groups $\mathcal{IG}$ following Algorithm 7.
 3: end procedure
 4: procedure GROUPITEMSEXTENDED($\mathcal{CI}$)
 5:     for each item group $g \in \mathcal{IG}$ do
 6:         Calculate $b_g$.
 7:     end for
 8:     for each item group $g \in \mathcal{IG}$ do
 9:         for each item $i \in g$ do
10:             if $b_i < 0.7b_g$ OR $b_i > 1.3b_g$ then
11:                 Set $g \leftarrow g \setminus i$.
12:             end if
13:         end for
14:     end for
15: end procedure
```

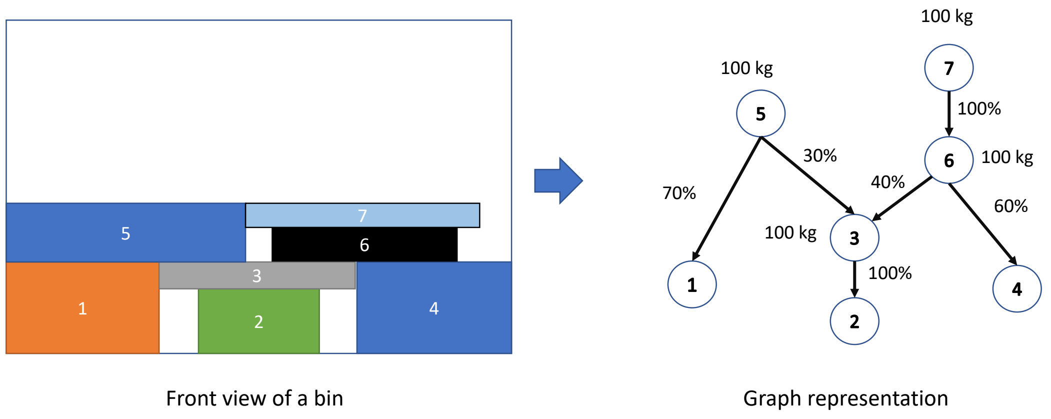


Figure 5.2: Graph representation of the items in a bin.

To construct pallets, we previously sorted the layers $\mathcal{L}$ based on their densities. To account for load bearing, a subset of the sorted layers are sorted according to their load bearing. In other words, layers are first sorted based on density, then groups of adjacent layers are sorted based on load bearing.

When either layers or individual items are stacked on top of another, the weight carried by the items beneath should be updated to determine if any item is carrying more than its load bearing limit. Since the placement of a layer or an item needs to be tested each time a layer is added, the updates should be done in a fast manner. To this end, we designed a level-based graph representation of the items in a pallet, where layers are levels.

An example of the graph representation is given in Figure 5.2, where every node i represents an item, and an arc (i, j) exists between the nodes i and j if and only if item i is supported by item j. Every item has a weight m_i and a load bearing limit b_i, and every arc has a percentage value p_{ij} that depicts how much of the surface of item i is supported by item j. Note that for every item i that is not at the bottom of a bin, $\sum_j p_{ij} = 100\%$. This graph lets us quickly traverse through every item that may be affected by the placement of a new item, while not considering the rest of the bin. For example, if an item is placed on top of item 5, only the total weights carried by items 1, 2, 3, and 5 are updated.

When the placement of an item is tested, we try adding a node at a higher level than the items that are supposed to support it. We, then, connect the node of the tested item to the supporting items with arcs and calculate the support percentages. Afterwards, we update the total weight carried by each node in the graph using a breadth-first search algorithm, starting from the tested item. If no node carries a higher weight than its load bearing limit, the placement is approved. Placement of a layer is done in the same way, with the addition of the load bearing test being done for every item in that layer.

5.1.3 Planogram Sequencing and Pallet Weight Limit

To account for planogram sequencing, subsets of n items are used at a time. n is the maximum number of items allowed in the buffer zone. In this work, we consider $n = 400$. Details are given in Algorithm 10. Finally, we satisfy the maximum pallet weight limit by checking the pallet weight after every item or layer placement.

Algorithm 10 Planogram sequencing

```
 1: procedure INITIALIZATION
 2:     Set $\mathcal{I}, \mathcal{CI}, \mathcal{CL} = \varnothing$.
 3: end procedure
 4: procedure plnSEQUENCE
 5:     while $|\mathcal{I}| > 0$ do
 6:         Sort items in $\mathcal{I}$ by their planogram sequence.
 7:         Take the first $n$ items from $\mathcal{I}$ into the list $\mathcal{CI}$.
 8:         Use column generation to generate layers $\mathcal{CL}$ for items $\mathcal{CI}$.
 9:         Construct a bin using $\mathcal{CL}$ and $\mathcal{CI}$, following the general solution approach and other
    practical constraints.
10:         Remove placed items from $\mathcal{CI}$.
11:         if $|\mathcal{CI}| > 0$ then
12:             Reintroduce $\mathcal{CI}$ into $\mathcal{I}$.
13:         end if
14:     end while
15: end procedure
```

5.2 Computational Experiments

We conducted extensive computational experiments to test the effectiveness of the proposed solution methodology. The testing is done using a workstation with Intel i7-5500 running at 2.4 GHz, 16 GB memory, and Microsoft Windows 10 OS. The algorithms are coded in C++ on Microsoft Visual Studio 2015 IDE, and the optimization models are solved using CPLEX Concert technology with IBM ILOG CPLEX 12.6.1.

5.2.1 Results and Analysis

The pallets used for testing are of size 1240 × 840 × 2200 mm ($W \times D \times H$) and weight limit of 1.5 tonnes. We considered that two side by side items can provide a continuous support surface if their height difference is within 10 mm. For items with edge or corner support surface shapes, we considered 27 mm surface thickness through the width and depth dimensions for each edge or corner. Additionally, width and depth edge reduction values are applied on each side (i.e., two times the width reduction value is subtracted from the overall width and so on). Finally, we considered a buffer area size of 400 items.

Testing is done on 160 instances ranging from 50 to 2000 items. Average results over 5 instances for each class and instance size combination are reported in Table 5.1. The columns represent the class and the item size combination, the number of pallets used to pack the items, the total and per pallet CPU times, volume usage, and pack density. The volume usage is the used percentage of the pallet volume, while the pack density is the density of the constructed pallet.

According to Table 5.1, the average number of items per pallet is 124.72, which is close to the expectation of the industry. And the average time to plan a pallet is 63.67 seconds, which is well inside the industry requirement of two minutes. Finally, the average pack density is 78.68%, which is within the industry expectation. This, combined with homogeneous load bearing distribution throughout the layers and pallets, shows that the constructed pallets are highly stable. Note that every practical constraint is satisfied. Sample pallets are displayed in Figures 5.3-5.6. Detailed results for each class, item size, and instance combination are given in Appendix D.

