DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF COPENHAGEN

Heuristics for Multidimensional Packing Problems

PhD Thesis

Jens Egeblad

July 4th, 2008
Academic advisor: David Pisinger

Preface

This Ph.D.-thesis was written in the years 2004 to 2008 at the Department of Computer Science at the University of Copenhagen (DIKU) under the supervision of Professor David Pisinger. The four years included two six month periods of leave of absence.

During those years I worked on several projects and visited several departments at other universities in two wonderful parts of the world. I spend a total of approximately seven months at Dipartimento di Ingegneria Meccanica e Gestionale at Politecnico di Bari in Puglia, Italy. I also spend six months during the fall of 2007 at Stanford Univesity in California, USA.

While all projects were both exciting and challenging one particular project stood out of the crowd. The project was a semi-commercial heuristic for container loading of furniture and contained many joyful elements. It was a wonderful way to get acquainted and experiment with the many requirements that must be tackled when dealing with practical problems for the industrial sector. Today, the difficulties of this project are hard to portray, but I still remember how great worries and concerns dominated the first many months.

During the first period of leave of absence I focused on the software development of this project. This was a wonderful opportunity to design a full-blown medium sized piece of computer software tool for use by the company, and the final version of that tool consisted of more than approximately 90,000 lines of code.

The most difficult part of this project was working with a tight schedule and having to come up with solutions to challenging problems within strict deadlines. Many sleepless nights were filled with frustrations and worries over not having a solution for yet another essential corner of the project. The paper presented in this thesis on the subject does not completely do justice to the huge amount of work involved.

In the end, however, it was extremely rewarding to visit the company's headquarters six moths after the project finished and find that many people were using the software concurrently on four different continents. Looking back, the frustrations and impossible deadlines are all forgotten. I now remember mainly the many fond memories of warm days and nights among good friends and colleagues, and think of this project as a successful marriage between theory and practice.

In a way, this particular project also represents the work of this thesis well. All of the projects are of strong direct relevance to other sectors – industrial or other scientific fields – and since I began I have been approached by several parties who have shown keen interest in commercializing ideas presented here. The thesis has involved many physical, cultural, and spiritual journeys, many were difficult but they were all enriching and exciting. It has been four wonderful years.

Acknowledgements

I would like to thank the many people who made this thesis not only possible but also joyful to work on. First, I would like to thank Professor Vito Albino and Professor Claudio Garavelli from Politecnico di Bari for allowing me to be part of their wonderful project and letting me stay for many times over a long period at their department. I would also like to thank Cosimo Palmisano, Stefano Lisi, and the many other members of the department for making each visit a joy both during work but also in the after hours.

A special thanks also goes to Professor Leonidas Guibas for letting me visit his department at Stanford University and the many members of the department that introduced me to interesting new scientific topics, but also life in The City, pizza and beer, and Happy Hours on fridays. Also Magda Jonikas

and Alain Laederach deserves many thanks for being patient with my lack of biological knowledge and for providing the data and ideas which made the chronological last paper of this thesis possible.

I would also like to thank the entire group of people in Copenhagen for our many discussions of both scientific and less scientific nature and for their patience with my use of the limited computational resources available to us. A huge thanks goes to Benny K. Nielsen for the vast number of discussions on relaxed placement methods, for wonderful team work, and, not the least, for tolerating several messy paper drafts.

My girlfriend, Zulfiya Sukhova, also deserves incredible appreciation for being open-minded and positive about the many weekends and nights which were spent on work, and the many months I have stayed abroad during these years. Surely, there cannot be a more understanding person.

Most importantly, I would like to thank my advisor Professor David Pisinger who has never been too busy to discuss a scientific topic or to give a useful advice. The support, advices, and general help through the entire four years have been completely solid and have all greatly exceeded my expectations.

– Thank you all

Jens Egeblad
July, 2008

Abstract

In this thesis we consider solution methods for packing problems. Packing problems occur in many different situations both directly in the industry and as sub-problems of other problems. High-quality solutions for problems in the industrial sector may be able to reduce transportation and production costs significantly. For packing problems in general are given a set of items and one of more containers. The items must be placed within the container such that some objective is optimized and the items do not overlap. Items and container may be rectangular or irregular (e.g. polygons and polyhedra) and may be defined in any number of dimensions. Solution methods are based on theory from both computational geometry and operations research.

The scientific contributions of this thesis are presented in the form of six papers and a section which introduces the many problem types and recent solution methods. Two important problem variants are the knapsack packing problem and the strip-packing problem. In the knapsack packing problem, each item is given a profit value, and the problem asks for the subset with maximal profit that can be placed within one container. The strip-packing problem asks for a minimum height container required for the items. The main contributions of the thesis are three new heuristics for strip-packing and knapsack packing problems where items are both rectangular and irregular.

In the two first papers we describe a heuristic for the multidimensional strip-packing problem that is based on a relaxed placement principle. The heuristic starts with a random overlapping placement of items and large container dimensions. From the overlapping placement overlap is reduced iteratively until a non-overlapping placement is found and a new problem is solved with a smaller container size. This is repeated until a time-limit is reached, and the smallest container for which a non-overlapping placement was found is returned as solution. In each iteration, a single item is translated parallel to one of the coordinate axes to the position that reduces the overlap the most. Experimental results of this heuristic are among the best published in the literature both for two- and three-dimensional strip-packing problems for irregular shapes.

In the third paper, we introduce a heuristic for two- and three-dimensional rectangular knapsack packing problems. The two-dimensional heuristic uses the sequence pair representation and a novel representation called sequence triple is introduced for the three-dimensional variant. Experiments for the two-dimensional knapsack packing problem are on-par with the best published in the literature and experiments for the three-dimensional variant are promising.

A heuristic for a three-dimensional knapsack packing problem involving furniture is presented in the fourth paper. The heuristic is based on a variety of techniques including tree-search, wall-building, and sequential placement. The solution process includes considerations regarding stability and load bearing strength of items. The heuristic was developed in collaboration with an industrial partner and is now being used to solve hundreds of problems every day as part of their planning process.

A simple heuristic for optimizing a placement of items with respect to balance and moment of inertia is presented in the fifth paper. Ensuring that a loaded consignment of items are balanced throughout a container can reduce fuel consumption and prolong the life-span of vehicles. The heuristic can be used as a post-processing tool to reorganize an existing solution to a packing problem.

A method for optimizing the placement of cylinders with spherical ends is presented in the last paper. The method can consider proximity constraints which can be used to describe how cylinders should be placed relative to each other. The method is applied to problems where a placement of capsules must be found within a minimal spherical or box-shaped container and to problems where a placement within a given arbitrarily container must be found. The method has applications for prediction of RNA tertiary structure.

Abstract in Danish

I denne afhandling betragtes løsningsmetoder til pakningsproblemer. Løsninger af meget høj kvalitet for industrielle problemer kan reducere transport- og produktionsomkostninger betydeligt. Pakningsproblemer opstår i mange forskellige situationer både direkte i industrien men også som delproblemer af andre problemer. Generalt består pakningsproblemer af en mængde af figurer og en eller flere containere. Figurer og containere kan være rektangulære og irregulære (fx polygoner og polyedra) og være defineret i et vilkårligt antal dimensioner. Figurerne skal placeres i containerne uden at overlappe sådan, at en målfunktion optimeres. De anvendte løsningsmetoder består af teknikker både fra geometri og optimering.

De videnskabelige bidrag i denne afhandling præsenteres i seks adskildte artikler, som indledes med en introduktion til problemtyper og populære løssningsmetoder. To vigtige varianter af pakningsproblemer er rygsækpakning og strip-pakning. I rygsækpakningsproblemer gives hver figur en profitværdi, og problemet er nu at bestemme en delmængde af figurer som kan være i rygsækken med en maksimal profit-sum. I strip-pakningsproblemet ønsker vi at finde en container med minimal længde, der kan indeholde alle figurerne. Afhandlingens hovedbidrag består af tre nye heuristikker til strip- og rygsækpakningsproblemer med rektangulære og irregulære figurer.

I de første to artikler præsenteres en heuristik til løsning af det multidimensionale strip-pakningsproblem. Heuristikken starter med en lang container og en tilfældig placering med overlap. Overlappet reduceres iterativt indtil en placering uden overlap findes, hvorefter et nyt problem med en mindre container løses. Dette gentages så mange gange som muligt indenfor en bestemt tidsgrænse, hvorefter den mindste container med en tilhørende placering uden overlap returneres som løsning. I hver iteration flyttes en enkelt figur parallelt med en af koordinatakserne til en placering med mindre overlap. De eksperimentielle resultater for denne heuristik er blandt de bedste, der er publiceret både for to- og tredimensionale problemer med irregulære figurer.

I den tredje artikel introduceres en heuristik til to- og tredimensional rektangulære rygsæksproblemer. Heuristikken til todimensionale problemer benytter *sekvensparrepræsentationen* og en ny repræsentation introduceres til tredimensional problemer. Eksperimentielle resultater for todimensionale problemer er på niveau med de bedste der er publiceret og resultater for de tredimensionale problemer er lovende.

En heuristik til det tredimensionale rygsækpakningsproblem, hvor figurer er møbler præsenteres i den fjerde artikel. Heuristikken er baseret på en række af teknikker heriblandt *træsøgning*, *vægbygning*, og *sekvential placering*. Heuristikken blev udviklet i samarbejde med en industripartner, og bliver nu anvendt til løsning af hundredevis af problemer dagligt som del af deres planlægning.

En enkel heuristik til optimering af placering af figurer med hensyn til balance og moment præsenteres i den femte artikel. Balancerede placeringer kan reducere brændstofsforbrug og forlænge levetiden af de anvendte transportmidler. Heuristikken kan bruges som en efterprocesseringsværktøj til omorganisering af en eksisterende løsning til et pakningsproblem.

En metode til optimering af placering af cylindere med kugleformede ender præsenteres i den sidste artikel. Metoden kan håndtere afstandskrav, som kan bruges til at angive, hvordan cylinderne skal placeres i forhold til hinanden. Metoden er anvendt på problemer, hvor en placering af cylindere skal findes indenfor en minimal kugleformet- eller rektangulær container, og til problemer, hvor en placering indenfor en vilkårlig container skal findes. Metoden kan anvendes til forudsigelse af tertiære RNA strukturer.

Contents

1 Introduction

Packing problems have spawned interest from the mathematical community for centuries and arise in numerous industrial situations with large potential for cost-savings and reduction in pollution caused by carbon dioxide emission.

The term packing problem is commonly used for a problem where one wishes to find a placement of small items within one or several larger objects. Although this definition may seem abstract, packing problems arise in many practical situations of our everyday life.

One of the most obvious examples occurs in the super-market. While the price of a superfluous bag will have a limited impact on the household economy, we are compelled to minimize the environmental impact of our purchases and therefore seek to minimize the number of grocery bags we use to pack our commodities in. When we divide the items among the bags, we reflect on their shape and weight, to distribute them evenly.

Another common problem occurs before an extended period of travel. Here the most useful set of items which can be packed compactly in a suitcase must be selected, and often this selection should be made such that the total weight is less than the weight limit imposed by any airline involved in the journey.

Although household problems are too small to warrant extended mathematical analysis, improvement of solutions to packing problems in the industrial sector may lead to substantial cost reductions. The problems occur both during manufacturing and transportation of goods and several solution methods will be presented in this thesis. Packing problem may also exist as sub-problems in other scientific fields and a problem with a biological origin will be discussed in this thesis along with a solution method.

Determining the optimal way to cut a large item into smaller pieces is equivalent to determining the optimal way to place the smaller pieces within a container shaped like the large item. Therefore, packing problems are synonymous with cutting problems and authors refer to the problems we consider in this thesis as either *packing*, *cutting*, or, *packing and cutting* problems.

Solution methods for these difficult problems consists of a combination of techniques from the fields of computational geometry, operations research, and algorithms. Therefore any researcher with a background in those fields should find the topic compelling to work with.

The problems have been studied since the nineteen-sixties and a plethora of solution methods have been presented over the years. The methods have been mostly heuristic, but exact algorithms have gained ground in recent years and even a few approximation algorithms have been presented.

Due to the required reduction of carbon emission, the increasing global competition, and the ever rising oil prices, new methods that improve the current results are needed by industrial sector. At the same time, increasing computational power and recent research in computer science enable us tackle larger problems with new and better methods.

Items can be rectangular (rectangles or boxes) or non-rectangular (e.g. polygons and polyhedra). Throughout this thesis we will refer to items which are non-rectangular and non-spherical as irregular. While computational methods for packing of rectangular items are able to produce high quality solutions at this time, it seems that there is still potential improvement of for methods that can handle non-rectangular item packing, especially in three-dimensions.

The main objective of this thesis is *to study novel heuristics for two- and three-dimensional packing problems involving rectangular and irregular shapes*.

1.1 Outline

The scientific contributions of this dissertation are presented as six individual papers. One has been published, one is in press and available online, one has been accepted for publication, two have been submitted for publication, and one represents work in progress. All submitted and accepted papers have been or are being peer-reviewed. The papers are reproduced completely as in their final revision before publication, acceptance, or submission. The six papers are as follows:

- **[A] Fast Neighborhood Search for two- and three-dimensional nesting problems**
  Jens Egeblad, Benny K. Nielsen, and Allan Odgaard. (published, 2007)

  In this paper we describe a novel heuristic for solving two- and three-dimensional strip-packing problems involving irregular shapes represented by polygons.

- **[B] Translational packing of arbitrary polytopes**
  Jens Egeblad, Benny K. Nielsen, and Marcus Brazil (accepted for publication, 2008)

  Packing of shapes represented by polyhedra, and a generalization of the work from the first paper ([A]) to three- and higher dimensions are described in this paper.

- **[C] Heuristic approaches for the two- and three-dimensional knapsack packing problem**
  Jens Egeblad and David Pisinger (in press and available online, 2007)

  In the paper, we introduce a new heuristic for the two- and three-dimensional rectangular knapsack packing problem which is based on the sequence pair and sequence triple representations for rectangle and box placements.

- **[D] Heuristics for container loading of furniture**
  Jens Egeblad, Claudio Garavelli, Stefano Lisi, and David Pisinger (submitted, 2007)

  A heuristic for the three-dimensional knapsack packing problem of furniture is presented in this paper. The heuristic is part of a semi-commercial program used by a very large Italian furniture producer and consists of a wide variety of sub-parts.

- **[E] Two- and three-dimensional placement for center of gravity optimization**
  Jens Egeblad (submitted, 2008)

  Heuristics for balanced placement of polygon and polyhedral items within a container are presented here. This builds on work from the papers [A] and [B].

- **[F] Three-dimensional constrained placement of capsules with application to coarse grained RNA structure prediction**
  Jens Egeblad, Leonidas Guibas, Magdalena Jonikas, and Alain Laederach (draft, 2008)

  This paper represents work in progress. A novel model for coarse grained RNA structure prediction based on compact placement of capsules (cylinders with spherical ends) and and technique for cylinder placement within a molecular envelope are presented.

The dissertation is organized as follows. First, in Section 2, the main problems of this thesis are presented and it is shown that they are $\mathcal{NP}$-hard. In order to aid the reader, the most popular and interesting solution methods are discussed in Section 3 along with speculations about future directions. In Section 4 the individual papers are presented and discussed. Finally, in Section 5, we give a conclusion and summarize possible future directions. The subsequent chapters (A,...,F) consists of the six papers which were produced as part of the thesis. An addendum for the paper [A] elaborates on a few missing details and presents updated experiments ([A.1]).

Definition of Packing Problems.

Given are two sets of elements, namely

- a set of large objects (input, supply) and
- a set of small items (output, demand)

which are defined exhaustively in one, two, three or an even larger number (n) of geometric dimensions. Select some or all small items, group them into one or more subsets and assign each of the resulting subsets to one of the large objects such that the geometric condition holds, i.e. the small items of each subset have to be laid out on the corresponding large object such that

- all small items of the subset lie entirely within the large object and
- the small items do not overlap,

and a given (single-dimensional or multi-dimensional) objective function is optimised. We note that a solution of the problem may result in using some or all large objects, and some or all small items, respectively.

Figure 1: The definition of packing problems given by Wäscher et al. [157].

2 Preliminaries

The immense number of papers on cutting and packing problems renders the field difficult to approach as a newcomer. At the time of writing, there are no textbooks on the field and while several surveys exists (see [30, 44, 45, 102]), they are often brief or with emphasis on particular problems or methods.

In this section we will list some of the problems which are discussed throughout this thesis. Our focus is on multidimensional problems; i.e., two- or higher dimensional. The problems are presented and defined in Section 2.1. In Section 2.2 we present a typology which covers the problem types, and in Section 2.3 we show that most of the problems are $\mathcal{NP}$-hard.

We consider problems where we wish to determine if (and often how) a set of items can be placed within one or several containers. In feasible solutions items must be placed such that they do not occupy the same area of the container, i.e. they are not allowed to overlap. An elaborate definition of the problems was given by Wäscher et al. [157] and is quoted in Figure 1.

The definition given by Wäscher et al. [157] is somewhat abstract, will match a very large number of problems, and it would be a long and tedious job to list them all. In this text we will deal with three main groups of problems: subset selection, container-count minimization, and container-size minimization. In *container count minimization* problems the number of containers should be minimized. In *subset selection problems* an optimal subset, with respect to an objective function, of items must be selected. Finally, in *container minimization problems* the size of the container should be minimized. The problems consist of the following ingredients (see Figure 2):

- **A container object C.** In most of this text, we will deal with only one *type* of container, but some problems are defined with respect to multiple types of containers. If C is rectangular, we let W be the width, H the height, and, for three-dimensional problems, D the depth of C.

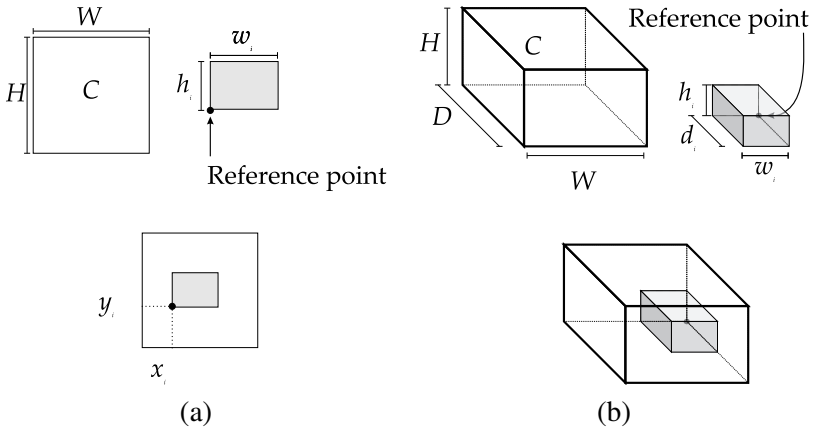


Figure 2: *(a) A two-dimensional container, item, and placement of an item within the container. (b) Three-dimensional variant.*

- **A set of items** I. The items are to be packed within C. We let $n = |I|$ be the number of items. If items have a rectangular shape, we let w_i be the width, h_i the height, and, for three-dimensional problems, d_i the depth of item $i \in I$. Non-rectangular shapes may be simple to describe mathematical like a sphere or be completely arbitrary. Arbitrary shapes may be represented in a number of different ways which we will consider in Section 3.1. In general, we will refer to arbitrarily shaped items as *irregular items*. Rotation of items may or may not be allowed.