		# Pallets				CPU per	Vol. Use (%)			Pack Dens. (%)		
Class	Items	Min	Max	Avg	CPU (s)	Pallet (s)	Min	Max	Avg	Min	Max	Avg
1	50	1	1	1	2.3	2.3	29.9	33.4	31.4	71.1	74.6	72.5
	100	1	2	1.2	4.3	4.2	32.2	62.2	53.1	77.2	82.5	78.9
	150	2	2	2	6.3	3.1	45.2	49.3	47	77.3	80.4	78.7
	200	2	3	2.2	16.1	7.1	41.4	62.4	57.4	79.5	82.8	81.2
	500	4	5	4.6	275.5	59.4	62.2	76.9	68.1	75.8	87.4	80.6
	1000	8	8	8	819.1	102.4	77.7	79.9	78.4	78.1	80	79
	1500	12	12	12	1310	109.2	77.5	78	77.8	77.8	79	78.5
	2000	15	16	15.6	2083.4	134.4	78.1	82.4	79.9	78.9	82.5	80.4
2	50	1	1	1	0.2	0.2	29.7	31.8	30.7	76.3	82	72.1
	100	1	2	1.4	4.2	3.4	31	61.6	49.1	71.4	82	73
	150	2	2	2	3.8	1.9	43.3	47.8	45.8	75.8	79.1	77.7
	200	2	2	2	14.4	7.2	59.6	63.6	61.2	74.4	82.4	81.7
	500	4	5	4.6	354	76.5	60.8	77.9	67.9	74	85.4	80.2
	1000	8	8	8	1227.7	153.5	76.6	77.6	76.9	76.7	77.8	77.1
	1500	12	12	12	1717.2	143.1	76.7	78.6	77.4	77.8	79.1	78.4
	2000	15	16	15.6	2398	153.6	77.5	82.6	79.5	78.4	82.7	80.2
3	50	1	1	1	1.3	1.3	23.4	28.8	25.8	78.8	85.5	81.9
	100	1	1	1	2.5	2.5	47.6	51.1	49.5	78.5	81.6	79.8
	150	2	2	2	8	4	35.8	38.4	37.1	72.6	78.6	75.6
	200	2	2	2	20.8	10.4	48.7	52.2	50.5	70.5	82.8	71.3
	500	4	4	4	469.4	117.4	61.1	63.5	62.1	72.8	83.3	74.7
	1000	6	7	6.8	1304	195.3	70.8	80.3	73.3	72.4	80.5	74.3
	1500	10	10	10	2539.3	253.9	73.6	75.4	74.5	74.2	76.5	75.5
	2000	14	15	14.6	1067.8	77.3	66.6	77.5	68.4	76	77.7	78.6
4	50	1	1	1	1.3	1.3	18.9	24	21.9	74.4	82.9	79.4
	100	1	2	1.2	1.5	1.4	23	46.1	40.6	73.9	79.5	76.6
	150	1	2	1.6	7.4	5.6	33.6	64.7	46.3	74.9	85.2	82.5
	200	2	2	2	17	8.5	45.5	49.8	47.3	77.3	85.6	78.2
	500	3	4	3.8	650.6	177.7	57.8	77.6	62.1	77.8	83.3	81.7
	1000	7	7	7	935.3	144.9	57	76.6	66.4	76.1	80.1	77.7
	1500	11	11	11	315.6	28.7	62.3	64.3	63.2	74.8	85.6	80.6
	2000	14	15	14.2	648.3	45.8	60.5	65.9	64.6	75.5	85.8	79
Average				5.5	569.6	63.7			57.3			78.7

Table 5.1: Results of the computational experiments.

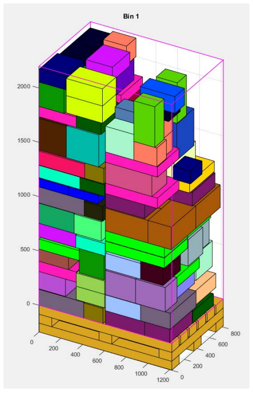


Figure 5.3: Sample pallet for Instance 2 of Class 3 and 150 items.

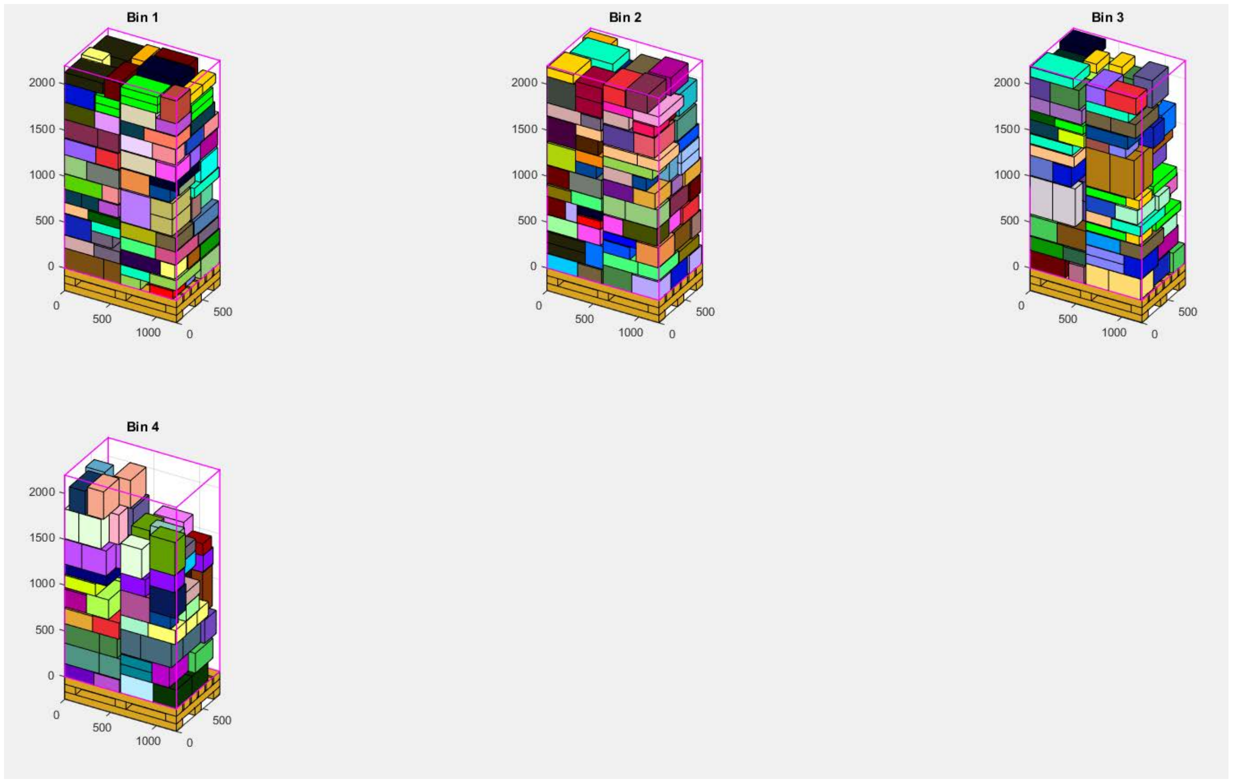


Figure 5.4: Sample pallets for Instance 3 of Class 2 and 500 items.

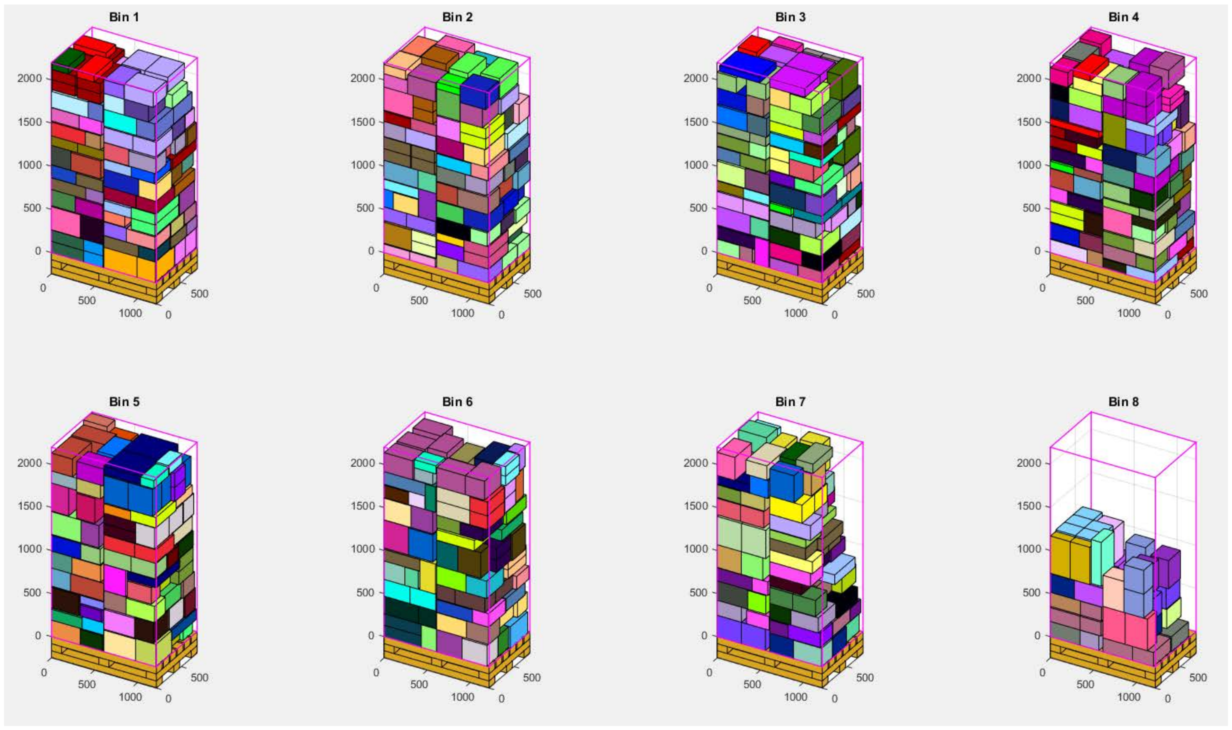


Figure 5.5: Sample pallets for Instance 1 of Class 1 and 1000 items.

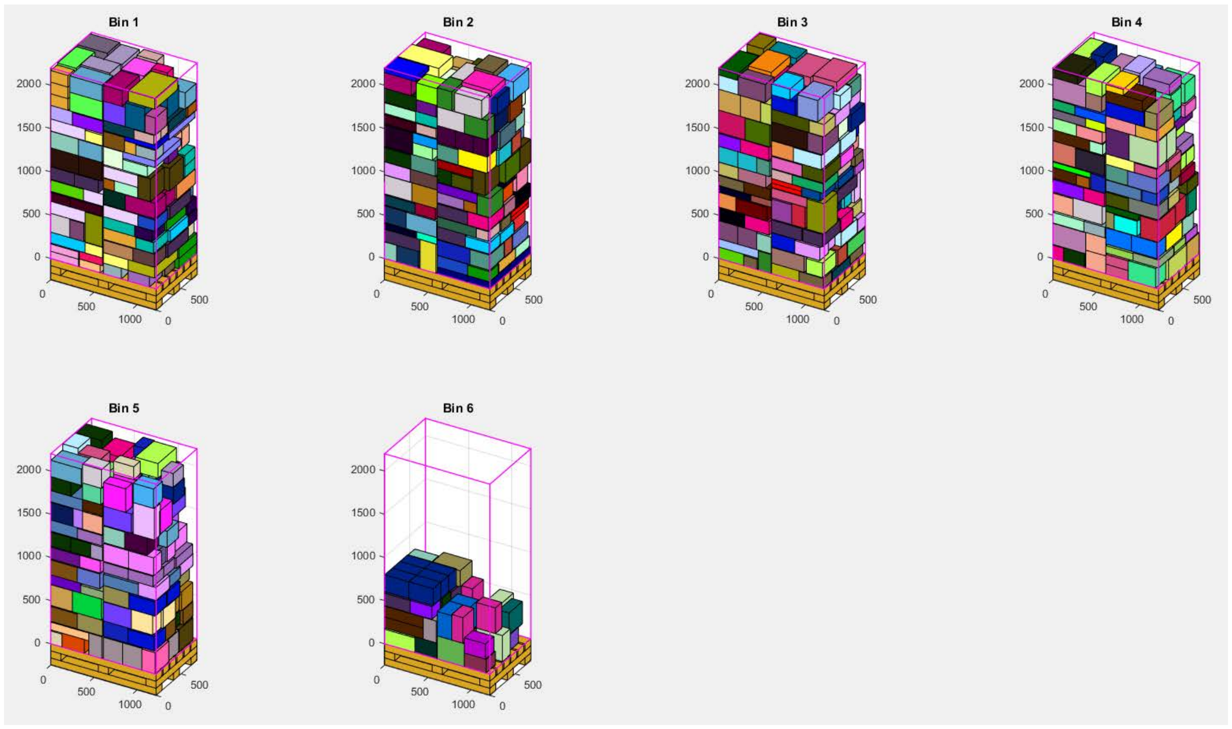


Figure 5.6: Sample pallets for Instance 2 of Class 4 and 1000 items.

5.3 Conclusion

Driven by the lack of an efficient solution methodology to solve the full DPLP, we propose an extension of LCGA to account for each practical requirement. The use of layers provided concrete advantages in incorporating sophisticated constraints such as load bearing and vertical support.

Vertical support, albeit being the most important practical constraint, is rarely considered in previous work. We proposed a novel nonlinear model for item spacing to ensure the 70% minimum industry support. We, then, reformulated as an SOCP. Additionally, we accommodated reduced support surfaces and different item shapes into the placement algorithm. To the best of our knowledge, this is the first work that considers such features.

We devised a placement algorithm to place layers based on their density and load bearing limits. As load bearing capabilities have to be updated throughout the construction of the pallet, we devised a new graph representation for fast weight distribution updates. Finally, we included planogram sequencing and pallet weight limits.

According to our extensive computational tests, the proposed approach provides remarkable solutions with low computational times and high stability.

Chapter 6

Conclusions

6.1 Summary of the thesis

Motivated by the lack of efficient solution methodologies to handle the three-dimensional bin packing and the distributor's pallet loading problems, we proposed a layer based column generation approach that is capable to handle industry-size instances and all related practical constraints in very short computational times. The approach is able to construct stable pallets where vertical support, load bearing, reduced support surfaces, different item shapes, planogram sequencing, and bin weight limit requirements are satisfied in about one minute of computational time.

In the first chapter, we introduced the two problems and underlined their importance. We, then, briefly discussed Lagrangean relaxation, which is one of the main building blocks of our proposed approach. The second chapter discussed the literature related to the three-dimensional packing problems, focusing on the proposed lower bounds, the heuristic and exact methods, and the practical constraints.

In Chapter 3, we described the proposed layer-based column generation algorithm in detail. We first defined the concept of a layer mathematically and proposed a novel mathematical model that uses them to solve the open dimension packing problem. Secondly, we reformulated the proposed model into a set covering formulation which we, then, solved using branch-and-price and column generation. We used exact and heuristic solution methodologies for the layer generation subproblem, and introduced post processing algorithms such as item replacement, item grouping, layer reorganization, and layer spacing, to improve the generated layers. We conducted extensive computational experiments on standard benchmark instances and compared to the state-of-the-art, and concluded that our approach provides superior solutions, both in terms of the number of bins used and stability, in better computational times.

We discussed the severe lack of realistic benchmark instances in Chapter 4 and proposed an instance generator that is trained using industry data. We conducted rigorous analyses on close to 200,000 items obtained from an industry partner to determine the parameters and distribution functions for basic item features such as dimensions, volume, and frequency of occurrence in an instance. We, then, extended the instance generator using practical item features such as weight, load capacity, different item shapes and support surfaces, and sequence numbers. Additionally, we provided the results that can be used for future comparison and benchmarking.

Finally, we proposed a solution methodology that tackles the entirety of the distributor's pallet loading problem in Chapter 5. First, we described the practical constraints and the related item features. Second, we considered these constraints individually and explained the different approaches we devised to account for them. These approaches range from a novel layer spacing model that is transformed into an SOCP, to a layer/item placement algorithm that uses a new graph representation to consider load bearing and weight distribution in a fast manner. We add these new methodologies to the previously described LCGA and conduct testing on the newly generated realistic instances. The results revealed the quality of the approach in providing compact pallets with high stability in an average of one minute, a solution time well under the industry requirement.