- **A *placement*** P. Each item has a reference point, and the placement describes the position of each item's reference point in the container(s). For rectangular items we let the reference point be the lower-left-back corner of the item. For problems, where rotation or mirroring is allowed, the placement may also describe the amount each item is rotated and if it is mirrored. To simplify matters we will generally not distinguish between particular placements in this and the subsequent section, but use x_i, y_i, and z_i to indicate the position of item i's reference point.

An important part of evaluating solutions to packing problems in general is the *utilization*. The utilization of a placement is the area or volume occupied by the items within the container divided by the the area or volume of the container.

2.1 Problem Types

One-dimensional problems are in a sense the simplest type of packing problems, and we will begin by, briefly, considering them, in order to move on to the more complicated higher dimensional problems which are the focus of this text.

2.1.1 One Dimensional Problems

One of the simplest packing problems is the *one-dimensional cutting stock problem* (1DCSP). Materials such as paper, textiles and metal are commonly manufactured in large rolls. Different customers may desire different sized rolls and the large rolls are therefore cut into smaller rolls (see Figure 3 (a)). Given one large roll size, a set of small rolls sizes, and for each small roll size, a number of required rolls of that size, the one-dimensional cutting-stock problem is to find the minimal number of large rolls required (see Figure 3 (b)).

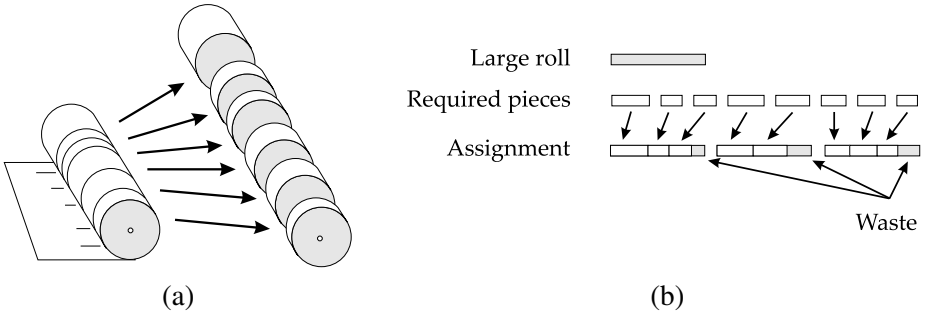


Figure 3: *The 1D cutting stock problem. (a) Large rolls must be cut into several smaller rolls. (b) Given large roll width, determine the number of large rolls required to cut the required pieces.*

A related problem is the *one-dimensional bin-packing problem* (1DBPP). For the bin-packing problem a set of items each with size w_i, for $i \in I$, and a bin size W are given, and the problem is to find the minimal number of bins such that all items are packed in a bin. Fundamentally, the two problems are the same, since one can transform an instance of one problem into the other, by replacing the term "large roll" with "bin" and "small roll" with item or vice versa. Authors generally distinguish between 1DCSP and 1DBPP by the number of each item size; cutting stock usually concerns few item sizes (*homogenous*), while bin-packing is used for problems with many item sizes (*heterogenous*). Both problems fall into the group of problems that we refer to as *container-count minimization*, since the number of large rolls or bins must be minimized. The bin-packing problem can be formulated as a mixed integer linear program (MIP):

$$
\begin{aligned}
\min \quad & \sum_{k=1}^{K} y_k, \\
\text{s.t.} \quad & \sum_{k=1}^{K} x_{ik} \geq 1, \;\; i \in I \\
& \sum_{i \in I} w_i x_{ik} \leq W y_k, \;\; k = 1, \ldots, K \\
\text{where} \quad & x_i \in \{0,1\}, \;\; i \in I \\
& y_k \in \{0,1\}, \;\; k = 1, \ldots, K.
\end{aligned} \tag{1}
$$

Here K is the maximum number of bins required. The binary variable y_k indicates whether bin k is used (opened). Variable x_{ik} indicates whether item i is in the kth bin. The first constraint ensures that all items are placed in a bin at least once, and the second constraint ensures that the items assigned to a single bin can be placed in the bin without overlap.

Another common packing problem is the one-dimensional knapsack problem (1DKPP). 1DKPP must also be solved when the cutting stock problem is to be solved using delayed column generation (see e.g. [33]). For the *one-dimensional knapsack problem* we are given a knapsack with a capacity and each item from I has a weight and a profit-value assigned to it. The objective is to determine the subset of items which can be packed in the knapsack without violating the weight capacity limit, such

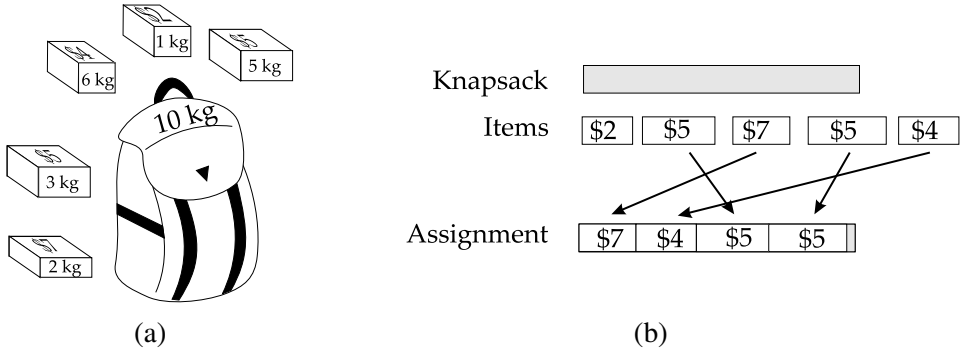


Figure 4: *The 1D knapsack packing problem. (a) A knapsack must be filled with the most profitable set of items with its weight limit. (b) The knapsack and items represented as a rectangles with equal height. The width of each item represents its weight.*

that the sum of the profit of the items from the subset is maximal. We can write this as a MIP:

$$\begin{aligned} \max \quad & \sum_{i \in I} p_i x_i, \\ \text{s.t.} \quad & \sum_{i \in I} w_i x_i \leq W \\ \text{where} \quad & x_i \in \{0,1\},\ i \in I. \end{aligned} \tag{2}$$

Here W is the knapsack's weight capacity, w_i is the weight and p_i the profit of each item $i \in I$. The binary variable x_i indicates whether item i is chosen. The constraint is the *capacity constraint*, which ensures that all items can fit inside the knapsack without "overlap". The problem is illustrated on Figure 4. The knapsack packing problem is a *subset selection problem*.

Since this text focuses on packing problems in higher dimensions, we will abandon the one-dimensions problems for now and instead move to the more challenging multi-dimensional problems. For more information on one-dimensional knapsack packing problems we refer to the book by Kellerer et al. [90].

2.1.2 Multi Dimensional Subset Selection Problems

The *two-dimensional knapsack packing problem* (2DKPP) and the one-dimensional variant are related. For the two-dimensional variant a plate of some size $W \times H$ is given as well as the set of items I. As for the one dimensional variant of the knapsack packing problem, one is given a set of items, each item is assigned a profit p_i and one must select a maximal profit subset of items $I' = \{i \in I | x_i = 1\}$, such that $\sum_{i \in I} p_i x_i$ is maximized and I' can be packed in the container.

Some authors (e.g. Boschetti et al. [21], Hifi [79], Lai and Chan [95]) also refer to this problem as the *two-dimensional cutting stock problem* (2DCSP). When this terminology is used there are a number of differences. For the 2DCSP items are usually more homogeneous and identical items are grouped. For each group, a lower-bound value designates the minimum number of items from that group to be packed.

These problems occurs in glass and metal industries, where large plates must be cut into smaller pieces which are sold to customers. A simple profit-value for each item is the area it occupies of the plate. If this value is used the objective is to maximize *utilization* of the plate. The problem appears in both *unconstrained* and *constrained* versions. In the unconstrained version the number of items of each type is infinite.

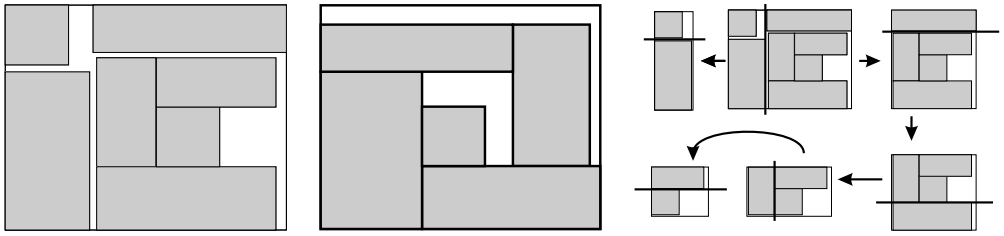


Figure 5: *Guillotine cutting. (a) Guillotine cuttable placement. (b) Non-guillotine cuttable placement. No sequence of cuts can cut all the items from the plate without cutting an item. (c) Cutting sequence for (a).*

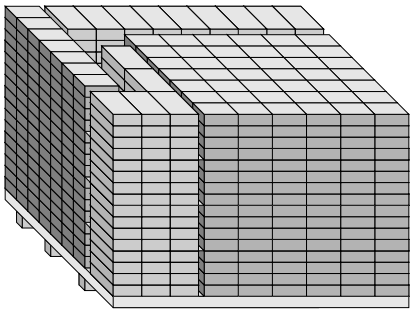


Figure 6: *The manufacturer's pallet loading problem is to determine the optimal loading of a set of identical items on a pallet.*

Some cutting-machines cannot stop in the middle of the plate, but need to divide the plate in its entirety either horizontally or vertically in each cut. This is referred to as a *guillotine cutting stock* and illustrated on Figure 5. Another form of cutting, right-angled cutting, where the cutting-machine can stop in the middle of the plate but only start from one of the edges, was investigated by Bukhvalova and Vyatkina [22].

A related two-dimensional problem is the manufacturer's pallet loading problem (PLP). Here one wishes to find an optimal loading pattern of identical pieces (boxes), which must be loaded on a pallet. Although the problem is really a three-dimensional problem as noted by Bischoff and Ratcliff [17], it is commonly modeled as a two-dimensional variant, where only one layer of items is considered and this layer is replicated a number of times. The problem is illustrated on Figure 6.

Since there is only one item-type, the objective is to find the loading pattern that maximizes the number of that item. It can therefore be considered a special case of the two-dimensional knapsack packing problem, where the lower bound of items is 0, the number of items is infinite, the number of item types is 1, the items can be rotated by 90 °, and the profit of each item is one. Since the items are assumed to be identical, we will not go into further details on the problem, but simply make a few remarks. Exact algorithms for this problem have been introduced by Dowsland [43] and Bhattacharya and Bhattacharya [14], a survey of solution methods for PLP was given by Balasubramanian [5], a polynomial time algorithm for a version of the problem where placements are required to guillotine-cuttable was presented by Tarnowski et al. [148], while several heuristics were presented by Herbert and Dowsland [77], Lins et al. [97], Young-Gun and Maing-Kyu [161].

2DKPP generalizes to three dimensions. A special variant of the problem in three-dimensions,

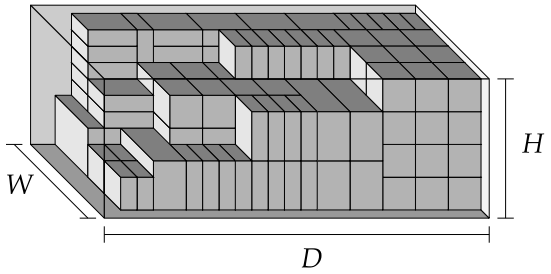


Figure 7: *The container loading problem and the common use of width (W), height (H), and depth (D).*

is the *container loading problem* (CLP). The container loading problem occurs in the transportation industry, where items are transported in shipping-containers. Here one is given a set of items and the objective is find a packing of the items that maximizes the volume occupied by them. This corresponds to a *three-dimensional knapsack packing problem* (3DKPP) where the profit of each item is set to its volume. The literature considers mostly rectangular items for the container loading problem, and, unlike the knapsack problem, rotation of the items is generally allowed, although each item may only be rotated a specified subset of the six axis-aligned rotations.

Standard internal dimensions of shipping containers are $234 \times 234 \times 592$ cm^3 for 20-feet containers and $234 \times 234 \times 1185$ cm^3 for 40-feet. Therefore, solution methods for CLP often take advantage of the elongated container and the large dimensions of the container relative to items, where as, methods for three-dimensional knapsack packing problems in general, cannot take advantage of this property. Figure 7 illustrates the CLP.

Another distinction between three-dimensional knapsack packing problems and container loading problems is that CLP generally considers hundreds of items with a volume sum close to the volume of the container, while 3DKPP considers less than 100 items with a combined volume which may be many times larger than the container size.

Apart from the differences related to container dimensions, the item numbers, and the profit value, container loading problems often also consider stability of the items, i.e, that the items are not floating in mid-air and will not drop to the floor once the container is moved.

2.1.3 Multi Dimensional Container-Count Minmization

The one dimensional bin-packing problem can also be generalized to more dimensions. Here one wishes to minimize the number of bins required to pack a set of items. The literature deals mostly with rectangular items and as for the knapsack packing problem, it is common not to allow rotation. The bin-packing problem occurs naturally in the industry. Often one can have a set of plates which must be cut into smaller items, or a number of containers into which items must be divided, and one wishes to minimize the number of plates or containers used.

Slightly confusing, this problem is also commonly referred to as the two-dimensional cutting stock problem (by e.g. Gilmore and Gomory [69]), and as for the variant related to the knapsack packing problem homogeneous and identical items are grouped and for each group, a lower-bound value designates the minimum number of items from that group to be packed. To avoid further complications we will refrain from using the cutting stock terminology in the remainder of this text.

Special variants of the bin-packing problem includes the *multi-container loading problem* (MLCP) and the *multi-pallet loading problem* (MPLP). In the multi-container loading problem, one wishes to

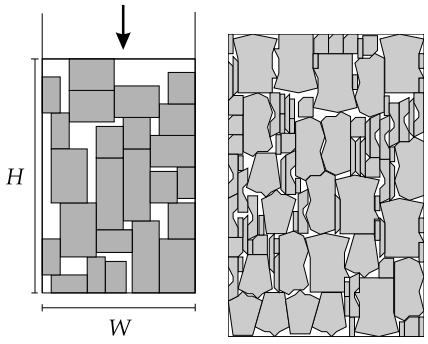


Figure 8: *(a) The strip-packing problem. The height (H), also referred to as the length, of the container must be minimized. (b) Example packing of irregular items.*

minimize the number of containers required to load a set of items. As for the ordinary container loading problem, a single container may be able to hold hundreds of items while the bins of the bin-packing problem may contain far fewer items.

Unlike single pallet loading, multi-pallet loading concerns different items, and the objective is to determine the minimal number of pallets required to load the items. Multi-pallet loading differs from multi-container loading, in that the container is smaller since it is a pallet, and that there may be stricter requirements related to stability.

2.1.4 Container Minimization

A set of problems, that occur only in two or more dimensions, are problems where the objective is to determine the minimal container that is large enough to enclose a given set of items. The objective may be to minimize one dimension of the container (*strip-packing*), minimize the area or volume (*area minimization*), minimize the circumference of the container, or minimize the radius of a circular container (*circle packing*).

Strip-Packing Strip-packing is probably the most commonly studied problem type in this category. The problem originates from the textile industry, but also occurs in the metal industry. In the textile industry one is given a roll of fabric, which must be cut into different components that make up clothing items when sewn together. The objective is to place the pieces such that the length of the strip is minimized. The is equivalent to a maximizing the utilization of the fabric. More specifically the problem is as follows: One is given a set of items and must place them within a rectangular container where one dimension is given (the width of the strip), and the second dimension must be minimized. The second dimensions which is the height of the container, is referred commonly referred to as the length of the strip. Since the problem occurs in the clothing industry, items are commonly irregular. The strip-packing problem involving irregular items is often referred to as the *nesting problem*. To avoid any confusing we will refrain from using that terminology in this introduction although it is commonly used by researchers in the field. The strip-packing problem is depicted on Figure 8.

Three-dimensional strip-packing Strip-packing problems may be generalized to higher dimension by requiring that all but one of the container dimensions are given. The three-dimensional variant applies to rapid-prototyping machines, that generate simple plastic objects from computer aided design

(CAD) systems. The generated objects can be used as design models or prototypes, and allow a designer to physically interact with his new design. The term rapid comes from the fact that the machine can produce a physical replica of a three-dimensional computerized model within hours. Rapid prototyping machines generate models by building a set of layers. Each layer comes with a significant process time, and so it is favorable to minimize the number of layers which must be generated for a given set of items. The problem may also occur as a subproblem in situations where a set of items should be packed as tightly as possible inside container, such that the remaining space can be used for a different set of items.

Area minimization Problems where several container dimensions must be minimized concurrently have also been studied in the literature. An example of such a problem is the *two-dimensional minimal area packing problem*, where one has to find the minimal area container large enough to encompass a set of items. This problem occurs in the Very Large Scale Integrated (VLSI) placement problem, that deals with the positioning of rectangular modules within a rectangular plate. However, commonly the problem of minimizing the area of such a plate is deemed less important compared to the more important problem of minimizing wire-length between the modules.

Circle Packing Another well-known container minimization problem is the circle packing problem, where one must find the minimal, with respect to radius, circular container which can enclose a set of circles. In the literature problems in the cable or oil pipeline industries are often cited as sources to this problem.

2.1.5 Mathematical problems

Packing problems have also attracted attention from the mathematical community, but here homogeneous or specific item dimensions are considered. As an example, the famous astronomer Johannes Kepler was looking for the most efficient way to pack equal-sized spheres in a large box in 1611. Kepler hypothesized that the optimal strategy was to stack the spheres similar to the way greengrocers stack oranges in crates. The utilization (volume occupied by the spheres divided by the volume of the box) of such a stacking is $\frac{\pi}{3\sqrt{2}} \approx 74.048\%$. Kepler's hypothesis turned out to be extremely difficult to prove, but Hales [75] recently published a convincing proof, although it has yet to be completely confirmed. Several other related problems have interested the mathematical community over the years. Some examples are:

- Determine the minimum square or circular container capable of containing n unit circles.
- Determine the maximal number of unit squares in square container with non-integer dimensions, and where the unit squares may be freely rotated.