Our main intuition at the start of this work was that the use of layers would implicitly produce bins that are more stable and lead to problems that are easier to solve. We also hypothesized that the concept of layers would enable the straightforward inclusion of some important practical constraints. The results of the proposed layer-based solution approach confirms the intuition and the hypotheses.

In conclusion, the contributions are: a novel layer-based mathematical model for the 3DBPP, which enabled the solution of the problem using column generation and branch-and-price for the first time; the first realistic benchmark instance generator with practical item features to better assess the real-life performance of the methodologies proposed for the 3DBPP and the DPLP; and a novel nonlinear layer spacing model that is solved rapidly using SOCP. Overall, this is the first work to solve industry-level instances of the DPLP in less than the industry requirement of 2 minutes.

6.2 Future Research Directions

There are three possible directions to further expand the current research. Additional practical constraints can be added to the methodology, different three-dimensional packing problems can be tackled using the proposed methodology, and the layer generation step can be enhanced.

Some of the optional practical constraints, not considered in this thesis, include the use of sheet separators, the account for isle friendliness, the minimization of orderline separation, and the consideration of more item footprints. Sheet separators are usually placed in between layers to increase the support surface. This can be included trivially and would improve the general bin stability. Isle friendliness dictates that items that are most similar to one another should be packed as closely as possible (e.g., beverages, dairy products, etc.). This can be achieved by further limiting the item grouping to separate different product types. Similar to isle friendliness, it is preferred that the orderlines, a batch of orders of the same product, are grouped as closely as possible for ease of unpacking. This can be done with a combination of better item grouping and a modification to the pallet construction heuristic. The modification would entail sorting the layers by similar orderlines, as well as density and load bearing capacity. Finally, different item footprints (e.g., placing an item on its depth-height surface) can be handled by adding copies of the same item with different dimensions to the layer generation subproblem and forcing the model to select only one of them.

The proposed methodology can also be extended to solve different and more sophisticated packing problems. The container loading problem is one possibility, where other requirements such as the delivery order could also be included. Moreover, multiple container size packing problems can be solved by generating layers with different dimensions.

Finally, the dimension reduction that is achieved by considering two-dimensional layers in 3DBPP can be applied to the layer generation itself. This translates to considering one-dimensional strips that are then combined to form two-dimensional layers. This would lead into a nested layer and strip generation solution approach.

References

[1] Brenda S Baker, Edward G Coffman, Jr, and Ronald L Rivest. Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9(4):846–855, 1980.

[2] A. Bettinelli, A. Ceselli, and G. Righini. A branch-and-price algorithm for the two-dimensional level strip packing problem. *4OR*, 6(4):361–374, 2008.

[3] E.E. Bischoff and M.D. Marriott. A comparative evaluation of heuristics for container loading. *European Journal of Operational Research*, 44(2):267–276, 1990.

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

[5] A. Bortfeldt and D. Mack. A heuristic for the three-dimensional strip packing problem. *European Journal of Operational Research*, 183(3):1267–1279, 2007.

[6] 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.

[7] M.A. Boschetti. New lower bounds for the three-dimensional finite bin packing problem. *Discrete Applied Mathematics*, 140(1):241–258, 2004.

[8] S. Ceschia and A. Schaerf. Local search for a multi-drop multi-container loading problem. *Journal of Heuristics*, 19(2):275–294, 2013.

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

[10] F. Clautiaux, A. Nguyen, and J.P. Brenaut. Model for the challenge renault/ESICUP: version 1.3. http://challenge-esicup-2015.org/doc/modele_renault.pdf, 2015. Accessed: 2016-10-20.

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

[12] T.G. Crainic, G. Perboli, and R. Tadei. TS^2PACK: A two-level tabu search for the three-dimensional bin packing problem. *European Journal of Operational Research*, 195(3):744–760, 2009.

[13] Y.-P. Cui, Y. Zhou, and Y. Cui. Triple-solution approach for the strip packing problem with two-staged patterns. *Journal of Combinatorial Optimization*, 34(2):588–604, 2017.

[14] A.P. Davies and E.E. Bischoff. Weight distribution considerations in container loading. *European Journal of Operational Research*, 114(3):509 – 527, 1999.

[15] M.L. Delignette-Muller and C. Dutang. fitdistrplus: An *R* package for fitting distributions. *Journal of Statistical Software*, 64(4):1–34, 2015.

[16] M. Eley. Solving container loading problems by block arrangement. *European Journal of Operational Research*, 141(2):393 – 409, 2002.

[17] O. Faroe, D. Pisinger, and M. Zachariasen. Guided local search for the three-dimensional bin-packing problem. *Informs Journal on Computing*, 15(3):267–283, 2003.

[18] S.P. Fekete, J. Schepers, and J.C. Van der Veen. An exact algorithm for higher-dimensional orthogonal packing. *Operations Research*, 55(3):569–587, 2007.

[19] H. Gehring, K. Menschner, and M. Meyer. A computer-based heuristic for packing pooled shipment containers. *European Journal of Operational Research*, 44(2):277–288, 1990.

[20] J.A. George and D.F. Robinson. A heuristic for packing boxes into a container. *Computers & Operations Research*, 7(3):147–156, 1980.

[21] M. Hifi, I. Kacem, S. Nègre, and L. Wu. A linear programming approach for the three-dimensional bin-packing problem. *Electronic Notes in Discrete Mathematics*, 36:993–1000, 2010.

[22] L. Junqueira, R. Morabito, and D.S. Yamashita. Three-dimensional container loading models with cargo stability and load bearing constraints. *Computers & Operations Research*, 39(1):74–85, 2012.

[23] J. Jylänki. A thousand ways to pack the bin - A practical approach to two-dimensional rectangle bin packing. *retrieved from* http://clb.demon.fi/files/RectangleBinPack.pdf, 2010.

[24] A.T. Kearney. 25th annual state of logistics report, 2014.

[25] A. Lodi, S. Martello, and D. Vigo. Heuristic algorithms for the three-dimensional bin packing problem. *European Journal of Operational Research*, 141(2):410–420, 2002.

[26] A. Lodi, S. Martello, and D. Vigo. Models and bounds for two-dimensional level packing problems. *Journal of Combinatorial Optimization*, 8(3):363–379, 2004.

[27] S. Martello, D. Pisinger, and D. Vigo. Algorithm 864. *retrieved from* `http://www.diku.dk/~pisinger/codes.html`, 1998. Accessed: 2017-03-30.

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

[29] S. Martello, D. Pisinger, D. Vigo, E.D. 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.

[30] C. Paquay, M. Schyns, and S. Limbourg. A mixed integer programming formulation for the three-dimensional bin packing problem deriving from an air cargo application. *International Transactions in Operational Research*, 23(1-2):187–213, 2016.

[31] C. Paquay, S. Limbourg, M. Schyns, and J.F. Oliveira. Mip-based constructive heuristics for the three-dimensional bin packing problem with transportation constraints. *International Journal of Production Research*, 0(0):1–12, 2017.

[32] F. Parreño, R. Alvarez-Valdés, J.F. Oliveira, and J.M. Tamarit. A hybrid GRASP/VND algorithm for two-and three-dimensional bin packing. *Annals of Operations Research*, 179(1):203–220, 2010.

[33] D. Pisinger. Heuristics for the container loading problem. *European Journal of Operational Research*, 141(2):382–392, 2002.

[34] A.G. Ramos, E. Silva, and J.F. Oliveira. A new load balance methodology for container

loading problem in road transportation. *European Journal of Operational Research*, 266(3):1140 – 1152, 2018.

[35] D.M. Ryan and B.A. Foster. An integer programming approach to scheduling. *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*, pages 269–280, 1981.

[36] T.A.M. Toffolo, E. Esprit, T. Wauters, and G.V. Berghe. A two-dimensional heuristic decomposition approach to a three-dimensional multiple container loading problem. *European Journal of Operational Research*, 257(2):526–538, 2017.

[37] A. Trivella and D. Pisinger. The load-balanced multi-dimensional bin-packing problem. *Computers & Operations Research*, 74:152 – 164, 2016.

[38] P.H. Vance, C. Barnhart, E.L. Johnson, and G.L. Nemhauser. Solving binary cutting stock problems by column generation and branch-and-bound. *Computational Optimization and Applications*, 3(2):111–130, 1994.

[39] Z. Wang, K.W. Li, and J.K. Levy. A heuristic for the container loading problem: A tertiary-tree-based dynamic space decomposition approach. *European Journal of Operational Research*, 191(1):86–99, 2008.

[40] Gerhard Wäscher, Heike Haußner, and Holger Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3): 1109–1130, 2007.

[41] 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.

[42] X. Zhao, J.A. Bennell, T. Bektaş, and K. Dowsland. A comparative review of 3D container loading algorithms. *International Transactions in Operational Research*, 23 (1-2):287–320, 2016.

[43] W. Zhu and A. Lim. A new iterative-doubling Greedy-Lookahead algorithm for the single container loading problem. *European Journal of Operational Research*, 222(3): 408–417, 2012.

[44] 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.

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

APPENDICES

Appendix A

Results for the Generated Benchmark Instances

# Items	Instance #	CPU (s)	LB	# Bins	Support (%)	(%) Items Supported
50	1	0.10	1	1	83.94	58.00
50	2	0.39	1	1	88.68	58.00
50	3	0.26	1	1	87.92	60.00
50	4	0.29	1	1	84.02	70.00
50	5	0.16	1	1	88.93	70.00
100	1	1.03	1	1	88.04	81.00
100	2	1.30	1	1	86.80	77.00
100	3	1.43	1	1	87.04	76.00
100	4	1.75	1	1	87.57	72.00
100	5	1.38	1	1	89.40	78.00
150	1	10.91	2	2	87.67	70.00
150	2	20.30	2	2	91.93	76.00
150	3	12.90	2	2	88.84	75.33
150	4	12.08	2	2	90.70	72.00
150	5	21.31	2	2	90.36	79.33
200	1	55.98	2	2	91.21	81.50
200	2	36.41	2	2	90.41	77.50
200	3	30.26	2	2	90.84	81.50
200	4	41.14	2	2	92.87	88.50
200	5	23.50	2	2	91.67	79.50
500	1	79.15	4	5	91.48	80.40
500	2	148.36	4	5	92.17	79.60
500	3	171.75	4	5	92.21	80.80
500	4	104.23	4	5	91.99	81.00
500	5	98.08	4	5	92.05	83.20
1000	1	226.20	7	9	92.47	83.70
1000	2	216.53	7	9	92.63	82.70
1000	3	215.23	8	9	92.46	83.80
1000	4	191.28	7	9	93.01	83.60
1000	5	240.99	7	8	92.76	82.60
1500	1	480.76	11	13	93.02	84.27
1500	2	426.49	11	12	93.32	86.47
1500	3	438.23	11	13	93.27	84.93
1500	4	433.17	11	12	93.36	86.00
1500	5	417.68	11	12	93.31	86.13
2000	1	784.98	14	16	93.06	84.05
2000	2	969.60	14	16	93.36	85.65
2000	3	855.88	14	16	93.12	85.20
2000	4	1037.95	14	16	92.64	83.60
2000	5	794.92	14	16	92.68	84.40

Table A.1: Results of LCGA on class 1 of the generated instances.

# Items	Instance #	CPU (s)	LB	# Bins	Support (%)	(%) Items Supported
50	1	0.24	1	1	85.76	66.00
50	2	0.15	1	1	90.30	64.00
50	3	0.21	1	1	92.86	74.00
50	4	0.25	1	1	87.48	68.00
50	5	0.08	1	1	91.10	66.00
100	1	1.32	1	1	81.91	66.00
100	2	0.94	1	1	86.11	74.00
100	3	1.63	1	1	87.08	73.00
100	4	2.03	1	1	91.75	77.00
100	5	2.08	1	1	87.36	70.00
150	1	13.80	2	2	92.37	76.00
150	2	13.51	1	2	90.78	74.67
150	3	14.43	1	2	91.17	74.67
150	4	25.25	1	2	92.01	76.00
150	5	10.20	1	2	93.67	76.00
200	1	49.60	2	2	89.62	80.50
200	2	56.44	2	2	93.68	82.50
200	3	46.87	2	2	92.94	81.50
200	4	88.03	2	2	91.96	79.00
200	5	35.49	2	2	89.85	75.50
500	1	148.14	4	5	92.63	80.00
500	2	207.27	4	5	92.17	81.20
500	3	149.97	4	5	91.14	78.40
500	4	101.81	4	5	91.09	80.00
500	5	114.67	4	5	91.79	78.60
1000	1	266.68	7	9	92.34	83.40
1000	2	273.50	7	9	92.20	82.70
1000	3	303.37	7	9	92.78	84.70
1000	4	190.73	7	9	91.67	82.10
1000	5	189.00	7	9	93.18	83.40
1500	1	504.79	11	12	93.04	85.13
1500	2	570.13	11	12	92.96	84.47
1500	3	508.58	11	12	93.51	86.13
1500	4	793.34	11	12	92.77	84.80
1500	5	550.26	11	12	92.59	84.20
2000	1	1187.42	14	16	92.71	84.30
2000	2	983.60	14	16	92.86	85.65
2000	3	915.78	14	16	92.71	85.05
2000	4	941.06	14	16	92.97	85.50
2000	5	1147.59	14	16	93.06	85.05

Table A.2: Results of LCGA on class 2 of the generated instances.

# Items	Instance #	CPU (s)	LB	# Bins	Support (%)	(%) Items Supported
50	1	0.13	1	1	82.99	58.00
50	2	0.15	1	1	85.53	64.00
50	3	0.30	1	1	94.85	70.00
50	4	0.14	1	1	90.68	64.00
50	5	0.12	1	1	90.39	58.00
100	1	1.64	1	1	83.29	70.00
100	2	1.30	1	1	90.97	78.00
100	3	1.71	1	1	86.15	68.00
100	4	2.09	1	1	90.64	74.00
100	5	0.76	1	1	86.26	68.00
150	1	29.15	1	2	89.93	80.67
150	2	13.76	1	1	90.67	78.00
150	3	18.79	1	2	88.86	80.00
150	4	17.55	1	2	88.32	77.33
150	5	25.78	1	2	88.94	79.33
200	1	66.92	2	2	91.52	77.00
200	2	44.47	2	2	92.82	80.50
200	3	98.57	2	2	89.97	77.00
200	4	76.35	2	2	92.03	77.00
200	5	66.14	2	2	90.24	76.50
500	1	181.33	3	4	92.73	83.00
500	2	416.39	3	4	91.22	79.60
500	3	252.67	3	4	91.76	80.40
500	4	410.61	3	4	91.04	79.20
500	5	462.00	3	4	92.04	81.80
1000	1	378.94	6	7	92.16	84.60
1000	2	389.29	6	7	92.09	84.80
1000	3	355.97	6	7	92.46	83.60
1000	4	658.97	6	7	91.61	84.20
1000	5	338.88	6	7	92.29	85.00
1500	1	1032.15	9	11	92.21	83.60
1500	2	801.09	9	11	92.30	83.33
1500	3	750.07	9	10	92.04	85.07
1500	4	780.79	9	10	92.84	84.93
1500	5	1059.09	9	11	92.28	83.60
2000	1	1330.08	12	14	91.89	83.35
2000	2	1068.59	12	14	92.94	85.35
2000	3	1243.27	12	14	92.53	85.10
2000	4	1103.31	12	14	92.71	84.75
2000	5	1295.56	12	14	92.45	84.95

Table A.3: Results of LCGA on class 3 of the generated instances.