Discussion of these variants of packing problems is beyond the scope of this text, and we will abandon them here in favor of the more practical problems that we have discussed so far.

2.2 Typologies

Because researchers come from different backgrounds, equivalent problems are often referred to with different names. This is already apparent in the prequel where the knapsack packing, container loading, and multi-pallet loading problems are all in a sense equivalent. To remedy this problem, several

Abbrev.	Description
IPPP	Identical Item Packing Problem, e.g., Pallet Packing.
SKP	Single Knapsack Problem. The knapsack packing problem as described in the prequel. Also includes the container loading problem.
MIKP	Multiple Identical Knapsack Problem. A subset of items for *multiple knapsacks* must be determined. Includes the multi-container loading problem and multi-pallet loading problem.
MHKP	Multiple Heterogenous Knapsack Problem. Variant of MIKP, where the knapsacks are not identical.
SLOPP	Single Large Object Placement Problem. This is a form of weakly heterogeneous single knapsack problem.
MILOPP	Multiple Identical Large Object Placement Problem. A variant of SLOPP with multiple knapsacks.
MHLOPP	Multiple Heterogenous Large Object Placement Problem. Variant of MILOPP with different knapsacks.

Table 1: Output maximization problems according to Wäscher et al. [157].

researchers have introduced typologies based on a limited set of core problems. In recent years the typology by Wäscher et al. [157] has replaced the older typology by Dyckhoff [47], and we will describe the former briefly here.

Wäscher et al. [157] start by dividing the input objects into two categories: large and small objects. The large object is/are the container(s), while the small objects are the items to be packed within them. Further, problems are divided into two main groups: *output maximization* and *input minimization*. Output maximization concerns problems where one is to place a set of small objects within a limited set of large objects. Input minimization concerns minimizing the number of large objects (or the size of one) required to pack the items.

Three groups are used to classify the variety of items: identical, weakly heterogeneous, and strongly heterogeneous. Similarly, the set of large objects are classified as either a single object (for output maximization), identical, or heterogenous. For input minimization the set of large objects may be further categorized as either identical or heterogenous.

These distinctions lead to a classification typology with a relatively low number of problem types. The problems types described in the previous sections can be classified using the typology by Wäscher et al. [157]. The subset selection problems of Section 2.1.2 are all output maximization problems and divided into the sub-categories listed in Table 1. The container-count minimization problems of Section 2.1.3 are all input minimization problems and are listed in Table 2. The container minimization problems from Section 2.1.4 are perceived by Wäscher et al. [157] as another category of input minimization problem named Open Dimension Problem (ODP).

To identify problem types further Wäscher et al. [157] also consider the "dimensionality" of a problem type, i.e., two-dimensional, three-dimensional, or d-dimensional, and the type of items, i.e, regular (rectangular, circular, cylindrical) or irregular.

Using the typology of Wäscher et al. [157] a two-dimensional knapsack packing problem with rectangles will be referred to as a "2-dimensional rectangular SKP", while a three dimensional strip-packing problem involving irregular shapes will be referred to as a "3-dimensional irregular ODP".

Abbrev.	Description
SBSBSP	Single Bin Size Bin Packing Problem. Bin Packing Problem as described in the prequel.
MBSBPP	Multiple Bin Size Bin Packing Problem. Bin Packing Problem with multiple bin sizes.
RBPP	Residual Bin Packing Problem (RBPP). Bin Packing with strongly heterogenous bin-types.
SSSCSP	Single Stock Size Cutting Stock Problem. Bin Packing Problem consisting of weakly heterogenous items.
MSSCSP	Multiple Stock Size Cutting Stock Problem. SSSCSP variant with multiple types of bins.
RCSP	Residual Cutting Stock Problem. MSSCSP with strongly heterogenous bin-types.
ODP	Open Dimension Problem. Container Minimization Problems such as strip-packing and area minimization problems.

Table 2: Input minimization problems according to Wäscher et al. [157].

2.3 $\mathcal{NP}$-completeness

From a computer science point of view, one of the most important properties of packing problems is that they generally fall in the category of $\mathcal{NP}$-hard problems. A notable exception is the manufacturers pallet loading problem (PLP), which Lins et al. [97] note has never been proven $\mathcal{NP}$-hard. This problem differs from the remaining problems in that it considers identical items to be packed.

For the remaining problems – in multi-dimensional rectangular variant without rotation – proving that they are $\mathcal{NP}$-hard problems may be done easily by reduction from the set partitioning problem. We will give a simple proof by first showing that a special problem, which we refer to as the two-dimensional packing decision problem (2DPDP) is $\mathcal{NP}$-complete and then explaining how the PDP can be reduced to the packing problems we have discussed in the prequel.

We begin by defining the set partition problem which is known to be $\mathcal{NP}$-complete (see [62]).

Definition 1. Set Partition Problem (SPP). *Given a set of items S, each with a positive integer value* $w_i \in \mathbb{N}$ *for* $i \in S$, decide *if we can divide S into two disjoint sets* S' *and* S'' *such that* $S' \cup S'' = S$ *and* $\sum_{i \in S'} w_i = \sum_{i \in S''} w_i = \frac{1}{2} \sum_{i \in S} w_i$.

We now define the two-dimensional packing decision problem.

Definition 2. Packing Decision Problem (2DPDP). *Given a rectangular container of a size* $W \times H$ *where* $W, H \in \mathbb{N}$, *and a set of two-dimensional rectangular items I, each with a width* $w_i \in \mathbb{N}$ *and height* $h_i \in \mathbb{N}$, decide *if a non-overlapping placement, i.e., a packing, of the items exists within the container. Here, a placement* $P : I \to \mathbb{N}_0^2$ *is a map of items into non-negative integer* x, y*-coordinates.*

2DPDP represents the fundamental aspect of packing problems, which is to determine a non-overlapping placement of items within one or more containers.

Theorem 1. *2DPDP is* $\mathcal{NP}$*-complete.*

Proof. Let x_i, y_i be the coordinate of the lower-left corner of each item in a placement. Verifying if a placement of items contains overlap amount to checking if the intersection $([x_i, x_i + w_i[\times [y_i, y_i + h_i[) \cap ([x_j, x_j + w_j[\times [y_j, y_j + h_j[)$ is $\emptyset$ for all $i, j \in I$, $i \neq j$. The open sets ensures that items are allowed to abut. This can be done in time $O(|I|^2)$. Since a placement may be represented by a list of reference-point positions, its can have a size which is polynomial of the input-size, and can therefore be used as a certificate to verify a solution to 2DPDP in polynomial time.

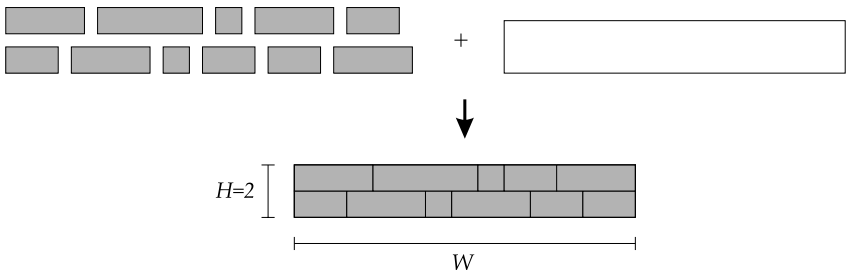


Figure 9: *Illustration of an instance of 2DPDP based on an instance of SPP used to prove that 2DPDP is $\mathcal{NP}$-complete. The height of all rectangles are* 1 *and W is the sum of all the widths divided by two. The solution of the 2DPDP problem, shown on the bottom, is also a solution to the SPP problem with numbers equal to the widths, since it forms two rows of items, each having the same total width.*

Further, given an instance $\mathcal{J}$ of SPP we can create an instance $\mathcal{J}'$ of 2DPDP such that, a yes-answer exists for $\mathcal{J}$ iff a yes-answer exists for $\mathcal{J}'$. To create $\mathcal{J}'$ from an instance of $\mathcal{J}$ with a set of items I, we simply create a set of rectangles each with dimensions $w_i \times 1$ for $i \in S$, and a rectangular container with dimensions $\left(\frac{1}{2}\sum_{i \in S} w_i\right) \times 2$.

If a yes-answer exists for $\mathcal{J}$, we know that two sets S' and S'' exists, and we may create a placement with the items from S' having y-coordinate 0 and the items from S'' having y-coordinates 1. If we set the x-coordinates appropriately this constitutes a non-overlapping placement of the rectangles of $\mathcal{J}'$.

Conversely, if a yes-answer exists for $\mathcal{J}'$, the rectangles must be divided into two groups with respectively y-coordinate 0 and 1. Since the total width of the items is two times the container width, the total width of the items in each group must be equal to the container width, and therefore these two groups represent a partition of S. See Figure 9. □

Since 2DPDP is $\mathcal{NP}$-complete, we can say that the following problems from Table 1 and Table 2 of Section 2.2 are $\mathcal{NP}$-hard: the 2D-rectangular-SKP is $\mathcal{NP}$-hard, since we may transform an instance of 2DPDP to an instance of 2D-rectangular-SKP where we will get a yes-answer to the 2DPDP-instance if and only if all items can be packed in the 2D-rectangular-SKP instance. $\mathcal{NP}$-hardness of 2D-rectangular-MIKP and -MHKP follows from 2D-rectangular-SKP, since we may create instances of 2D-MIKP and 2D-MHKP with a single knapsack. The 2D-rectangular-SLOPP, -MILOPP, and -MHLOPP are $\mathcal{NP}$-hard, since it is trivial to transform each of the 2D-rectangular-SKP, -MIKP, and -MHKP to one of the former.

The 2D-rectangular-SBSBSP is $\mathcal{NP}$-hard, since we may transform an instance of 2DPDP to an instance of 2D-rectangular-SBSBSP where we will get a yes-answer to the 2DPDP-instance if and only if we only need *one* bin in the solution of the 2D-rectangular-SBSBSP. By using the same argumentation as above, we may state that also 2D-rectangular-MBSBPP, -RBPP, -SSSCSP, -MSSCSP, and -RCSP are $\mathcal{NP}$-hard problems.

Finally, rectangular strip-packing is $\mathcal{NP}$-hard, since we may transform an instance of 2DPDP to an instance of 2D-rectangular-ODP where we will get a yes-answer to the 2DPDP-instance, if and only if we can find a container with a width equal to $\frac{1}{2}\sum_{i \in I} w_i$ [1].

Variants of 2D-irregular-SKP, -MIKP, -MHKP, -SLOPP, -MILOPP, -MHLOPP, -SBSBSP, -RBPP, -SSSCSP, -MSSCSP, -RCSP, and -strip-packing where the set of allowed irregular items is a superset of rectangles, are also $\mathcal{NP}$-hard, provided a certificate exists which can be used to verify a solution

[1] The width and height are interchanged here relative to the definition given in section 2.1.4

in polynomial time, since it is trivial to transform the rectangular variants to the irregular variants. Because it can be verified if two polygons overlap in polynomial time, problems involving general polygons (convex, simple, and with holes), are thus $\mathcal{NP}$-hard.

Finally, all d-dimensional variants of the problems listed above are $\mathcal{NP}$-hard for $d > 2$, since the two-dimensional variants can be transformed into d-dimensional variants by setting all remaining dimensions of items to 1. For completion's sake, we note that the 2D-rectangular-area-minimization problem is also $\mathcal{NP}$-hard as proven by Murata et al. [117].

3 Computational Techniques

The computational techniques used to solve packing problems involve methods from computational geometry, operations research, as well as mathematical observations. In this section we will review many of these techniques. As always, space only permits a limited number of contributions to be summarized, and the selection made in this text is purely subjective. The focus is on methods that were either common for several researchers, has returned promising results, or which simply *feel* powerful or universal according to the author of this thesis.

In order to limit the size of the review we will not consider one-dimensional problems or methods for guillotine cutting. While guillotine cutting approaches may have relevance to the topics considered a line much be drawn somewhere and guillotine cutting problems are not within the focus of this thesis.

The section is divided into four parts. First, in Section 3.1, we will consider how the items and container can be represented. Then, in Section 3.2, we will address the problem of determining if two items overlap and how to avoid it. In section 3.3 we discuss many of the known solution strategies. Finally, in Section 3.4 we speculate on possible and likely directions for future research in the field.

3.1 Representing the Items and Container

Up until this point we have not disclosed the true nature of the container or the items involved. Generally, for a d-dimensional problem, items and containers consists of a closed subset of $\mathbb{R}^d$ which may undergo some some sort of rigid transformation (i.e. translation and rotation if allowed).

Most literature deals with items and containers which can be represented by axis aligned rectangles. For a d-dimensional problem that means sets of the form $\Pi_{k=1}^{d}[0,w_k]$, where $w_k \in \mathbb{R}$ is the k-th dimension of the item. Problems involving circles or spheres are modeled similarly.

For other items, we may consider their so-called bounding-box. Assume an item i covers the closed set $s_i \subset \mathbb{R}^d$, then the bounding-box of s_i is defined as the set $\Pi_{k=1}^{d}[x_k^{\min}, x_k^{\max}]$ where:

$$\begin{aligned} x_k^{\min} &= \min\{x_k \mid \mathbf{x} = (x_1,\ldots,x_k,\ldots,x_d) \in \mathbb{R}^d,\ \mathbf{x} \in s_i\} \\ x_k^{\max} &= \max\{x_k \mid \mathbf{x} = (x_1,\ldots,x_k,\ldots,x_d) \in \mathbb{R}^d,\ \mathbf{x} \in s_i\}. \end{aligned}$$

Rectangular shapes are compelling for two reasons. Firstly, many practical situations deal with plates or boxes. Secondly, the use of rectangles instead of arbitrary shapes, simplifies the overlap constraints, which may enable a solution method to find better objective values in less time than what would be possible with a highly detailed description. It should be noted that significant space can be lost around items, if bounding-boxes are used rather than an accurate description.

In two dimensions non-rectangular shapes may take the form of circles, polygons, and polygons with curved boundaries. In three dimensions they may take the form of polyhedra and raster-like models. In general, we refer to arbitrary shapes as irregular.

Polygons are generally described by the sequence of endpoints of the line-segments that make up their boundaries (see Figure 10 (a)). Authors may distinguish between convex polygons, simple (non self-intersecting boundary) polygons, and simple polygons with holes. Endpoints are said to be in either clockwise or counter-clockwise order, and the order determines the interior of the polygon. Polygons with holes are generally represented by a list of sequences, one for each closed boundary.

Polyhedra are described by the linear subspaces (faces) that make up their boundary. The simplest general such definition is a triangle mesh, which is a data-structure composed of a series of triangles often combined with neighboring information (see Figure 10 (c)).

Researchers have also investigated various types of *raster models* (see Figure 10 (b)). An item i which occupy the subset $s_i \subset R^d$, may be represented implicitly by the function $f(\mathbf{p}) : \mathbb{R}^d \rightarrow \{0,1\}$

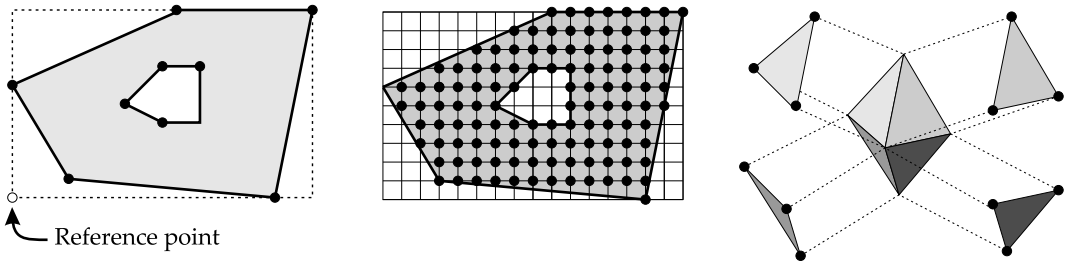


Figure 10: *Polygons and polyhedra. (a) Polygon with hole with its bounding-box. The polygon is represented by the endpoints of the edges (black circles). (b) Raster representation of the polygon. Black circles indicate positions where the raster-function is 1. (c) Polyhedron represented as a number of triangles. Only four triangles shown here. Each triangle may be represented by its endpoints (the black circles).*

where $f(\mathbf{p}) = 1$ if and only if $\mathbf{p} \in s_i$. The raster representation of such an item, is a function defined on a coarser domain. E.g., if only integer coordinates $\mathbf{p} \in \mathbb{Z}^d$ are represented and n_k is the kth dimension of the item then such a grid can be represented by $\Pi_{k=1}^{d} n_k$ bits. It should be noted that several straight-forward compression techniques may use less space. For instance, in two dimensions one may represent the shapes as a series of x-slabs for each y value.

A particular compression technique for three dimensions is an *octree* representation. Octrees are well-known in the area of computer graphics, but we will briefly summarize them here. An octree is a hierarchical tree data structure commonly used to partition three-dimensional space. Each level of an octree divides the previous level uniformly into 8 cubes. Each cube is marked depending on whether the cube is completely inside (black), partially inside (gray) or completely outside the shape (white). The top level of the octree consists of one cube circumscribing the entire shape. This means that level n uses up-to 8^{n-1} cubes. Only gray cubes are sub-divided. A *quad-tree* is a similar data-structure in two dimensions consisting of a hierarchy of squares.

Most recent methods represent irregular items by polygon or polyhedra, and in general when we use the term *irregular* it we will be for problems involving *polygons* or *polyhedra* unless we state otherwise.

The representation of items must be sufficiently detailed to reach high quality solutions without complicating the solution process. Thus the choice of representation is strongly connected to the ability to check for overlap between shapes efficiently which will be discussed in Section 3.2.

3.2 Avoiding Overlap

The two reoccurring constraints for packing problems are that items are not allowed to overlap, and that all items must be positioned within the container.

In other words, let $s_i \subset \mathbb{R}^d$ be the set occupied by item $i \in I$ in a d-dimensional problem, then, in order to ensure that items do not overlap, we require that for any pair of items i and j $\text{int}(s_i) \cap \text{int}(s_j) = \emptyset$, where $\text{int}(\cdot)$ is the interior of the sets. Note that this requirement allows the items to abut. Likewise assume the container covers the set $C \subseteq \mathbb{R}^d$ then we also require that $s_i \cap C = s_i$ for any item i.

We will discuss methods to detect overlap in Section 3.2.1 and present the No fit polygon in Section 3.2.2 and ϕ-functions in Section 3.2.3. These concepts can be used to determine efficiently,

how items must be placed relative to each other to avoid overlap. In Section 3.2.4 we show how a compact non-overlapping placement can be described by a graph that represents how rectangles need to placed relative to each other.

3.2.1 Detecting Overlap

Verification of the two constraints depends on the representation of the items and the container. Circles, and hyper-spheres in general, are particular easy to check for overlap, since the distance between their centers must be larger than the sum of their radii. *Orthogonally placed* rectangles, i.e. with axis aligned sides, are also simple to test for overlap. If we let w_i^k be the size of rectangle i in dimension k, then item i occupies the set $[x_i^1, x_i^1 + w_i^1] \times \ldots \times [x_i^d, x_i^d + w_i^d]$. To ensure that two rectangular items i and j do not overlap, we have to require that either $x_j^k \geq x_i^k + w_i^k$ or $x_i^k \geq x_j^k + w_j^k$ for at least one dimension $1 \leq k \leq d$. In two dimensions these constraints corresponds to the requirement that i must be either left of, right of, above, or below j. We refer to this as a required *relation* between i and j. The procedure becomes a bit more tricky when we deal with irregular items or free rotation of rectangles.

For convex polygons their intersection can be found in $O(n)$ asymptotic time (see e.g. [125, 151]). The intersection of irregular polygons can be determined in time $O(n \log n + k \log k)$ (see [39]), where n is the sum of the number of edges of the input polygons, and k is the number of edges of the output polygon. Since the output polygon can have $O(n^2)$ edges $O(n^2)$ is also a lower bound for the running time of any algorithm that can return the intersection of two polygons, although determining if two polygons intersect may be done quicker.

For raster models the process can be done in linear time of the input size. Let two two-dimensional items i and j have raster-functions f and g represented at integer coordinates. To test if the two items overlap, one may verify if $f(x+x_i, y+y_i) + g(x+x_j, y+y_j) = 2$ for any (x,y) with

$$x \in \{\max(x_i^{\min}, x_j^{\min}), \ldots, \min(x_i^{\max}, x_j^{\max})\}, y \in \{\max(y_i^{\min}, y_j^{\min}), \ldots, \min(y_i^{\max}, y_j^{\max})\}, \quad (3)$$

where $[x_i^{\min}, x_i^{\max}] \times [y_i^{\min}, y_i^{\max}]$ is the bounding-box of item i.

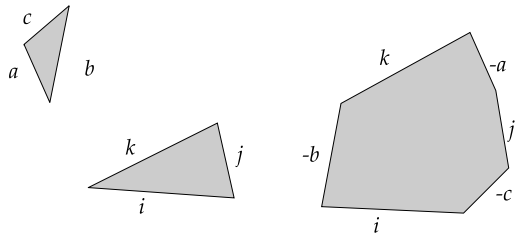
For octree representatons, one may test the cubes recursively. If two black cubes overlap then the shapes overlap. If gray cubes overlap one may recursively consider their higher resolution sub-cubes until the highest resolution is reached, or until only white cubes overlap, in which case the actual shapes do not overlap.

An interesting idea for three-dimensional objects was suggested by Dickinson and Knopf [41, 42] for a heuristic for three-dimensional layout of irregular shapes. Here each of the six sides of the bounding box of each shape is divided into a uniform two-dimensional grid and the distance perpendicular to the box-side from each grid cell to the shape's surface is stored. Determining if two shapes overlap now amounts to testing the distance at all overlapping grid points of bounding box sides, when the sides are projected to two dimensions.

3.2.2 No-Fit Polygon

A popular and efficient way to deal with overlap detection is with the so called *no-fit-polygon (NFP)*. If $s_i \subseteq \mathbb{R}^d$ represents item i, we let $s_i(\mathbf{p}) = \{\mathbf{p} + \mathbf{q} \mid \mathbf{q} \in s_i\}$ be s_i translated by the vector $\mathbf{p}$. For two items i and j which are represented by polygons the NFP describes the set of translations of i and j that will cause the two items to overlap.

If we translate i by $\mathbf{p}_i$ and j by $\mathbf{p}_j$, then $s_i(\mathbf{p}_i) \cap s_j(\mathbf{p}_j) = \emptyset$ if and only if $s_i(\mathbf{p}_i - \mathbf{p}_j) \cap s_j = \emptyset$, and we say that the vector $\mathbf{p}_i - \mathbf{p}_j$ is i's position relative to j. We may refer to set of relative positions between



(a) (b)

Figure 11: *The clockwise sorting principle for NFP generation. (a) Two convex polygons. The edges of the polygons are a,b,c and i,j,k respectively (sorted by slope). (b) The NFP of the two polygons. The polygon with edges a,b,c has been negated to the polygon with edges $-a,-b,-c$. The NFP can now be constructed by visiting the edges sorted by slope which is the following sequence $i,-c,j,-a,k,-b$.*

s_i and s_j that will cause i and j to overlap as their set of *overlap resulting positions*, $ORP(i,j)$, which is defined as follows:

$$ORP(i,j) = \{\mathbf{q} \mid s_i(\mathbf{q}) \cap s_j \neq \emptyset\}. \tag{4}$$

For items in the plane there is an appealing physical approach to calculate $ORP(i,j)$. Cut both items out of cardboard. Position j on a sheet of paper with its reference point in $\mathbf{0}$, and slide the cardboard-model of i around the model of j, such that the boundary of the two polygons abut. As i is slid around j, follow the reference point of i, and draw its trajectory on the paper. The boundary of the figure drawn by the pencil represents the translations of i that will result in i abutting with j and its interior is $ORP(i,j)$.

To do this mathematically we use a concept which is referred to as the Minkowski-sum. The Minkowski-sum of two sets $s_1 \subseteq \mathbb{R}^d$ and $s_2 \subseteq \mathbb{R}^d$ is defined as (see [39]):

$$s_1 \oplus s_2 = \{\mathbf{p}+\mathbf{q} \mid \mathbf{p} \in s_1,\ \mathbf{q} \in s_2\}, \tag{5}$$

where $\mathbf{p}+\mathbf{q}$ is the sum of vectors. Let $-\mathbf{p}$ be the vector where all of $\mathbf{p}$'s coordinates are negated, then for a set s we define, $-s = \{-\mathbf{p} \mid \mathbf{p} \in s\}$.

It can be seen (see [39]) that, $ORP(i,j) = s_i \oplus -s_j$, so the ability to evaluate the Minkowski-sum allows us to determine which relative translations will cause i to overlap with j. Note that, since we generally allow the items to abut in packing problems, we must really define $ORP(i,j)$ from the interior of the sets of the items to get:

$$ORP(i,j) = \text{int}(s(i)) \oplus \text{int}(-s(j)),$$

so that the boundary of the items can overlap.

For two items represented by polygons P and Q we let $\text{NFP}(P,Q) = \text{int}(P) \oplus \text{int}(-Q)$ be their NFP which itself is a polygon although it may contain degenerate elements. Using a method which is quite similar to the physical traversal technique described above, one can determine the NFP for two convex polygons in $O(n)$ time where n is the sum of edges from the two polygons, provided that the points of both the polygons are sorted in clockwise (or counter-clockwise) order. The reason for this, is that the NFP of two convex polygons consists of all the edges of each of the polygons sorted by slope. This is illustrated on Figure 11.

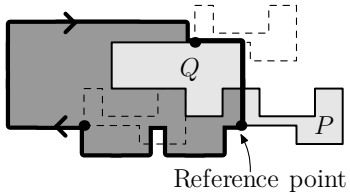


Figure 12: *The NFP of two polygons P and Q. The boundary of the NFP corresponds to the position of P as P is translated around Q.*

Unfortunately, it is not quite as simple when dealing with arbitrary polygons. One of the intrinsic difficulties concerns the ability to position polygons within holes or concavities of other polygons. If one polygon can be fitted exactly into the hole of another, then the translation that leads to such a position, should be represented by a single point within the NFP. The NFP of two polygons is illustrated on Figure 12.

Recent robust methods for calculating the NFP have been introduced by both Bennell and Song [11] and Burke et al. [24]. Both papers divide the methods for calculating the NFP in three different categories: Decomposition techniques, orbiting techniques, and Minkowski-sum techniques. The decomposition techniques consists of methods that break the polygons into convex or star-like components which are easier to handle individually. The orbiting techniques revolve around methods that implement the trajectory strategy that was mentioned in the beginning of this Section. Finally, Minkowski-sum techniques are generalizations of the slope-sorting techniques. The algorithm for generating the NFP by Bennell and Song [11] has worst case running time $O(mn+m^2n^2log(mn))$.

The main problem when dealing with NFPs is that algorithms for their computation are highly complicated which is evident from the fact that robust methods have only begun to appear in the last few years. Another draw-back of NFPs is that they must be computed for each pair of polygons that one wishes to test for intersection. This computation is generally done in a preprocessing step and will result in a number of NFPs which is quadratic in the number of input polygons, which requires both computational time and storage. Additionally, if items can be rotated, NFPs for each pair of allowed rotation angles and each pair of polygons are required.

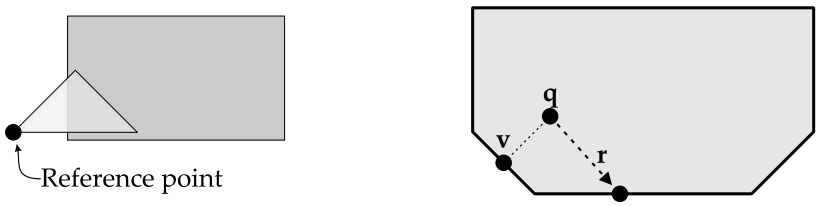
It should be noted though, that the recent methods of Burke et al. [24] and Bennell and Song [11] are capable of calculating between hundreds and thousands of polygons per second on commodity hardware. The largest set of NFPs reported by Burke et al. [24] consists of 90,000 NFPs which are calculated in 142 seconds on a Pentium 4 2 GHz based on an instance consisting of 300 polygons. However, NFPs for the most common benchmark instances involving polygon shapes are processed within 1-5 seconds, so the preprocessing time may be negligible with todays hardware.

To solve the problem of determining the NFPs the EURO Special Interest Group on Cutting and Packing (ESICUP), have made the NFPs for the commonly used benchmark data set for strip-packing problems with polygons available along with the data-set on their website ([2]). This way, researchers are spared the trouble of implementing algorithms for NFP-generation.

Once the NFP of two polygons P and Q has been determined, one can determine if they overlap if placed at the position $\mathbf{p}$ and the position $\mathbf{q}$ respetively, by evaluating if their relative position $(\mathbf{p}-\mathbf{q})$ is within NFP(P,Q). This amounts to using a simple point-in-polygon tests such as *ray-shooting* (see e.g. de Berg et al. [39]) which can be done in linear time of the size of NFP(P,Q).

If the polygons overlap, we may determine a minimal two-dimensional translation of one of the polygons, which will result in a non-overlapping relative position. This is referred to as the *penetration*

[2] `http://paginas.fe.up.pt/~esicup/`



(a) (b)

Figure 13: *(a) An overlapping triangle and retangle. (b) Their NFP.* $(\mathbf{p}-\mathbf{q})$ *is the relative position of the two polygons.* $\mathbf{v}$ *is the closest point to* $(\mathbf{p}-\mathbf{q})$ *on the boundary and the translation corresponding to* $\mathbf{v}-(\mathbf{p}-\mathbf{q})$ *is the minimal translation of the triangle required to ensure that the two polygons do not overlap. The length of this is the intersection or penetration depth. The intersection of the line in direction* $\mathbf{r}$ *from* $(\mathbf{p}-\mathbf{q})$ *determines the amount required to move the triangle in direction* $\mathbf{r}$ *so that the two shapes do not overlap.*

depth. Given a relative position $(\mathbf{p}-\mathbf{q})$ we can calculate the penetration depth of polygons P and Q in linear time in the size of the NFP by evaluating the length of $\mathbf{v}-(\mathbf{p}-\mathbf{q})$ where $\mathbf{v}$ is the point nearest to $(\mathbf{p}-\mathbf{q})$ on the boundary of NFP(P,Q). Given a direction $\mathbf{r}$, we may also use the NFP to determine the amount we need to translate P along $\mathbf{r}$ for the two polygons to not overlap (see Figure 13 (a)). This is done, simply by shooting a ray from $(\mathbf{p}-\mathbf{q})$ along $\mathbf{r}$. The first intersection between the ray and the boundary of NFP(P,Q) is the nearest non-overlapping translation along $\mathbf{r}$ (see Figure 13 (b)).

According to Bennell and Song [11] some researchers have come to feel that the introduction of the NFP has stifled research in non-overlapping techniques, but NFPs describe non-overlapping positions between two polygons which is a fundamental aspect of solving packing problems involving polygons. So it seems, that NFPs by their very definition, are hard to overcome.

Theoretically, the notion of an NFP can be generalized to higher dimensions. Minkowski-sums of polyhedra have certainly been investigated (e.g. by Varadhan and Manocha [154] and Zhong and Ghosh [163]), but suffer from the fact the complexity of the output, i.e. the number of edges, faces, and vertices, can be $O(n^3m^3)$ (see [139]) for two polyhedra with complexity n and m respectively. At this time, methods for packing three-dimensional irregular shapes are still in their infancy, but it would be interesting to examine the potential of a three dimensional NFP (No fit polyhedron).

3.2.3 ϕ-functions

A concept which is related to the NFP is the notion of ϕ-functions which were introduced by Stoyan and Ponomarenko [141] and investigated for two- and three-dimensional problems by Stoyan et al. [142, 143, 144, 145].

For two d-dimensional sets $s_1 \subseteq \mathbb{R}^d$ and $s_2 \subseteq \mathbb{R}^d$, a ϕ-function for s_1 and s_2 is a function $\phi : \mathbf{R}^{2d} \rightarrow \mathbf{R}$, for which we require the following:

- $\phi(\mathbf{p},\mathbf{q}) > 0$ for $s_1(\mathbf{p}) \cap s_2(\mathbf{q}) = \emptyset$.
- $\phi(\mathbf{p},\mathbf{q}) = 0$ for $\text{int}(s_1(\mathbf{p})) \cap \text{int}(s_2(\mathbf{q})) = \emptyset$ and $s_1(\mathbf{p}) \cap s_2(\mathbf{q}) \neq \emptyset$ (i.e. s_1 and s_2 have been translated such that they abut).
- $\phi(\mathbf{p},\mathbf{q}) < 0$ for $\text{int}(s_1(\mathbf{p})) \cap \text{int}(s_2(\mathbf{q})) \neq \emptyset$.

ϕ-functions can be perceived as a measure of distance between $s_1(\mathbf{p})$ and $s_2(\mathbf{q})$, and this is used several times by Stoyan et al. [143]. For instance, the ϕ-function of circles and spheres is defined as the distance between their centers minus the sum of their radii. Stoyan et al. [143] also give a definition of a ϕ-function for two convex polygons, such that $\phi(\mathbf{p},\mathbf{q})$ is the distance from $(\mathbf{p}-\mathbf{q})$ to the boundary of their Minkowski-sum. This can be done by considering the minimal signed distance from $(\mathbf{p}-\mathbf{q})$ to any of the infinite lines that are coincident with the edges of the convex Minkowski-sum.

ϕ-functions are adequately abstract to be used to detect overlap by many different simple shapes (circles, spheres, rectangles, cylinders, regular polygons, and convex polygons) and combinations which may include unions and disjunctions. For instance, the ϕ-function of two non-convex polygons, may be represented by breaking them into convex subparts and combining the set of ϕ-functions which comes from each pair of convex subregion.

We may percieve ϕ-functions as an implicit representation of the Minkowski-sum of two-sets, since the set $\{\mathbf{q} \in \mathbb{R} \mid \phi(0,\mathbf{q}) = 0\}$ is the boundary of the Minkowski-sum. Further, since any relative position of the two sets is mapped into a real value, which is less than 0 only for overlapping positions, we may use the value of the ϕ-function to indicate the amount the two sets overlap. Conversely, since we are generally interested in compact placements, we may also use positive values to indicate that the two sets can be moved closer.

While recent research has consistently referred to ϕ-functions as "promising", they have yet to prove competitive with conventional methods for objects other than circles and spheres, but this could be due to the relatively simple optimization techniques applied thus far.

Further discussions of ϕ-functions are beyond the scope of this text, but, because they are an implicit form and a multi-dimensional generalization of NFPs to a variety of shapes and combinations hereof, they may provide many advantages over NFPs as the researchers familiarize themselves with the concept in the coming years.

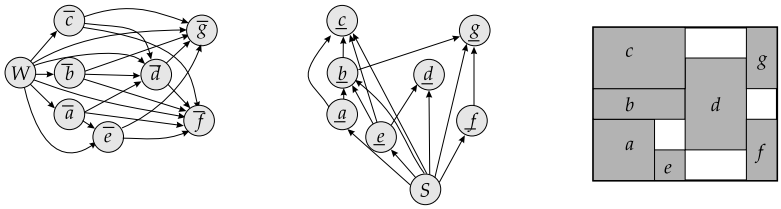
3.2.4 Constraint Graphs

Constraint graphs are a set of directed acyclic graphs width edge-weights, which can be used to describe the relative position of rectangular items. For a d-dimensional problem, d graphs can be used to describe the position of each rectangular item – One graph for each dimension.

For instance, to model constraint graphs for a two dimensional placement two graphs G_h and G_v are needed. For both graphs a node is added for each rectangle, and we name the nodes of a rectangle a, $\overline{a}$ in G_h and $\underline{a}$ in G_v. Further, in G_h a node W (west) is added, while in G_v a node S (south) is added. Now directed edges are added between the nodes which describe their relative position. An edge from $\overline{a}$ to $\overline{b}$ in G_h indicates that rectangle a must be placed left of b, while an edge from $\underline{a}$ to $\underline{b}$ in G_v indicates that a must be placed below b.

A directed edge from W is added to each node in G_h and a directed edge from S to each node is added in G_v. The weights of these edges are all set to 0. Edges are added between all pairs of rectangles such that for any two rectangles a and b, exists exactly one edge from $\overline{a}$ to $\overline{b}$, $\overline{b}$ to $\overline{a}$, $\underline{a}$ to $\underline{b}$, or $\underline{b}$ to $\underline{a}$. The edges indicate that a is to be placed left, right, below, or above b, respectively. In G_h each edge $(\overline{a},\overline{b})$ is given weight equal to the width of rectangle a. In G_v the weight of each edge $(\overline{a},\overline{b})$ is set to the height of rectangle a.

The graphs can be used to generate the coordinates of each rectangle in a placement which is compact in the sense that all rectangles have minimal x- and y-coordinate as described by edges from the constraints of the graphs. To determine the x-coordinate of a rectangle a one only needs to determine the longest path (or the critical path) from W to $\overline{a}$ in G_v which can be done in $O(n^2)$ time, where n is the number of rectangles, since the graph is acyclic (see e.g. [36]). Similarly, one can



(a) (b) (c)

Figure 14: *Constraint graphs for two-dimensional placement of rectangles a, b, c, d, e, f and g.. (a) G_h. (b) G_v. (c) Resulting placement. Edge weights have been removed but are respectively width and height of the rectangles for G_h and G_v.*

determine the longest path from S to $\underline{a}$ to determine its y-coordinate. Furthermore one may determine the position of *all* rectangles in $O(n^2)$ time. The construction of each graph and determination of the item positions is straight-forward to generalize to higher dimensions.