# Items	Instance #	CPU (s)	LB	# Bins	Support (%)	(%) Items Supported
50	1	0.14	1	1	90.20	62.00
50	2	0.10	1	1	88.06	50.00
50	3	0.18	1	1	90.98	62.00
50	4	0.18	1	1	87.48	62.00
50	5	0.06	1	1	87.28	60.00
100	1	1.82	1	1	88.91	75.00
100	2	1.47	1	1	86.08	71.00
100	3	1.89	1	1	86.65	74.00
100	4	1.39	1	1	89.13	72.00
100	5	2.30	1	1	88.61	77.00
150	1	25.24	1	2	91.82	80.67
150	2	37.13	1	1	88.41	76.67
150	3	10.17	1	1	88.58	78.00
150	4	16.41	1	1	90.90	80.67
150	5	19.38	1	1	88.81	78.67
200	1	49.42	2	2	88.42	73.50
200	2	77.96	1	2	91.60	76.50
200	3	46.22	2	2	90.71	74.50
200	4	113.91	2	2	90.26	75.00
200	5	88.18	2	2	90.66	76.50
500	1	340.42	3	4	91.64	81.80
500	2	261.55	3	4	91.58	82.00
500	3	167.61	3	4	90.59	78.60
500	4	242.38	3	4	91.53	80.00
500	5	252.75	3	4	91.97	80.60
1000	1	516.74	6	7	90.78	82.20
1000	2	706.60	6	7	91.21	81.40
1000	3	470.70	6	7	91.93	82.80
1000	4	395.40	6	7	91.48	82.20
1000	5	592.44	6	7	90.45	81.70
1500	1	1074.27	8	10	91.70	82.87
1500	2	838.53	8	10	90.93	82.67
1500	3	1099.02	8	10	92.72	83.53
1500	4	886.32	8	10	91.48	82.80
1500	5	823.89	8	10	91.73	83.40
2000	1	1458.63	11	12	92.72	85.80
2000	2	1145.12	11	13	92.16	84.20
2000	3	1442.41	11	13	92.24	85.10
2000	4	1813.35	11	13	92.04	83.60
2000	5	1451.93	11	13	91.85	84.55

Table A.4: Results of LCGA on class 4 of the generated instances.

Appendix B

Sample Figures for LCGA Results on Generated Instances

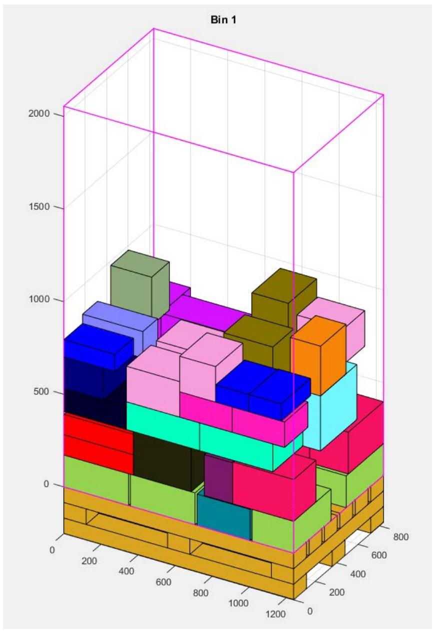


Figure B.1: Solution for class 1, 50 items, instance 1.

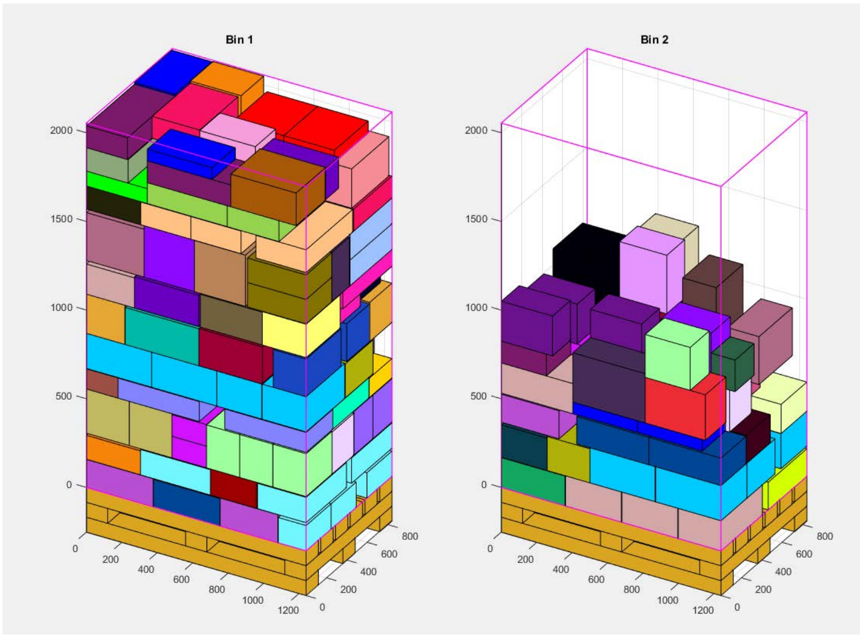


Figure B.2: Solution for class 2, 200 items, instance 1.

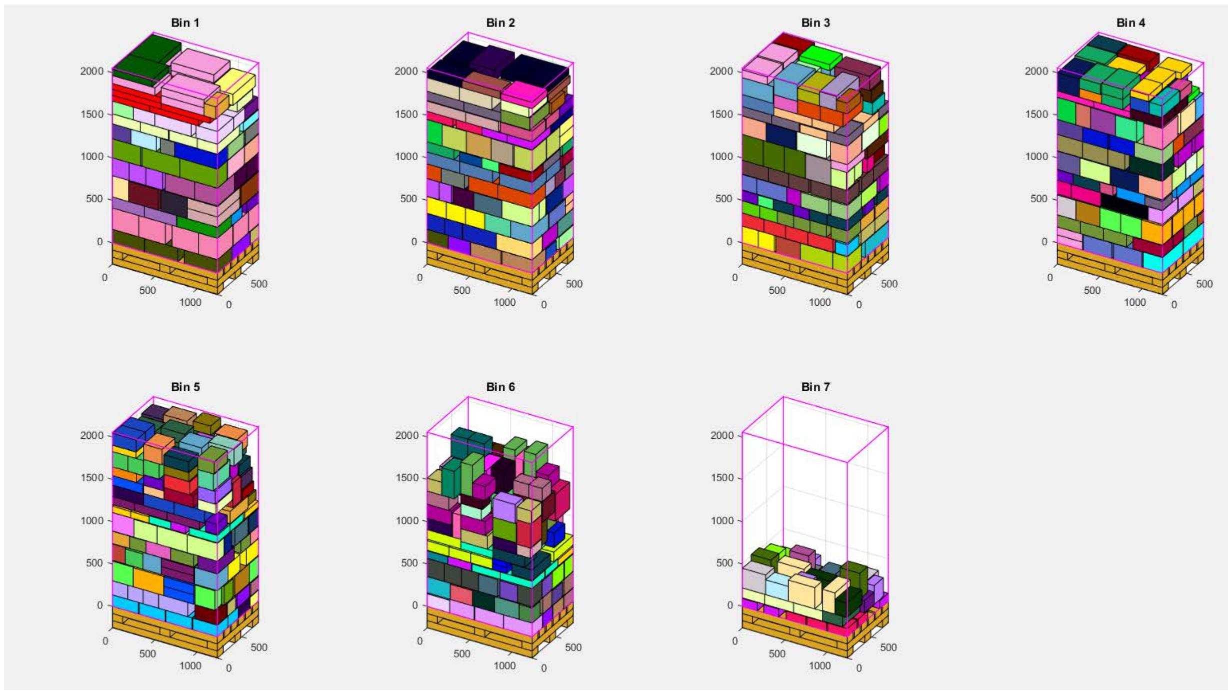


Figure B.3: Solution for class 3, 1000 items, instance 1.