Figure 14 illustrates this concept. Here the x-coordinate of g is equal to the longest path from W to $\overline{g}$ in G_h which goes through $\overline{b}$ and $\overline{d}$, while the y-coordinate is equal to the length of the longest path from S to $\underline{g}$ in G_v which goes through $\underline{a}$ and $\underline{b}$.

An important aspect of constraint-graphs is that, as long as an edge exists between the nodes of any two rectangles a and b in any of the graphs, the placement is guaranteed to be without overlap. This comes from the fact that such an edge describes one of the required *relations* (see Section 3.2) between the two rectangles.

3.3 Solution Approaches

In this section we present many popular solution approaches. An issue which have not covered previously is with respect to rotation. Most solution methods are not used to solve problems where items are allowed to rotate. This is especially true for exact algorithms; most likely because enumeration schemes and bounds are not powerful enough to manage rotated items. We will not dwell any further on this topic but simply remark that heuristics which do consider rotation, generally only consider 180°, or multiples of 90°, rotation of items.

Solution approaches for packing problems fall into many different categories. We will begin by introducing Mixed Integer Programming Formulations in Section 3.3.1, and proceed to consider some of the simpler paradigms in sections 3.3.2, 3.3.3, and 3.3.4. A strategy for representing free-space will be considered in Section 3.3.5. Methods which consider overlap of items during the solution process are discussed in Section 3.3.6. The notion of envelopes which is the core element of methods that *constructs* a placement by positing one item at a time is discussed in section 3.3.7. In Section 3.3.8 non-trivial representations of placements are presented. Advanced techniques for recent exact algorithms are presented in sections 3.3.9, 3.3.10, and 3.3.11. Finally, the current state of the field of approximation algorithms is summarized in Section 3.3.12.

3.3.1 MIP Formulations

Since packing problems are optimization problems, an obvious choice for modeling the problem is through *mixed integer programming* (MIP). A number of MIP formulations for packing problems have been considered over the years. Here we restrict ourselves to consider three such models for

orthogonal rectangular packing and a single model for irregular packing. MIP models are generally used for exact algorithms.

Beasley [8] introduced a model for two-dimensional knapsack packing problems. However, the model is simple to generalize to higher dimensions. In this model an integer variable is allocated for each item and each location throughout the container area. Specifically, we set

$$x_{ipq} = \begin{cases} 1 & \text{if item } i \text{ is placed with its bottom} - \text{left corner at } (p,q) \\ 0 & \text{otherwise} \end{cases}$$

The coefficient matrix A is a form of "occupancy" matrix that describes which cells are occupied for each item location and we set

$$a_{ipqrs} = \begin{cases} 1 & \text{if grid} - \text{cell } (r,s) \text{ is occupied when} \\ & \text{item } i \text{ is placed with its bottom} - \text{left corner at} (p,q) \\ 0 & \text{otherwise} \end{cases}$$

Beasley considered a knapsack packing problem with the profit of each type of item i set to v_i, and required that the number of packed items of type i should be between P_i and Q_i. The model is as follows:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} v_i x_{ipq} \\ s.t. \quad & \\ & \sum_{i=1}^{n}\sum_{j=1}^{W}\sum_{k=1}^{H} a_{ipqjk} x_{ipq} \leq 1 \qquad (p=1,\ldots,W)\ (q=1,\ldots,H) \\ & \sum_{p=1}^{W}\sum_{q=1}^{H} x_{ipq} \geq P_i, \quad (i=1,\ldots,n) \\ & \sum_{p=1}^{W}\sum_{q=1}^{H} x_{ipq} \leq Q_i, \quad (i=1,\ldots,n) \end{aligned}$$

$$x_{ipq} \in \{0,1,2\ldots\}\ \ (i=1,\ldots,n),\ (p=1,\ldots,W),\ (q=1,\ldots,H)\ ,$$

where the first type of constraint ensures that each location is filled by only one item, and the following two ensures that the required number of items are packed.

This formulation contains a massive number of integer values, but Beasley introduced a way to reduce this number, by considering possible positions for each item i based on the dimensions of the other items. However, even with this reduction, the model still suffers from an exponential number of integer variables which depends on the container dimensions.

Despite the large number of binary variables, Beasley [8] was actually able to solve many problems to optimality, by using a combination of lagrange relaxation, sub-gradient search and tree-search.

An advantage of the model by Beasley [8] is that it can be used to model any shape provided that the shape can be represented accurately using a grid-structure. In a way, one may therefore view this model as an integer formulation of the raster model used to detect overlap (see section 3.2.1).

A similar model was later used by Hadjiconstantinou and Christophides [74], but unlike the model of Beasley [8], which used an integer variable for each combination of item and x- and y-coordinate, Hadjiconstantinou and Christophides [74] uses one binary variable for each item and each x-coordinate and one binary variable for each item and each y-coordinate. This model was also used for exact methods in conjunction with lagrange relaxation, sub-gradient search and tree-search.

The model by Beasley [8] suffers from the fact that an exponential number of variables that depend on the dimension of the container are used. A model for rectangular packing that uses only a polynomial number of binary variables in the number of items was presented by Onodera et al. [124] and Chen et al. [29]. For each rectangle a linear variable $x_{i,k}$ is used to describe the coordinate of each rectangle i in dimension k. To ensure that two rectangles do not overlap two binary variables $l_{i,j,k}$ and $l_{j,i,k}$ are introduced for every pair of items $i, j \in I$ in each dimension $1 \leq k \leq d$. The model (without objective function) looks as follows:

$$\begin{array}{rcll} x_{i,k} - x_{j,k} + W_k l_{i,j,k} & \leq & W_k - w_{i,k}, & 1 \leq k \leq d,\ i,j = 1,\ldots,n \\ x_{j,k} - x_{i,k} + W_k l_{j,i,k} & \leq & W_k - w_{i,k}, & 1 \leq k \leq d,\ i,j = 1,\ldots,n \\ \sum_{j=1}^{n} l_{i,k,j} + \sum_{j=1}^{n} l_{k,i,j} & \geq & 1, & 1 \leq k \leq d,\ i = 1,\ldots,n \\ x_{i,k} + w_{i,k} & \leq & W_k & 1 \leq k \leq d,\ i = 1,\ldots,n \end{array}$$

$$x_{i,k} \geq 0,\ l_{j,k},\ l_{i,j} \in \{0,1\} \text{ for } i,j = 1,\ldots,n,\ k = 1,\ldots,d \qquad ,$$

where $l_{i,k,j} \in \{0,1\}$ and W_k is the kth dimension of the rectangular container. The two first constraints ensures that for every pair of rectangles i and j, the kth coordinate of i meets the requirement that $x_{i,k} + w_{i,k} \leq x_{j,k}$ if $l_{i,j,k} = 1$ and vice versa if $l_{j,i,k} = 1$. If just one of any of $l_{i,j,k}$ or $l_{j,i,k}$ are equal to one for $k = 1,\ldots,d$ then rectangles i and j cannot overlap, and the third constraint ensures that this is true for at least one dimension k. The last constraint ensures that all rectangular items are placed within the container boundaries. The model is general and can be used for different packing problems, but the formulation is likely not suitable for branch-and-bound based algorithms, where the *LP*-relaxation is used in each node, since roughly n^2 binary variables must be set to 1 to avoid overlap. Onodera et al. [124] use the model for an exact branch-and-bound algorithm for the minimum area rectangular packing problem, while Chen et al. [29] use it for the container loading problem. In both cases the authors report experiments for only around six items.

The first MIP formulation for two- and higher dimensional packing problems is commonly credited to Gilmore and Gomory [69]. The formulation is based on a principle which had proven successful for one-dimensional cutting-stock problems using column generation (see Gilmore and Gomory [67, 68]), and is commonly used to introduce column generation to students. Their strategy for the one-dimensional problem is to enumerate all possible cutting patterns of the stock, i.e. is all possible feasible placements of items. If we let A_j describe the jth cutting pattern, then we set $a_{ij} = 1$ if item i belongs to the jth cutting pattern and $a_{ij} = 0$ otherwise. Let M be the number of feasible cutting patterns, then all the feasible cutting patterns can used for columns in a $n \times M$ matrix A. Since the problem to be solved is a one-dimensional bin-packing problem, the objective is to minimize the number cutting patterns required in order for all items to be cut. The full formulation may now be written as:

$$\begin{aligned} & \min \sum_{i=1}^{M} x_i \\ s.t. & \\ & \textstyle\sum_{i=1}^{M} a_{ij} x_j = 1 \qquad (i = 1,\ldots,n) \\ & x_j \in \{0,1\} \qquad (j = 1,\ldots,n), \end{aligned}$$

where x_j is a binary variable indicating if pattern j is used or not and the constraints ensures that all items are cut exactly one time. This model is easily generalizable to higher dimensions, since each column A_j may simply represent any feasible d-dimensional cutting pattern.

Unfortunately, even for one dimension the number of cutting-patterns is prohibitively large, so rather than representing all of them in A, Gilmore and Gomory [67, 68] generate the columns dynamically. To do this one must solve a knapsack problem, that finds a feasible pattern with maximal reduced cost. For the one-dimensional case, the knapsack problem is one-dimensional, and easy to solve using dynamic programming (see e.g. [90]). The dimension of the associated knapsack problem follows the dimension of the primary problem, and so for higher-dimensional problems one must solve a higher-dimensional knapsack packing problem. This problem is not quite as simple to manage as the one-dimensional case, since the dynamic programming approach that works so well for one-dimensional problems cannot be used for higher dimensions. However, this can be done with one of the two first models (see also [130, 131]).

So far we have only considered MIP models for orthogonal rectangular items. MIP models for the strip-packing problem with polygons were introduced by Daniels et al. [38]. The models are based on NFPs and the principle that the relative position of two polygons, must be outside the NFP (see Section 3.2.2). For a pair of polygons this is modeled by requiring that the the relative position of the polygons is located within one of the convex subregions that make up the set of feasible locations which is the complement of the NFP. Ensuring that a relative position is within a convex subregion can be done with a set of linear constraints. Ensuring that it is within exactly one is done by introducing a binary variable for each subregion. Daniels et al. [38] considered the strip-packing problem, but because of the large number of binary variables, they were unable to solve realistic problems with more than 6-9 polygons to optimality.

The MIP formulation by Daniels et al. [38] were used by Li and Milenkovic [96] to introduce *Compaction and separation* techniques for the strip-packing problem. The compaction procedure starts from a non-overlapping placement, and a linear-program formulation is used to determine a set of translational vectors of the polygons, which describes how the polygons should be translated to reach a feasible solution with less strip-length. NFPs are used to determined the constraints of the linear programming formulation, such that the resulting translations constitute a feasible placement. The method is similar to solving the MIP described by [38] with all integer values fixed to match the current placement.

Since only neighboring polygons are used to determine a set of linear programming constraints in the original placement, and because the constraints generated depend on the relative position of two neighboring polygons, the translated placement can give rise to a different linear programming formulation with a new set of constraints. Therefore Li and Milenkovic [96] repeats the process a number of times until the constraints of two consecutive placements are equal which is considered a local minimum of the compaction problem.

A similar method is presented for *separation* of the polygons, i.e. transform a placement with overlap to one without overlap such that its strip-length is minimal. Li and Milenkovic [96] use the compaction and separation technique combined with a database of of good solutions to solve the strip-packing problem for polygons.

3.3.2 Levels

Heuristics by Baker and Schwarz [4], Berkey and Wang [12], Chung et al. [31] and Lodi et al. [100] employ a principle based on levels or "shelfs" for two-dimensional problems. The strategy is to fill the container area row by row. First a row which begins in the lower left corner of the container area is filled by positioning rectangles one-by-one from left to right. No rectangle may be positioned above any currently placed rectangle. Once no more items can be placed at the bottom of the container, that row is full, and the next row, which starts on top of the tallest item of the first row, is filled. Each row is

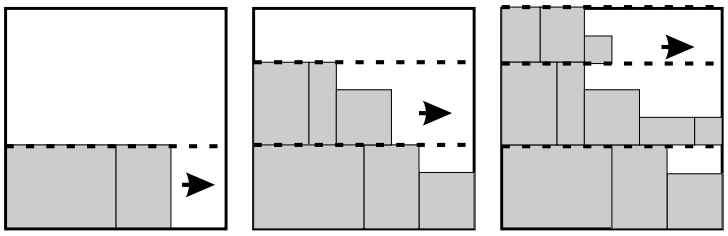
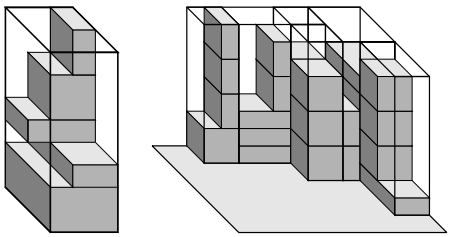


Figure 15: Levels *or* Shelves. *The container is filled from left to right in row, with the rectangles of a row "standing" on the bottom of that row. The height of a row is equal to the height of the tallest item.*



(a) (b)

Figure 16: *Stacks. (a) A stack of 5 items. (b) A placement consisting of 6 stacks.*

completely flat and rectangles cannot be placed above each other within the same row. By repeatedly filling rows, one will eventually have placed all items or reach the top of the container. Authors refer to the rows as "*levels*" or "*shelves*". The procedure is illustrated on Figure 15.

Because of their simplicity, level-based algorithms are easy to analyze and bounds and exact algorithms for problems constrained to level-based packing have recently been investigated by Lodi et al. [104] and Bettinelli and Ceselli [13]. These methods are based on column generation, which level-based packing is particular suitable for, since each set of items in a level may be represented by a column just as cutting patterns of the one-dimensional cutting stock problem were represented by columns in the column generation technique by Gilmore and Gomory [67, 68] (see 3.3.1). Level-based packing is also commonly used for approximation algorithms (see Section 3.3.12). For other older studies of level packing see the papers by Coffman et al. [35] and Frenk and Galambos [60].

3.3.3 Stacks, Layers, and Walls

Stacks, layers and walls are generalizations of shelf-packing to three dimensions. Gilmore and Gomory [69] arrange boxes in *stacks* that are placed on the bottom of container and fill its height (see 16). Once a suitable set of stacks have been determined, one can solve the three-dimensional problem by solving the two-dimensional problem where the position of each stack must be determined. This principle was also used for a heuristic approach based on Genetic Algorithms by Gehring and Bortfeldt [64].

An advantage of stacks is, that since each item is supported by only one item, one can ensure global stability of the items simply by ensuring that the largest items are placed at the bottom of the stack and that higher items do not extend beyond the top of lower boxes.

Stacks reduces the three-dimensional problem to a set of one-dimensional problems and a single two-dimensional problem. An alternative is to build a set of layers in the height of the container, where the placement of items in each layer is determined by solving a two-dimensional problem, and no item

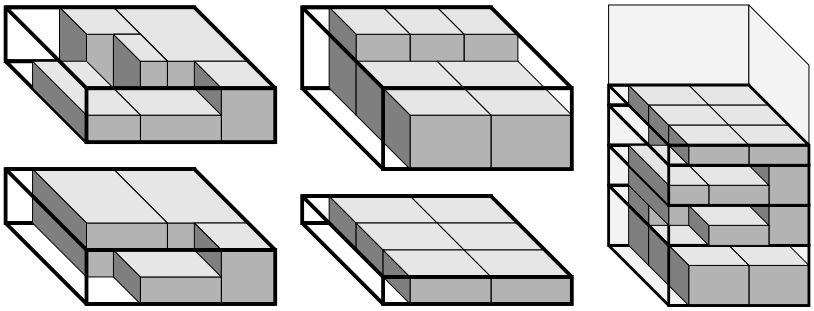


Figure 17: *Layers. (a) 4 layers. (b) A placement consisting of the four layers placed on top of each other.*

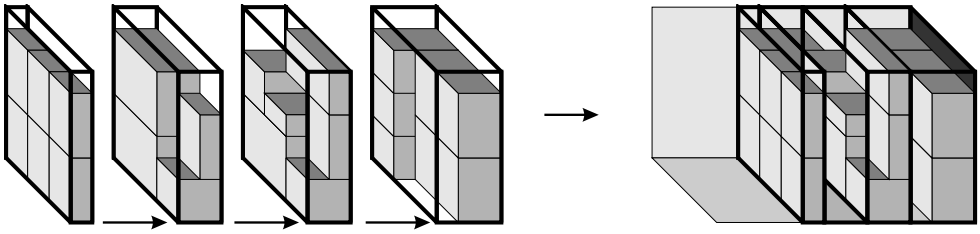


Figure 18: *Wall-building. 4 Walls are constructed and placed in a container.*

can be above any other item. The height of each layer may be set to the height of the tallest box within it. A tabu-search based heuristic for three-dimensional bin-packing based on layers was presented by Lodi et al. [103]. Despite of the fact that this method is based on layers, results are surprisingly comparable to the results of the method by Faroe et al. [53] which consider free placement of items (see section 3.3.6).

In container loading problems, the largest dimension is the depth of the container. Therefore the "layers" are constructed using the width W and height H build in the depth of the container. Because these layers are standing on the container floor as a set of walls which fill the container in its depth (see Figure 18), this process is called *wall-building* and was introduced by George and Robinson [66].

To determine the depth of each wall, authors begin by selecting a "layer-defining-box" (LDB) that fixes the depth of the wall to the depth of the box. Generally wall-building approaches rely heavily on selection of the LDB and efficient strategies for packing each wall. Bischoff and Marriott [16] compare different ranking functions for the LDB, but does not determine any clear winner, and therefore recent methods use wall-building in conjunction with some form of meta-heuristic method. Examples include the heuristics by Bortfeldt and Gehring [18], Gehring and Bortfeld [63], and Pisinger [128] which are based on genetic algorithms, tabu-search, and tree-search respectively.

To fill the individual walls authors either solve a simpler three-dimensional packing problem such as Gehring and Bortfeld [63] or a two-dimensional packing problem. George and Robinson [66] solve a two-dimensional packing problem by placing items in shelves and Pisinger [128] divides the wall recursively into horizontal and vertical strips and each strip is packed by solving a one-dimensional knapsack problem, which can be done efficiently in pseudo-polynomial time (see Figure 19).

Wall- and layer building strategies have the clear advantage that they reduce an otherwise hard problem into simpler sub-problems. However, the most important disadvantage is that space is lost between walls, if the boxes cannot fully utilize the depth of a wall. This problem is somehow coun-

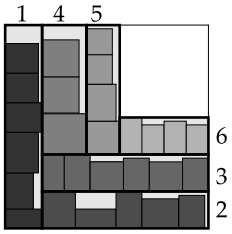


Figure 19: *When filling each wall during wall-building Pisinger recursively fills the wall with horizontal and vertical strips using tree-search. Here six strips are used; first a vertical, then two horizontal strips, then two vertical strips and finally one horizontal strip.*

tered by the fact that container-loading problems often contain hundreds of smaller homogeneous boxes, but wall-building is very likely to fail for a smaller set of large items.

3.3.4 G4 structures

An interesting divide and conquer structure dubbed G4 was presented by Scheithauer and Terno [137]. If the container is partitioned either horizontally or vertically, then the two smaller packing problems can be solved and used to form a solution to the entire problem. This can be done recursively, however, only placements that are guillotine cuttable (see Section 2.1.2) may be considered. To remedy this situation Scheithauer and Terno [137] introduced a third possibility in addition to horizontal or vertical partitions. The third possibility is expressed with the G4-structure depicted on Figure 20 (a), which divides the container into four compartments. If we let $n(W,H)$ denote the maximal number of items on a plate of size $W \times H$ using a G4-structure, then $n(W,H)$ can be expressed with the recursion

$$n(W,H) = \max_a \max_b \left\{ n(a,b) + \max_e \left\{ n(c,d) + \max_f \{ n(e,f) + n(g,h) \} \right\} \right\},$$

with $a,\ldots,h$ as indicated on Figure 20 (a). It should be noted that the values of $a,\ldots,h$ can be chosen from a subset of the values $1,\ldots,W$ and $1,\ldots,H$ which depend on the item dimensions. This recursion may be calculated efficiently using dynamic programming and forms the basis of the heuristic by Scheithauer and Terno [137] for the pallet loading problem where there is only one item type. It was later used for heuristics for multi pallet loading and container loading problems by Scheithauer and Sommerweiss [136] and Terno et al. [149].

Although G4 structures seem versatile, they are not capable of represent all non-guillotine cuttable placements. An example of one such placement is illustrated on Figure 20 (c).

3.3.5 Representing Free Space

One of the intrinsic problems when filling a three-dimensional container is to represent the residual space efficiently. So far we have touched upon simple strategies such as wall- and layer-building where this is trivial since the residual space can be represented by a single container.

Eley [50] considered three-dimensional rectangular placement in a more free-form manner. Here residual space is represented as a set of overlapping boxes. Initially the residual space is the complete container. The first item is positioned at the lower left back corner of the container, and three overlapping boxes which represent the residual space are created; one describes the full space above, one the space in front of, and one the space right of the item.

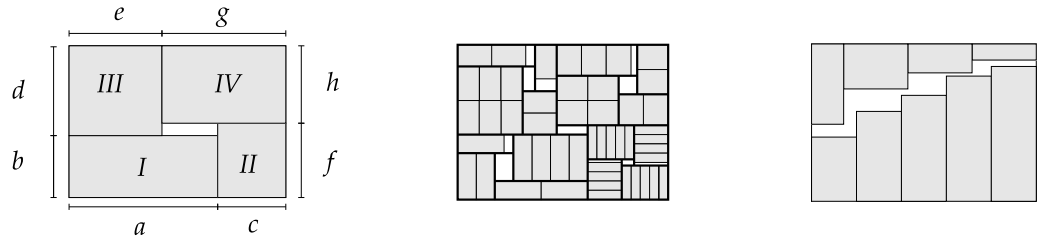


Figure 20: *(a) G4 structure consisting of 4 blocks. (b) Placement using the G4 structure with four item types. (c) Placement which cannot be represented by a G4 structure.*

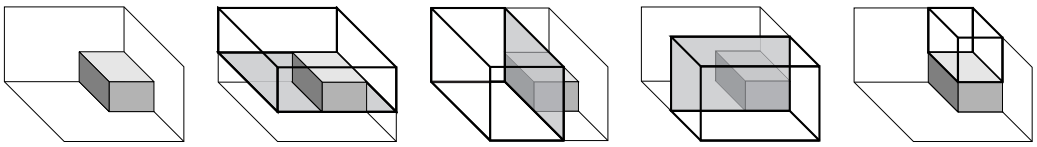


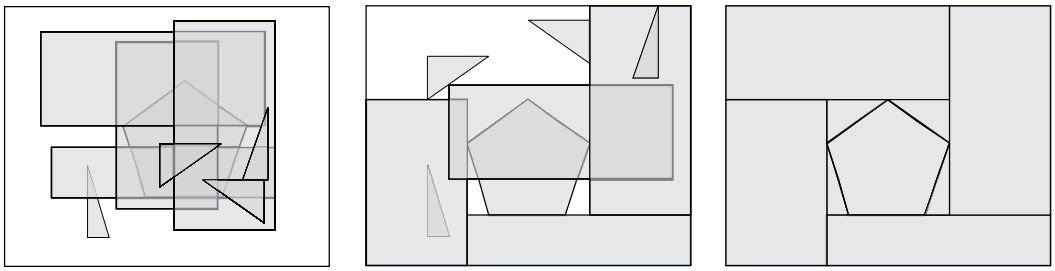
Figure 21: *Eley's space representation. (a) A single item is positioned in the container. (b,c,d) The three residual spaces generated with the insertion of the item. (e) Alternative to the residual space above the item, which will ensure that any new item positioned above, will be place at a stable position.*

The following items are position one by one. Each item is positioned in a residual space, and each of the residual spaces it overlaps with are divided into new residual spaces that describe the area surrounding the item. An item may only be placed in a residual space which is large enough to contain it. As the placement progresses residual spaces may be merged to reduce their number. The process is demonstrated on Figure 21.

An important part of this process is that stability can be ensured by limiting the residual space above items. When new items are positioned, the smaller residual space ensures that they do not extend beyond the top dimensions of an underlying item. Since residual spaces are merged together, entire planar areas can be constructed, so that larger items can be positioned above a group of smaller ones. Eley [50] integrated this method in a variant of tree-search which is often referred to as the pilot method.

Alvarez-Valdes et al. [1, 2] presented GRASP and tabu-search based heuristics for two-dimensional knapsack problems with a strategy similar to the one by Eley [50]. However, here the residual spaces does not overlap, but are simply merged to create large regions.

Several authors have suggested alternative ways to represent free space in the container. Ngoi et al. [118] use a matrix for each cross section in the height of the container to represent free space. The matrices are not a complete discretization of the container volume, but represents a non-uniform grid-structure with cells for every x- and z-coordinate of every corner of the placed boxes between two consecutive y-coordinates. Bischoff [15] later simplified this apporach by representing the available height for every location with just one matrix.



(a) (b) (c)

Figure 22: *Relaxed placement. (a) A placement with overlap. (b) Overlap has been reduced significantly by moving items. (c) A feasible placement without overlap.*

3.3.6 Relaxed Placement Methods

Although solutions require that there is no overlap of items, several methods where overlap is allowed during the solution process have been investigated in the literature. This is especially the case for problems involving irregular items. Commonly, these methods either include overlap in the objective function or attempt solely to solve the decision variant of the packing problem, i.e. find a feasible placement given a set of items and container dimensions. The procedure works by iteratively reducing overlap. This can be implemented within a local search framework, where one may simple change the coordinates of one or more items, evaluate the overlap of the new placement, and accept it if it contains less overlap. The procedure is illustrated on Figure 22

It should be noted that although the area of overlap of two items in the placement, is one of the most obvious ways to determine the amount of overlap, overlap can be measured in a variety of different ways (e.g. intersection depth) some of which are presented in Nielsen [120].

Heuristics that employ this principle for two-dimensional problems involving irregular shapes were investigated by Heckmann and Lengauer [76], Jain and Gea [85], Jakobs [86], Lutfiyya et al. [105], Oliveira and Ferreira [122], Theodoracatos and Grimsley [150] and Bennell and Dowsland [10]. Most noteworthy are the methods by Oliveira and Ferreira [122] and Heckmann and Lengauer [76] where overlap is removed by simulated annealing. Oliveira and Ferreira [122] allow random moves of items and use a raster model to evaluate the amount of overlap. Heckmann and Lengauer [76] solves the problem in four phases, where the first phases consider crude representations of items, while the later phases consider finer representations based on polygons. Also the distance items can move is determined by the temperature of the simulated annealing.

Recent methods for irregular packing in two dimensions have abandoned the raster models in favor of polygons. Bennell and Dowsland [9] and Gomes and Oliveira [73] consider translations of items which result in overlap, but use the compaction and separation techniques described in Section 3.3.1 to remove overlap. Bennell and Dowsland [9] apply the separation techniques whenever the overlap climbs above a certain threshold level while Gomes and Oliveira [73] conduct separation and overlap removal after each exchange of shapes.

A relaxed placement heuristic for rectangular packing was used by Faroe et al. [53] for the two- and three-dimensional bin-packing problem. The number of bins are minimized by starting with a large number of bins that are iteratively reduced. For each number of bins a decision problem which ask for a feasible solution is solved. To solve the decision problem, the heuristic iteratively reduces overlap by translating items either horizontally or vertically. Rather than considering overlap for single

positions, all horizontal or vertical translations are considered when items are moved and the position with the least overlap is chosen. This is done efficiently using an algorithm with asymptotic time that is polynomial in the number of items. The technique has also been applied to four of the papers in the thesis ([A],[B], [F], and [E]), for strip-packing of polygons and polyhedra, and for placement of cylinders with spherical ends. A heuristic based on the procedure from [A] for strip-packing of polygons was later developed by Umetani et al. [152], but unlike the method by Egeblad et al. [A] which does not use any form of preprocessing, the method in [152] uses NFPs to calculate the overlap.

Another relaxed placement heuristic by Imamichi et al. [83] takes a more global approach. Here overlap is calculated as the sum of intersection depths (see Section 3.2.2) of pairs of overlapping polygons. Overlap is iteratively removed by moving the placement in the direction of the gradient of the objective function, i.e., each item is translated in the direction that determines its minimal penetration depth. Once overlap has been removed two items are swapped to generate a new overlapping placement, which is then used to find yet another non-overlapping placement.

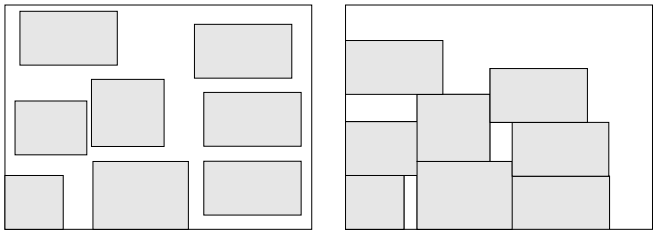
Relaxed placement methods have also been applied to three-dimensional problems involving irregular items. The method from [83] was generalized to three dimensions by Imamichi and Hiroshi [82], where objects are represented by a collection of spheres. Ikonen et al. [81] represents items as triangle meshes, and use a genetic algorithm to control the search. Overlap is evaluated by checking bounding boxes for intersection and subsequently the triangles of the items. Cagan et al., Yin and Cagan, Yin and Cagan [25, 158, 159] use simulated annealing and also handle various additional optimization objectives such as routing lengths. Intersection checks are done using *octree decompositions* of shapes.

An interested problem was investigated by Eisenbrand et al. [49] where the maximum number of uniform boxes that can be placed in the trunk of a car must be determined. For any placement of boxes they define a potential function that describes the total overlap and penetration depth between boxes and trunk sides and of pairs of boxes.

Relaxed placement methods work well for decision problems where one must find a non-overlapping placement within a specific container or number of containers. It is not clear if they can be used for knapsack packing problems where a specific subset of items must be selected. The method by Eisenbrand et al. [49] removes and inserts new items into the placement during the solution process, but the problem involves only one type of item.

The author of this thesis attempted to solve two-dimensional knapsack packing problems using the relaxed overlap method described in [A] and the following procedure: First, the canonical one-dimensional relaxation of the two-dimensional knapsack problem is solved. This relaxation is the one-dimensional knapsack problem which is created by considering the same set of items and item profit values, but setting the size of each item in the one-dimensional problem to its area from the two-dimensional problem. Once solved, a two-dimensional decision problem involving the given set of items is sought for using the method by [A]. If no solution is found within a given time-limit, a constraint is added to the one-dimensional relaxation that ensures that the same set of items cannot be selected, and the one-dimensional problem is solved again to reveal a new set of items. The procedure iterates until a solution can be found.

Since two-dimensional solutions to rectangular problems are often found to be within 2% of the optimal solution of the canonical one-dimensional relaxation, as described in Egeblad and Pisinger [C], one would expect that few constraints were required. Unfortunately, it proved impossible even to find solutions to relevant rectangular problems using this method, which is likely caused by the fact that the difference between the one-dimensional solution and the two-dimensional solution is too great and too many constraints are actually needed.



(a) (b)

Figure 23: *Normalized placement. (a) A placement of a set of rectangles. (b) Normalized placement, where non of the rectangles can be moved left or down without causing overlap.*

3.3.7 Bottom-Left Strategies and Envelopes

A placement of rectangles is *normalized* if it is impossible to translate any rectangle in the placement to the left, downwards, or – in three-dimensional packing – backwards, without causing overlap. It was proven by Herz [78] that an optimal placement which is normalized exists for the packing problems described in Section 2.1. The intuition for this proof is that the rectangles in an optimal solution may be translated to the left and downwards until no further translation is possible without introducing overlap (see Figure 23). The consequence of this fact is that the search for an optimal solution can be limited to placements which are normalized.

A commonly studied paradigm for two-dimensional rectangular packing is the so-called bottom-left principle. The bottom-left principle takes advantage of the property of normalized placements and is as follows: We are given an ordering of the items I as a list L. The rectangles are positioned in the order of L. The leftmost lowest (*bottom-left*) possible position is chosen for each rectangle. Chazelle [28] presents a "bottom-left" algorithm with $O(n^2)$ running time for n rectangles. Jakobs [86] and later Liu and Teng [98] considered heuristics for the rectangular strip-packing problem based on genetic algorithms in which the sequence (L) is the genotype, i.e. placements are represented by sequences and each individual represents its own sequence of items.

The method by Jakobs [86] was also extended to consider polygons and is one among several methods for polygons that use the bottom-left or a similar principle to determine the position of the next item in a sequential placement. Many of these methods rely on envelopes.

The notion of *envelopes* or *profiles* for packing problems are particular useful in methods that constructs a placement one item at a time. The purpose of the envelope is to reduce the set of feasible positions for each item, by "cutting" off part of the placement area (see figure 24). As that set is reduced, finding a suitable position for each item may be done more efficiently. In general, methods that use envelopes for rectangular placement rely on the fact that the set of normalized placements includes an optimal placement, since each rectangle is placed such that it abuts with the envelope.

For two-dimensional rectangular packing problems a variant of the envelope structure was presented by Scheithauer [134] and later for two- and three-dimensional problems by Martello et al. [108]. The placement is constructed starting from the lower-left corner of the placement area. The concept is illustrated on Figure 25. Whenever a new rectangle is placed, it may not be placed such that its lower-left corner falls under *and* to the left of any of the previously placed rectangles' upper-right corner. The boundary of the feasible positions is a stair-case pattern, and new rectangles may only be placed such that their lower-left corner abuts with the inner corner of steps of the stair-case (the circles on the figure). Once a rectangle is placed, the staircase is expanded, to contain the new rectangle.

The rectangular envelope may be represented by using the rectangles which have already been placed and can be updated in amortized constant time, each time a rectangle is placed. This renders

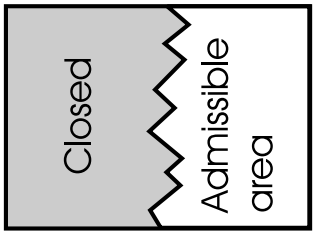


Figure 24: *The notion of an envelope. As the placement is constructed a part of the placement area is inaccessible (closed) and the admissible area for a shape is the remaining area of the placement.*

the envelope structure extremely efficient, and it has been used in a number of methods; Martello et al. [108] used it in conjunction with a branch-and-bound algorithm for the two- and three-dimensional bin-packing problem to determine if a selection of items were feasible. Given the set of items, Martello et al. [108] construct a placement recursively, by branching on each remaining rectangle and each position in the envelope. The same principle was reused in Martello et al. [109] for an exact method for the two-dimensional strip-packing problem, and the method was revisited by Caprara and Monaci [27] for the two-dimensional knapsack packing problem. The same envelope was also used by Pisinger [127] for a heuristic for the area minimization problem, and by Egeblad and Pisinger [C] for a heuristic for the two-dimensional knapsack packing problem.

While a three-dimensional variant of the envelope structure was presented by [108], it was later discovered by den Boef et al. [40], that this structure can represent only a subset of three-dimensional placements which is referred to as robot-packable that are also considered in Egeblad and Pisinger [C]. Although, an optimal solution may not be robot-packable, this subset still represents a comprehensive set of placements.

Envelopes have also been used extensively for polygon items and were introduced for a heuristic for the two-dimensional strip-packing problem with polygons by Art, Jr. [3]. While the set of feasible locations for each new rectangle is reduced to a discrete set for the rectangular envelope, the set of feasible locations remains infinite for irregular shapes.

Several other heuristics for the strip-packing problem with polygons use some form of envelope principle along with a greedy strategy similar to the bottom-left principle. The heuristic by Oliveira et al. [123] places the polygons sequentially at the position in an "envelope" which is deemed most promising according to different measures. Gomes and Oliveira [72] later added a 2-Exchange neighborhood to the heuristic, which exchanges the position of items in the sequence. A tabu-search based heuristic in which the sequence is modified was also presented by Burke et al. [23].

The "jostling" heuristic by Dowsland et al. [46] places the polygons sequentially, repeatedly from left to right and right to left. In each iteration the sequence is changed to reflect the last placement.

While normalized optimal solutions exists for rectangular packing problems, this is not the case for problems involving polygons, and one cannot expect to find the optimal placement for problems with polygons using bottom-left principles.

3.3.8 Abstract Representations

A direct representation of placements is a list of the individual coordinates of each item. This representation has the main draw-back that infeasible placements which contain overlap can be represented, and transitions from one placement without overlap to another placement without overlap are not simple to achieve as discussed in Section 3.3.6. This is illustrated on Figure 26 (a) and (b) where the

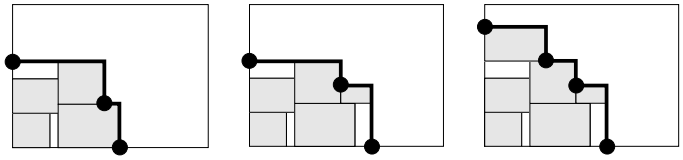


Figure 25: *Rectangular envelope. The shape of an envelope as a placement is constructed by sadding one rectangle at a time. Rectangles are placed in normalized fashion and may not be placed lower or left of the envelope which is indicated by thick lines. Circles indicate feasible positions of the lower-left corner of the next rectangle to be added.*

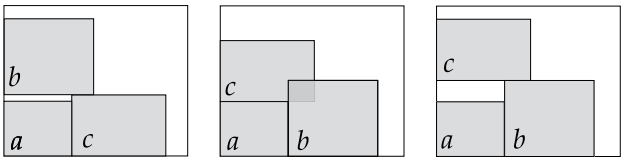


Figure 26: *(a) A placement of rectangles a, b, c. (b) Overlap will appear if b and c are exchanged directly. (c) No overlap will appear if the rectangles are exchanged in the sequence pair representation and then positioned using a decoding algorithm (see text).*

rectangles *b* and *c* exchanges position.