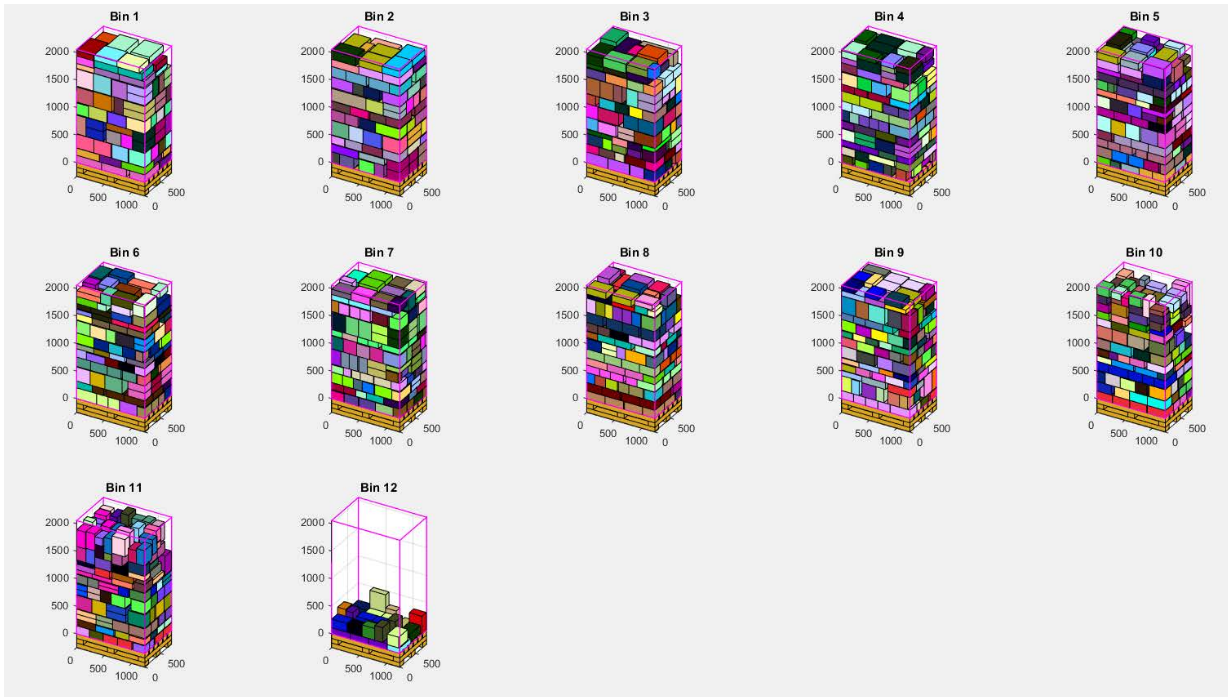
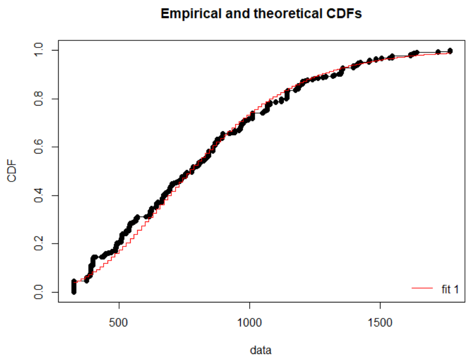


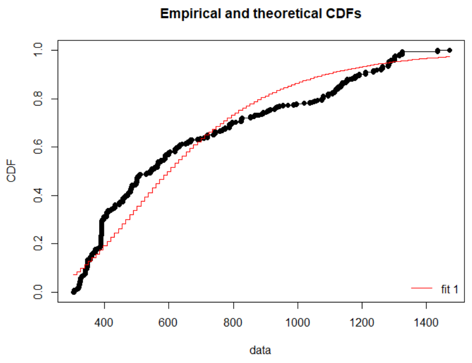
Figure B.4: Solution for class 4, 2000 items, instance 1.

Appendix C

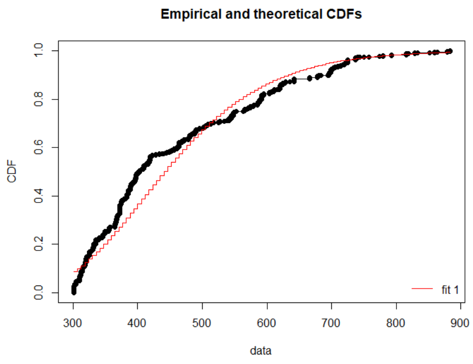
Cumulative Distribution Function Plots



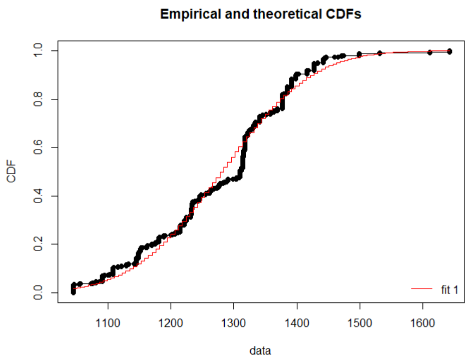
(a) Curve 1.



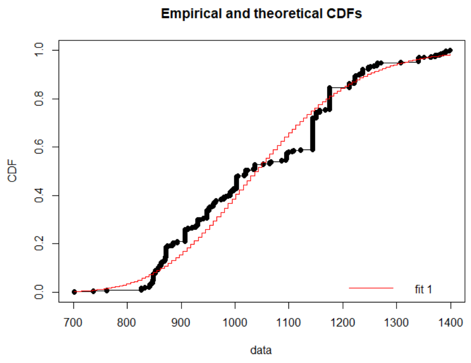
(b) Curve 2.



(c) Curve 3.



(d) Curve 4.



(e) Curve 5.

Figure C.1: Cumulative distribution functions of the fitted load capacity curves.

Appendix D

Results for the generated benchmark instances

Items	Instance	Pallets	CPU (s)	CPU per Pallets (s)	Avg Vol Use (%)	Avg Density (%)
50	1	1	0.23	0.23	30.80	71.24
50	2	1	5.43	5.43	30.36	71.91
50	3	1	0.21	0.21	33.35	74.61
50	4	1	5.53	5.53	29.87	71.09
50	5	1	0.18	0.18	32.39	73.87
100	1	1	6.42	6.42	57.10	77.15
100	2	1	6.35	6.35	57.38	77.46
100	3	1	6.27	6.27	56.74	77.47
100	4	2	0.94	0.47	32.18	79.64
100	5	1	1.27	1.27	62.19	82.51
150	1	2	6.42	3.21	47.35	78.04
150	2	2	10.11	5.05	49.26	79.84
150	3	2	7.78	3.89	47.42	77.76
150	4	2	4.03	2.01	45.17	77.31
150	5	2	2.92	1.46	45.73	80.35
200	1	3	29.19	9.73	41.35	79.46
200	2	2	13.52	6.76	61.51	81.83
200	3	2	13.28	6.64	61.59	81.83
200	4	2	16.84	8.42	59.92	80.27
200	5	2	7.75	3.88	62.40	82.78
500	1	5	256.04	51.21	62.64	76.07
500	2	5	245.32	49.06	62.22	87.35
500	3	5	447.20	89.44	62.27	75.82
500	4	4	278.50	69.63	76.60	76.62
500	5	4	150.32	37.58	76.87	76.91
1000	1	8	541.26	67.66	77.67	78.07
1000	2	8	817.21	102.15	77.91	79.72
1000	3	8	1330.33	166.29	79.88	80.00
1000	4	8	623.54	77.94	78.26	78.42
1000	5	8	783.11	97.89	78.32	78.62
1500	1	12	1495.68	124.64	77.64	78.93
1500	2	12	890.11	74.18	77.97	78.30
1500	3	12	1264.60	105.38	77.71	78.34
1500	4	12	1209.59	100.80	77.90	78.96
1500	5	12	1689.82	140.82	77.51	77.80
2000	1	16	1428.50	89.28	78.26	79.14
2000	2	16	1747.20	109.20	78.12	79.31
2000	3	15	2297.68	153.18	82.39	82.54
2000	4	15	2687.25	179.15	82.16	82.36
2000	5	16	2256.28	141.02	78.50	78.85

Table D.1: Results of DPLP on class 1 of the generated instances.