To tackle this problem, many heuristics rely on some form of *abstract representation* of the position of items, which does not deal directly with coordinates. Instead the placement is represented as either sequences or graphs which can be used to assign coordinates to each item using some form of *decoding* algorithm. The decoding algorithm generally ensures that the resulting placement is feasible.

The heuristic by Jakobs [86], which was touched upon in Section 3.3.7, actually uses an abstract representation of placements. Here a placement of items is represented implicitly by a sequence which can be decoded to a feasible placement using a bottom-left algorithm that places items one-by-one in the order or the sequence. The advantage of this representation is that any modification of the sequence will still lead to a feasible placement.

On the other hand, small changes in the sequence may lead to completely different placements, which makes them difficult to work with during the intensification stage of local search heuristics, where the main focus is to reach a local minimum.

Abstract representations work well with local search heuristics since they can easily perform a small alteration of the current abstract representation, and evaluate the outcome. Alterations can be as simple as exchanging the position of two items in the ordered list mentioned above, which is difficult without causing overlap using a direct representation.

During the last half of the 1990's several more advanced representations for two-dimensional rectangular packing problems were proposed. The intention of these representations is to maintain the overall relative positions of rectangles, when small changes are made. The representation that ignited this research was the *Sequence Pair* which was introduced as part of a heuristic for the minimal area rectangle packing problem by Murata et al. [117],

A sequence pair consists of two sequences of the rectangles. If the rectangles are numbered $1,\ldots,n$, the sequence pair consists of two permutations of the numbers (two sequences) $< \sigma_+(1),\ldots,\sigma_+(n) >$ and $< \sigma_-(1),\ldots,\sigma_-(n) >$, where σ_+ and σ_- are permutation functions. A sequence pair can be converted into a placement using two simple rules. For $i,j \in \{1,\ldots,n\}$:

- $\sigma_a^{-1}(i) < \sigma_a^{-1}(j)$ and $\sigma_b^{-1}(i) < \sigma_b^{-1}(j)$, j is placed to the right of i,
- $\sigma_a^{-1}(i) > \sigma_a^{-1}(j)$ and $\sigma_b^{-1}(i) < \sigma_b^{-1}(j)$, j is placed above i.

By symmetry all four possible relations between i and j can be deduced. Once the relations are established one can use them to construct constraint graphs as described in Section 3.2.4, that can be used to determine positions of i and j. A sequence pair $< c,b,a,d,e,g,f >, < a,e,b,c,d,f,g >$ represents the constraint graphs on Figure 14. It should be noted that all normalized placements can be represented by a sequence pair as proven by Murata et al. [117] and a method that can convert both a non-overlapping placement and a placement with overlap to a sequence pair was presented by Egeblad [48].

Since determining a placement of a set of rectangles based on constraint graphs requires $O(n^2)$, the transformation from a sequences pair to a placement can be done in $O(n^2)$ time by first constructing the constraint graph and then using it to determine a placement.

Figure 26 (a) and (c) illustrates an exchange of two rectangles in both sequences of the sequence pair. The placement in Figure 26 (a) is represented by the sequence pair $< b,a,c >, < a,c,b >$. Figure 26 (c) shows the placement of the sequence pair $< c,a,b >, < a,b,c >$ after the rectangles b and c have exchanged position in the sequences. Unlike the direct exchange based on coordinates shown on Figure 26 (b), the exchange of positions in sequences does not lead to overlap.

A number of authors have suggested faster decoding methods. Tang et al. [147] introduces an algorithm which can convert a sequence pair to a placement in $O(n \log n)$ time using longest weighted common subsequence algorithms. They also describe a simple $O(n^2)$ time algorithm which circumvents constraint graphs completely and uses only the sequences to determine the position of each rectangle. This result was later improved to $O(n \log\log n)$ time by Tang and Wong [146] with advanced data structures.

An $O(n \log\log n)$ decoding algorithm was also introduced by Pisinger [127] who used an envelope as described in Section 3.3.7. Items are placed one by one using the envelope, and the position in the envelope to be used for each rectangle is based on the sequence pair. The envelope structure is used in such a way that only relations between items from the envelope and the item to be placed are considered, therefore, this algorithm does not completely place rectangles according to the relative positions induced by the sequences. However, this decoding will generally generate more compact (semi-normalized) placements, and it was proven that any normalized minimal area packing solution can be still be represented with a sequence pair and this decoding. The method by Pisinger [127] can also be simplified to a decoding algorithm with running time $O(n^2)$.

The decoding algorithms mentioned above were all used for heuristics for the minimal area rectangle packing problem, but the sequence pair representation was also used by Egeblad and Pisinger [C] in conjunction with a 2-exchange neighborhood and simulated annealing for solving the two-dimensional knapsack packing problem. A three-dimensional variant of the sequence pair, referred to as sequence triple, is also introduced to solve the three-dimensional knapsack packing problem. The sequence-pair was also used in conjunction with a branch-and-bound method for an exact algorithm for the two-dimensional rectangular strip-packing problem by Kenmochi et al. [91].

The list of other similar representations include O-trees by Pang et al. [126], B*-trees by Chung et al. [32] and Corner Block List by Hong et al. [80]. All of these representations were introduced for

the minimal area rectangle packing problem. They are commonly used together with a metaheuristic, such as simulated annealing that controls local search based alterations of the representation.

Abstract representations for polygon problems have not been investigated to the same extent. A major issue is that while a normalized optimal feasible placement always exist for rectangular packing problems, a similar property is unlikely to exist for more general packing problems, especially if items can fit within holes of other items. Therefore it may be impossible to represent all relevant placements by something as simplistic as a pair of sequences. The current methods for irregular packing rely on a sequence and on the bottom-left principle or something similar, and decoding the sequence into a placement is a computational complex process. This is in contrast to the sequence pair representation for which an efficient implementation can decode *hundreds of thousands* of sequences containing 20-40 rectangles per second on modern commodity hardware.

3.3.9 Packing Classes

An interesting abstract representation for rectangular problems where multiple placements are represented by a single data-structure is the *packing class* which was introduced by Fekete and Schepers [55]. Fundamentally, a packing class consists of a set of undirected graphs $G_i = (V_i, E_i)$ for $i = 1, \ldots, d$ – one for each dimension of the problem. Each graph G_i contains a set of nodes which corresponds to the rectangles of the problem similar to the constraint graphs of Section 3.2.4. Additionally, each graph G_i is an *interval graph* which means that it represents the intersection of intervals on the real line. To create a graph from a placement, one connects two nodes in graph G_i with an edge if and only if the two corresponding rectangles overlap when considering their extents in the ith dimension; i.e., an edge is added between nodes of rectangles a and b in G_1 if and only if $[x_a, x_a + w_a] \cap [x_b, x_b + w_b] \neq \emptyset$.

Fekete and Schepers [55] consider construction of such graphs and denote a set of edges $E_1, \ldots, E_d$ for the d graphs as a *packing class* if it satisfies the following properties:

- P1: The graphs $G_i = (V, E_i)$ for $i = 1, \ldots, d$ are interval graphs.
- P2: Each stable set of S of G_i is x_i-feasible for $i = 1, \ldots, d$. A stable set in this context is a set of unconnected vertices and the requirement means that the sum of the width of a set of non-overlapping rectangles in one dimension cannot exceed the placement area width.
- P3: $\cap_{i=1}^{d} E_i = \emptyset$ for $i = 1, \ldots, d$. This means that two rectangles cannot overlap in all dimensions.

A packing class defines a whole set of placements. To convert a packing class into a placement, one must consider the complement of each graph $G_i^C = (V, E_i^C)$ and assign an orientation to each of the edges E_i^C. Let the set F_i be the assigned orientation of E_i^C then F_i must be a *transitive orientation*, i.e., the directed graph $G_i^F = (V, F_i)$ must be transitive. Once the transitive orientation is known the graphs G_i^F can be converted to a placement by setting the coordinates of rectangle a using $x_i(a) = \max\{x_i(b) + w_i(b) \mid (a,b) \in F_i\}$, which is the same principle as was used to convert constraint graphs into placements (see Section 3.2.4). Figure 27 illustrates the concept. The 36 placements which belong to same packing class but corresponds to different orientations are illustrated on Figure 28.

Fekete and Schepers [55, 57] and Fekete et al. [58] also show how to construct the sets $E_1, \ldots, E_d$ such that they constitute a packing class. They use a form of tree-search which adds an edge between two rectangles in one of the graphs in each node of the tree. To limit the size of the tree they rely on a set of mathematical theorems which can identify if the properties P1, P2, and P3 are all satisfied. It should be noted that the actual positions of the rectangles, are generally not required to solve a problem and therefore a transitive orientation is not required.

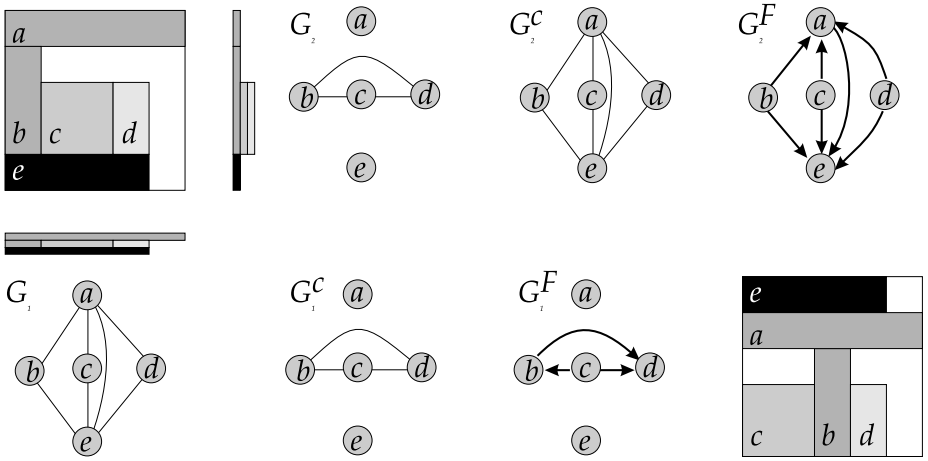


Figure 27: *Packing classes. A placement of rectangles in the upper-right corner is used to generate the interval graphs G_1 and G_2. The edges of G_1 and G_2 constitute the packing class that the placement is part of. The edges of the complementary graphs G_1^C and G_2^C are given an orientation to reveal graphs G_1^F and G_2^F which can be used to generate the placement of the lower-right corner.*

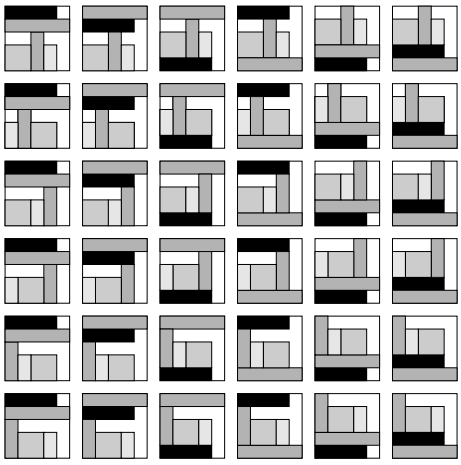


Figure 28: *The 36 placements which arise from the packing class represented by the edges of G_1 and G_2 from figure 27.*

Their method is used to solve the multidimensional orthogonal knapsack packing problem to optimality by Fekete et al. [58], but strategies for both rectangular strip-packing and bin-packing are also discussed by Fekete and Schepers [57]. It should be noted that the authors also take advantage of bounds which we will return to in Section 3.3.11.

The appeal of packing classes is that they can reduce the solution space significantly, not only by completely disregarding the coordinates of the individual rectangles, but also by representing many symmetric placements with one single packing class.

On the other hand, the drawback of packing classes is that they are relatively difficult to construct. Further, since they do not consider actual coordinates, they seem unsuitable for layout problems where an objective such as balance or interconnectivity (as in VLSI-layout optimization) must be considered. It is also not entirely obvious how to handle problems where single items may be rotated.

3.3.10 Constraint Programming

Constraint programming techniques for determining if a feasible packing can be found were used by den Boef et al. [40] for an an exact method for the three-dimensional rectangular knapsack packing problem.

For each pair of items one of six relations – since it is a three-dimensional problem – can be selected similar to the IP-formulation by Onodera et al. [124], where a binary variable is used to decide the relation between two boxes.

The algorithm uses tree-search to find a feasible assignment of the boxes; in each node of the tree a pair of boxes is considered, and the algorithm branches on each of the six possible relations between the rectangles. The algorithm back-tracks if this leads to a feasible assignment.

To determine if the assignment is feasible the algorithm checks if the chosen set of relations will cause rectangles to be positioned beyond the placement area. This can be done in $O(n^2)$ time since the set of relations induces a constraint graph as discussed in Section 3.2.4. A number of "look-ahead" techniques are used to determine if a branch will lead to an infeasible solution, which reduces the total number of branches required. The technique was later used by Pisinger and Sigurd [131] to solve the two-dimensional bin-packing problem to optimality with column generation.

From a certain point of view the IP technique by Onodera et al. [124], the packing class generation technique by Fekete and Schepers [55], and the constraint programming technique are all equivalent. A placement (or a class of placements) is constructed by branching on a number of relations. The IP model and the constraint programming technique both consider relations of the type "left-of" or "right-of". Packing-classes are oblivious to the exact relation since they consider undirected graphs and only designate if two items overlap in a dimension or not. This makes the feasibility checks required by Fekete and Schepers [55] harder than the techniques used for constraint programming, but each packing class cover several placements. Therefore it is not clear which of the methods can consider most placements within some specified amount of time.

3.3.11 Bounds

Both exact methods and heuristics often take advantage of *bounds* which predict the optimal value efficiently. Exact methods are commonly based on the branch-and-bound paradigm and use both upper and lower bounds to avoid unfruitful branches. Bounds have been utilized mainly for the bin-packing problem. Here upper bounds can be found with a heuristic, while lower bounds are mostly based on analysis of the total item area or volume. We consider only the latter type of bounds here and only for orthogonal rectangular problems where rotation of the items is not allowed.

The simplest way to determine if a set of items may be placed within a container, how many containers are required for the items, or how large a container will be required, is to consider the total volume of the items. If it exceeds the container space the items cannot be placed within the container. For the the bin-packing problem the continuous lower bond given by

$$L_0(I) = \left\lceil \frac{\sum_{i=1}^{n} w_i h_i}{WH} \right\rceil,$$

for an instance I can be used to determine the number of bins.

A more accurate bound, L_2, was presented by Martello and Vigo [107]. This bound belongs to a class of bounds which fit into a general scheme developed by Fekete and Schepers [54] which is based on the notions of *dual feasible functions* and *conservative scales*. A function $u : [0,1] \rightarrow [0,1]$ is dual feasible if, for any *finite* set of *non-negative* real numbers $S \subset \mathbb{R}$:

$$\sum_{x \in S} x \leq 1 \Rightarrow \sum_{x \in S} u(x) \leq 1.$$

Fekete and Schepers [54] show that a class of dual feasible functions is the basis for the bounds presented by Martello and Toth [106] and Martello and Vigo [107]. To evaluate these bounds item dimensions are changed; items with sides larger than $1 - \varepsilon$ are expanded to 1 and items with sides less than ε are discarded.

Fekete and Schepers [54] proceed to introduce conservative scales. Intuitively, a conservative scale alters the dimensions of the items, but in such a way, that if we cannot find a feasible placement of items with the modified dimensions, then we cannot find a feasible placement of items with the original dimensions.

If all items are scaled such that the container dimensions become $[0,1]^d$ for a d dimensional problem, then, according to the proof by Fekete and Schepers [54], any dual feasible function can be used to modify item dimensions. Fekete and Schepers [54] present three classes of dual feasible functions which may be used to alter item dimensions. Once item dimensions have been altered, one may use the volume criteria stated in the beginning of this section to determine if the items with modified dimensions are feasible to place within a container for the knapsack packing problem, the number of bins required for the bin-packing problem, or the required container length for the strip-packing problem. Since the item dimensions were changed by a dual feasible function, the volume based bounds for the problem instance with modified item dimensions holds for the original instance.

The bounds based on the dual feasible function presented by Fekete and Schepers [54] dominates the bounds of Martello and Toth [106] and Martello and Vigo [107]. Other bounds were presented by Boschetti and Mingozzi [19] and Clautiaux et al. [34] for the two-dimensional bin-packing problem without rotation, and by Boschetti and Mingozzi [20] for two-dimensional bin-packing with ninety degree rotation of items.

The bounds by Martello and Vigo [107] were used for a branch-and-bound algorithm for the two-dimensional bin-packing problem, and later for the three-dimensional variant by Martello et al. [108]. Martello et al. [109] use an extension of the bounds by Fekete and Schepers [54] and Martello and Toth [106] for an exact algorithm for the bin-packing problem. This method was also by used Caprara and Monaci [27] for the two-dimensional knapsack packing problem. The bounds by Fekete and Schepers [54] were used for an exact algorithm for the d-dimensional bin-packing and strip-packing problems by Fekete and Schepers [56], and for an exact algorithm for the the two- and higher-dimensional knapsack packing problem by Fekete et al. [58].

It should also be noted that the MIP formulations of Section 3.3.1 can be relaxed to linear programs which may be used for bounds. This observation was used in conjunction with column generation by Scheithauer [135] to find bounds for the container loading problem.

The problem with bounds based solely on volumes is that the dimensions of a particular set of rectangular items can render a feasible placement of the items impossible even though their combined volume is far less than that of the container. The scheme presented by Fekete and Schepers [54] remedies part of this problem, but based on the papers by Caprara and Monaci [27] and Fekete et al. [58] it seems that the current bounds are still not strong enough to enable branch-and-bound methods to reach optimal solutions for problems where more than 20 rectangular items can fit within the container at the same time.

3.3.12 Approximation Algorithms

In recent years the term approximation algorithm has become synonymous with a polynomial time algorithm with guaranteed performance bounds. An approximation algorithm A for a minimization problem has ratio bound ρ if for any instance I we have $\frac{A(I)}{\mathrm{OPT(I)}} \leq \rho$, where $\mathrm{OPT}(I)$ and $A(I)$ are the optimal and the value returned by A respectively, An asymptotic ratio bound describes the worst ratio bound as the size of the problem instances approaches ∞.