Items	Instance	Pallets	CPU (s)	CPU per Pallet (s)	Avg Vol Use (%)	Avg Density (%)
50	1	1	0.28	0.28	30.28	80.47
50	2	1	0.19	0.19	31.84	82.03
50	3	1	0.17	0.17	29.68	76.26
50	4	1	0.20	0.20	31.64	81.80
50	5	1	0.30	0.30	29.98	80.14
100	1	1	6.32	6.32	61.23	71.35
100	2	2	6.35	3.17	31.04	80.48
100	3	1	6.05	6.05	61.63	81.99
100	4	2	1.07	0.53	31.72	81.60
100	5	1	0.96	0.96	59.70	79.70
150	1	2	5.40	2.70	46.26	79.06
150	2	2	2.14	1.07	47.82	78.40
150	3	2	4.54	2.27	45.35	75.79
150	4	2	4.19	2.10	43.31	78.61
150	5	2	2.74	1.37	46.44	76.52
200	1	2	11.86	5.93	60.77	81.24
200	2	2	23.04	11.52	62.17	82.41
200	3	2	7.60	3.80	63.61	74.40
200	4	2	18.81	9.41	59.67	80.38
200	5	2	10.81	5.40	59.60	79.85
500	1	5	444.73	88.95	62.51	85.38
500	2	5	394.82	78.96	61.39	75.99
500	3	4	291.49	72.87	77.88	78.51
500	4	4	281.21	70.30	76.69	77.08
500	5	5	357.88	71.58	60.80	74.00
1000	1	8	733.34	91.67	76.79	77.00
1000	2	8	1466.15	183.27	77.61	77.78
1000	3	8	1753.32	219.17	76.55	76.74
1000	4	8	675.23	84.40	76.65	76.91
1000	5	8	1510.22	188.78	76.70	76.98
1500	1	12	1931.69	160.97	76.65	77.75
1500	2	12	1776.36	148.03	78.56	78.94
1500	3	12	1642.30	136.86	77.66	79.14
1500	4	12	1494.14	124.51	76.83	78.05
1500	5	12	1741.43	145.12	77.35	78.28
2000	1	15	1769.22	117.95	82.58	82.70
2000	2	16	1622.88	101.43	77.84	79.66
2000	3	15	2738.17	182.55	81.60	81.82
2000	4	16	2882.99	180.19	78.20	78.41
2000	5	16	2976.50	186.03	77.45	78.58

Table D.2: Results of DPLP on class 2 of the generated instances.

Items	Instance	Pallets	CPU (s)	CPU per Pallet (s)	Avg Vol Use (%)	Avg Density (%)
50	1	1	0.57	0.57	24.49	84.51
50	2	1	0.15	0.15	23.94	81.38
50	3	1	5.52	5.52	23.39	85.48
50	4	1	0.27	0.27	28.51	78.84
50	5	1	0.18	0.18	28.75	79.06
100	1	1	2.30	2.30	48.99	79.17
100	2	1	1.78	1.78	50.53	80.58
100	3	1	0.77	0.77	51.06	81.62
100	4	1	6.73	6.73	49.19	79.26
100	5	1	0.88	0.88	47.62	78.50
150	1	2	8.30	4.15	37.68	74.98
150	2	2	7.74	3.87	37.63	74.83
150	3	2	4.13	2.06	38.35	78.61
150	4	2	12.75	6.38	36.01	76.79
150	5	2	6.83	3.41	35.82	72.59
200	1	2	14.98	7.49	51.10	71.40
200	2	2	12.98	6.49	52.23	72.59
200	3	2	32.61	16.31	48.66	79.34
200	4	2	26.52	13.26	50.36	70.52
200	5	2	16.89	8.44	50.11	82.76
500	1	4	384.73	96.18	61.25	74.71
500	2	4	407.83	101.96	61.52	75.62
500	3	4	393.07	98.27	61.12	72.83
500	4	4	499.77	124.94	63.17	83.30
500	5	4	661.79	165.45	63.48	77.12
1000	1	7	1295.87	185.12	71.46	72.44
1000	2	7	1229.16	175.59	71.51	72.87
1000	3	7	967.71	138.24	72.55	72.73
1000	4	7	1128.10	161.16	70.81	72.94
1000	5	6	1899.21	316.54	80.26	80.51
1500	1	10	1868.64	186.86	73.59	75.24
1500	2	10	2108.89	210.89	75.41	76.34
1500	3	10	1938.94	193.89	74.14	75.35
1500	4	10	5131.19	513.12	73.87	74.21
1500	5	10	1648.88	164.89	75.30	76.53
2000	1	13	2955.90	227.38	77.50	77.68
2000	2	15	424.85	28.32	66.05	76.18
2000	3	15	497.05	33.14	65.81	76.50
2000	4	15	409.02	27.27	65.83	76.00
2000	5	15	1051.96	70.13	66.56	76.70

Table D.3: Results of DPLP on class 3 of the generated instances.

Items	Instance	Pallets	CPU (s)	CPU per Pallet (s)	Avg Vol Use (%)	Avg Density (%)
50	1	1	0.14	0.14	22.50	81.42
50	2	1	5.57	5.57	24.03	82.94
50	3	1	0.33	0.33	20.27	76.63
50	4	1	0.38	0.38	18.89	81.80
50	5	1	0.24	0.24	23.95	74.40
100	1	2	1.35	0.68	23.03	79.49
100	2	1	1.07	1.07	46.07	76.86
100	3	1	1.84	1.84	43.28	73.90
100	4	1	1.60	1.60	45.75	77.05
100	5	1	1.68	1.68	44.97	75.78
150	1	1	6.84	6.84	64.70	75.17
150	2	2	6.37	3.19	35.59	84.25
150	3	1	12.18	12.18	62.89	83.09
150	4	2	5.10	2.55	34.68	74.91
150	5	2	6.59	3.29	33.64	85.18
200	1	2	14.82	7.41	46.64	79.02
200	2	2	16.03	8.02	45.49	85.63
200	3	2	12.76	6.38	48.75	79.03
200	4	2	14.44	7.22	49.77	80.19
200	5	2	26.91	13.45	45.80	77.29
500	1	3	901.08	300.36	77.58	77.77
500	2	4	740.01	185.00	58.35	82.37
500	3	4	448.59	112.15	57.82	82.44
500	4	4	785.41	196.35	58.11	82.71
500	5	4	377.79	94.45	58.48	83.30
1000	1	7	1485.92	212.27	67.22	80.07
1000	2	6	2583.54	430.59	76.57	76.70
1000	3	8	278.57	34.82	56.96	78.91
1000	4	7	124.87	17.84	64.98	76.10
1000	5	7	203.41	29.06	66.15	76.87
1500	1	11	259.83	23.62	63.03	85.55
1500	2	11	248.11	22.56	63.71	74.81
1500	3	11	260.18	23.65	62.79	83.69
1500	4	11	341.02	31.00	62.33	82.71
1500	5	11	469.05	42.64	64.25	76.03
2000	1	14	796.95	56.93	65.66	85.83
2000	2	14	645.68	46.12	65.90	76.10
2000	3	14	536.66	38.33	65.47	75.64
2000	4	14	716.31	51.17	65.29	75.45
2000	5	15	545.65	36.38	60.51	81.87

Table D.4: Results of DPLP on class 4 of the generated instances.