Approximation algorithms for multidimensional packing are scarce. The techniques in this category are commonly based on sequential placement according to either *first fit decreasing* (FFD) or *nearest fit decreasing* (NFD) algorithms which proceed as follows: First items are sorted according to decreasing height (FFDH and NFDH) or decreasing size (FFDS and NFDS). Then items are placed one-by-one in bins, shelves, or layers. To simplify the description we will call them bins in the following. Initially one bin is open. As an item is placed it is either positioned in the first of the open bins which has enough space to accommodate it (FFD) or only in the last currently opened bin (NFD). If none of the examined bins are large enough for the item, a new bin is opened and the item placed is in the new bin.

Bansal et al. [7] proved that no Asymptotic Polynomial Time Approximation Scheme (APTAS) exists for the two-dimensional rectangular bin-packing problem. They also present a polynomial time algorithm in n to find the optimal number of bins, as long as bin-sizes are increased by ε. Note that $\frac{1}{\varepsilon}$ appears in the exponent of n in the asymptotic running time of their algorithm. A similar result which was discovered independently was presented by Correa and Kenyon [37]. Bansal et al. [7] also present an APTAS for the problem of placing rectangles into a minimal enclosing rectangle. The algorithm is based on the *nearest fit decreasing height* (NDFH) principle and the running time of the algorithm is polynomial in n and $\frac{1}{\varepsilon}$.

Approximation algorithms for the two-dimensional bin-packing problem initially revolved around squares. An approximation algorithm for square packing, that is packing squares in a minimal number of squared bins, with an *absolute* worst case ratio of 2 was presented by van Stee [153] who also argued that this is the best possible provided $\mathcal{NP} \neq \mathcal{P}$. Ferreira et al. [59] presented an algorithm with asymptotic ratio bound of 1.988 for the same problem using an NFDS principle. The asymptotic ratio bound has later been improved by Seiden and van Stee [138] to $\frac{14}{9} + \varepsilon$ and also Kohayakawa et al. [94] who present an algorithm for the d-dimensional cube bin-packing problem with a general ratio bound of $2 - \frac{2}{3}^d$. Caprara [26] also presents an algorithm for this problem with a conjectured asymptotic ratio bound between 1.490 and 1.507, which is supported by experimental evidence.

Recently Bansal et al. [6] managed to move well beyond with an approximation algorithm for the general case where items are rectangles (not squares) which builds on the work by Caprara [26]. This algorithm has an asymptotic ratio bound of $\Pi_\infty + \varepsilon \approx 1.525... + \varepsilon$ and was generalized to higher dimensional problems, albeit with a higher ratio bound of $\ln(d + \varepsilon) + 1 + \varepsilon$ (for a d dimensional problem) which comes arbitrarily close to 2.0986 for $\varepsilon \to 0$.

Kenyon and Remila [92] presented an APTAS with a $(1+\varepsilon)$ performance guarantee for the *two-dimensional* strip-packing problem which is polynomial in n and $\frac{1}{\varepsilon}$ and is based on linear programming relaxation. The APTAS was later extended to handle the general case where items may be rotated by Jansen and van Stee [88]. For higher dimensions, Jansen and Solis-Oba [87] introduced an approximation algorithm for the *three-dimensional* strip-packing problem with asymptotic ratio bound $2+\varepsilon$. This improves on results by Miyazawa and Wakabayashi [111, 112, 113] although problems with square box-sides and ninety-degree rotation are also considered by Miyazawa and Wakabayashi [111, 113] as particular cases. The algorithm by Jansen and Solis-Oba [87] also generalizes to an algorithm with asymptotic ratio bound $4+\varepsilon$ for the three-dimensional bin-packing problem.

For the two-dimensional knapsack packing problem Caprara and Monaci [27] presents an approximation algorithm with an absolute ratio bound of $\frac{1}{3}-\varepsilon$.

At this point, approximation algorithms for packing problems are mostly of theoretical interest since either ratio bounds are too large or asymptotic running times too high. A notable exception is the tabu search heuristic for the two-dimensional bin-packing problem presented by Lodi et al. [101] which used an approximation algorithm with a performance ratio bound of 4 to generate initial solutions. The initial solutions generated by the approximation algorithm proved to act as good starting solutions despite the high ratio bound.

3.4 Speculations on The Future

Section 3.3 revealed many of the efficient and effective solution methods which exist for the majority of packing problems. As evident from the long list of methods we are still far from a complete unified solution approach which can handle any possible variant of packing problems. Authors still use individual strategies for individual problems and coming solution methods may still depend on the specific problem type. In this section we will attempt to shed some light on the current state of the different problem types and the future directions in each of them.

3.4.1 Rectangular Packing

The current exact methods for two-dimensional knapsack packing ([27, 58]) and three-dimensional bin-packing ([57, 108, 110, 131]) seem incapable of finding optimal solutions for problems where more than 20-30 items can be loaded at the same time inside the container within relevant computational time.

To increase the size of problems that can be considered, bounds must be stronger and the verification techniques, such as constraint programming and packing classes, must be extended to handle symmetries better to avoid considerations of equivalent placements or sub-placements. In general, exact methods are also still incapable of handling problems involving rotation of the items.

Heuristics for the two-dimensional knapsack ([1] and [C]) and bin-packing problems ([51, 114]) reveal promising results even when a large amount of rectangles can fit it the container at the same time. While heuristics exist for the three-dimensional bin-packing problem (see e.g. [51]), heuristics for three-dimensional knapsack packing problems, other than the container loading problem, are practically non-existing with the exception of the one presented in this thesis ([C]). This could be due to the fact that most practical problems are in the container loading domain where items are relatively small compared to the container and a large fraction of the items to choose from can fit within the container at the same time. Since most heuristics for container loading problems try to fill the container, rather than considering individual profit values of items, it would also seem relevant to consider methods for container loading problems where profit values are not proportional to item volumes.

3.4.2 Two-dimensional Irregular Packing

The strip-packing problem with rectangular items has received less attention in later years, and the field of strip-packing has been dominated mostly by methods for polygons. Recent heuristics reach utilization levels of between $85-95\%$ ([9, 23, 73, 83, 152]) including the one presented in this thesis ([A]).

The main missing puzzle in this area seems to be dealing with rotation efficiently. Methods for free rotation were presented by Liu and He [99] and Nielsen [120] but in both cases results for free rotation of items are not convincingly better than results when not allowing rotation or only allowing 90° or 180° rotation. This could either be because the rotation angle which is implicitly selected in the definition of the items of the instances is such that the non-rotational variant gives good results, or because the solution space when allowing rotations is much larger and therefore more difficult to search within.

The NFP is not suitable for rotational problems in its current form and this rules our generalizations of methods that utilize NFPs to rotational problems. However, it is not unlikely that a rotational variant of the NFP could somehow be generated. NFPs are related to robot motion planning and a few considerations for rotational planning are discussed in the book by de Berg et al. [39].

Another element missing for irregular packing is exact methods capable of handling more than 10 items such as the one mentioned in Section 3.3.1 ([38]). The problem here lies in finding proper branching rules and better bounds than the trivial area bound.

Surprisingly few methods for irregular packing deal with bin-packing, but many methods including the one of this thesis ([A]) are likely generalizable to bin-packing problems.

3.4.3 Irregular Three-dimensional Packing

Strip-packing of non-rectangular shapes in three dimensions have also been dealt with for both polyhedra by Stoyan et al. [145] and for spheres by Stoyan and et al. [140] Imamichi and Hiroshi [82] and in this thesis for polyhedra ([B]) and spheres/capsules ([F]). In this thesis we also present a heuristic for three-dimensional container loading or knapsack packing of furniture ([D]).

Methods for irregular shapes in three dimensions are still in their infancy and a utilization of more than 55–65 % seems out of reach with current methods as confirmed both by the papers on three-dimensional strip-packing ([145], [B], [F]) and the paper on container loading of furniture ([D]).

The low utilization for three-dimensional problems could be both due to the geometry of the items or simply because our methods are not powerful enough. From rectangular packing problems it is known that, while two-dimensional rectangular problems can be solved with utilization of $95-100\%$ (see [C]), the solutions for the three-dimensional variants, even with many small items in the container loading problem, rarely reach 90%. Better bounds for the three-dimensional problems involving irregular shapes could shed more light on the low utilization levels reached. However, the Kepler conjecture (see Section 2.1.5) which states that the maximal asymptotic utilization of homogeneous spherical packing is 74.048% indicates that utilization levels above 70% for irregular shapes in general, may be unlikely.

As for two-dimensional irregular packing, methods capable of solving problems involving three-dimensional irregular shapes with free rotation, may become more relevant in coming years. Especially, since it may be possible to reach higher levels of utilization if free rotation is allowed. A method that expands on the method of this thesis ([B]) to handle free rotational packing of three-dimensional polyhedra was presented by Nielsen [120].

3.4.4 New Constraints and Objectives

As methods for packing problems are becoming widely used in the industrial sector, more complicated objectives and constraints appear. We will discuss a few of them here.

For the strip-packing problem quality regions must be considered when cutting leather from hides. This is also discussed in this thesis ([A]).

For container loading one must often ensure that the load on an item is no larger than its strength. Items should also be positioned such that transportation is feasible and items will not drop and break. The problems are both described and considered in more detail in the paper on container loading of furniture in this thesis ([D]).

Another problem in container loading, is with respect to proximity. If a large consignment of items are to be delivered to the same location, but is for different customers, items for the same customer should also be close to each other to simplify the unloading process. A similar problem may occur when loading items; items may be selected from various locations within a large warehouse and the free space outside the container may only accommodate a limited number of items. To minimize the number of trips made in the warehouse one should try to place items which are close to each other in the warehouse close to each other in the container.

An aspect which has not been touched upon is the requirement that solutions can actually be physically packed. In many cases, human beings still handle the loading, but high utilization levels may be reached at the sacrifice of placements which are simple to achieve manually. This problem may have less significance as the loading process is increasingly managed by robots in the future.

Often containers should be loaded such that the consignment is balanced and the inertia moment is minimized. For airplanes this is important to minimize fuel. For trucks this is important to ensure that the axles of the trucks carry equal weight. Considerations and methods for this type of problem involving both rectangular and irregular shapes are presented in more detail in the paper which appears in this thesis ([E]).

This type of problems may be dealt with either by imposing new constraints, including them as a term in the objective function, or attempt to modify a good solution with respect to the "clean" packing problem in a posterior step. It is likely that methods which are capable or easily generalizable to handle the constraints mentioned above will receive more focus as the field matures in coming years.

3.4.5 Sensitivity Analysis

Many methods used by the industry may be used as parts of decision support systems where sensitivity and what-if scenarios must be analyzed. Here the problem may be to quickly answer the consequence of replacing a subset of the input items with a new set of items. Although re-optimizing the entire problem can answer such a query, the industry is interested in quick responses, and methods which can start from an existing solution may turn out to be beneficial.

Also methods which can perform their own set of analysis and suggestions – I.e.:"Replace input item set A with set B to achieve 2 % higher utilization" – could be of strong interest to the industry. The author of this thesis is unaware of any method that is capable of answering such queries or suggestions, but know from first hand experience that the industry desire this functionality.

A possible related topic is that the industry already use solution methods during the design phase of production for "simulation" purposes. Here the problem may be to select the set of item dimensions that return the best possible utilization given other constraints for instance with respect to required volume. To solve this problem with current methods one can re-optimize a problem with different

item dimensions and select the dimensions that return the highest utilization. New methods targeted for this problem may be able to get around re-optimization.

3.4.6 Integration with other problems

As techniques for solving packing problems are becoming better and the processing power increases, research may turn towards problems where packing appear in conjunction with other types of problems.

One of the well-known problems in operations research is the *vehicle routing problem* (VRP) (see e.g. Golden et al. [71]). In the recent years there has been an increasing interest in the integration of packing problems with vehicle routing problems. An exact algorithm for rectangular packing and VRP was introduced by Iori et al. [84]. Heuristics were introduced by Fuellerer et al. [61], Gendreau et al. [65] and Zachariadis et al. [162] for two-dimensional rectangular problems and a heuristic for the three-dimensional problem by Moura and Oliveira [116]. While this problem is difficult to handle since it involves two $\mathcal{NP}$-hard sub-problems, one may expect that solving the routing problem renders the associated packing problems easier since the items for one individual route could be insufficient to fill a complete container. In any case this topic seems open, especially for three-dimensional packing.

Another difficult problem appears in *production planning* and *supply-chain management* (see e.g. Pochet and Wolsey [132]). Here the set of items to be produced may depend on which items can be shipped in a container or a fleet of vehicles. Likewise, it may also depend on the set of raw-materials required for production which can be shipped in a single container. A model for such problems may involve both supply-chain optimization, VRP, and packing problems.

4 About the Papers

The thesis consists of six papers and this section contains a short presentation of each of the papers accompanied by a discussion. The papers [A], [B], [E] all use the same relaxed placement method and we will begin by discussing them in Section 4.1. In Section, 4.2 we discuss the paper on rectangular knapsack packing problems [C] and in Section 4.3 the paper on container loading of furniture [D]. Finally, in Section 4.4 we discuss the working paper on capsule packing for tertiary RNA structure prediction [F].

4.1 Relaxed Packing and Placement

All three of the papers [A], [B], and [E] take advantage of the same principle. The method is based on iterative overlap minimization and originates from the heuristic by Faroe et al. [53] for rectangular two- and three-dimensional bin-packing which uses the metaheuristic Guided Local Search by Voudouris and Tsang [155, 156]. To a large degree the framework presented by Faroe et al. [53] has remained unchanged in the papers considered in this thesis. The main difference between the work by Faroe et al. [53] and the work presented in this thesis is that we consider irregular items.

The papers all present methods that solve decision problems, i.e. determine if a feasible placement of items within given container dimensions exists. The procedure to solve the decision problem closely mimics that by Faroe et al. [53] and is as follows: The methods starts from a placement with overlap and repeatedly translates a single item either horizontally or vertically to a position with less overlap. A zero-overlap placement corresponds to a solution for the decision problem. Whenever a placement cannot be improved by a single translation, two items which overlap a lot are "penalized", i.e., placements where these two particular items overlap will receive a high objective value. This is the GLS element of the heuristic. The effect of this is, that the items are pushed away from each other in the following steps of the solution process since the heuristic will avoid placements where they overlap.

The heuristic which is used for the papers [A] and [B] starts with decision problems for large container lengths and then decreases the container length every time a solution to a decision problem has been found. The heuristic was also used for research outside this thesis. Nielsen [120] considered different measures of overlap instead of the area, free rotation of shapes, and arbitrary direction translation (non only horizontal or vertical). Nielsen [119] also considered *repeated pattern nesting*, i.e. achieving high utilization where the strip is infinite and the pattern generated is repeated an infinite number of times.

4.1.1 Two-dimensional Nesting

The first paper in this thesis ([A]) also represents the chronological first work and describes a heuristic for the two-dimensional strip-packing problem of polygon shapes using the principle described above. The main novelty of the paper is the minimal overlap translation algorithm that finds the horizontal or vertical translation of a single polygon which minimizes its overlap with the other polygons.

A proof of the correctness of this algorithm was given in the earlier work by Nielsen and Odgaard [121] (based on a note by the author of this thesis). However, this proof, which considered a more versatile set of translations, was deemed too complicated for a single paper and instead a simpler proof was presented in [A].

Recent updated experiments are presented in [A.1] and show that the current implementation still produce some of the best results of the literature.

An element which we never fully investigated was two-dimensional translation of polygons where the minimal overlap position can be found for the entire placement area instead of in just one direction. An algorithm to solve this problem was presented by Mount et al. [115] with a running time of $O((mn)^2)$ where n is the number of edges from the polygon to be translated and m the number of edges from all other polygons. The approach is based on an *arrangement* of line segments. Our initial investigations with implementations of this algorithm showed that there were many problems with degeneracies where lines and points in the arrangements would coincide and calculation with rational numbers was required to ensure stability. The running time of the first prototype implementation was far too high for our local search based heuristic since each translation would take seconds to calculate, and this local search neighborhood was dropped completely. However, it is possible that a similar neighborhood, which would just consider two-dimensional translations in a small area around the polygon to translate, is computational feasible.

The importance of the Guided Local Search (GLS) metaheuristic was not made clear in the original paper. The Guided Local Search heuristic seems particularly strong for this problem, because it works well in conjunction with the piecewise continuous objective function (overlap) and the minimal overlap translation algorithm. An interesting future direction would be to replace GLS with other metaheuristics. It is not clear how this could be done. For instance, a tabu-search heuristic (see e.g. [70]) with a tabu-list of placements would probably be difficult to combine with the minimal overlap translation algorithm. Simulated annealing (see e.g. Kirkpatrick et al. [93]) would require the acceptance of some form of randomly generated solution and it is not clear how such a solution would be generated to take advantage of the minimal overlap translation algorithm. There are similar concerns with other metaheuristics such as genetic algorithms.

A meta-heuristic which would be interesting to investigate as a replacement of GLS is adaptive large neighborhood search (ALNS) (see e.g. [129]). ALNS's ability to handle several different neighborhoods could make it interesting for this problem, since it would make it possible to introduce new neighborhoods such as exchange of the position of two items or the two-dimensional translation neighborhood mentioned above.

4.1.2 Three-dimensional Nesting

Although the first paper did sketch a three-dimensional translation algorithm, several details were missing, and only a prototype implementation for the decision variant of the three-dimensional problem was made. Results presented by Stoyan et al. [145] motivated a further investigation of the three-dimensional strip-packing problem with polyhedra and a generalization of the procedure to higher dimensions at the same time. The details of a heuristic for the three-dimensional strip-packing problem and a generalization to higher dimensions were reported in the second paper [B].

While the two-dimensional and three-dimensional procedures are the same and even share implementation, there are several aspects of the proof behind the correctness of the minimal overlap translation algorithm that were changed. Most important was the introduction of the notion of *balanced assignment*.

In the paper, the interior of the polytopes is defined as the set of points where a ray shot parallel to the x-axis intersects the boundary an odd number of times. Therefore, a ray from a point of the intersection of two polytopes should intersect the boundary of both the two polytopes an odd number of times. However, since the boundary is broken into many different pieces (faces and facets), the pieces cannot overlap, since that would cause problems in the even/odd counting principle used throughout the proofs. The notion of balanced assignment ensures that the pieces do not overlap.

Another difference between the two papers is that sides of the three-dimensional polyhedra (and

polytopes in general) are not limited to triangles but can take any convex form. Additionally, a greedy method for the three-dimensional strip-packing problem had to be introduced.

4.1.3 Optimization of the Center of Gravity

The paper [E] represents a minor addition to the family of papers on relaxed placement methods for two- and three-dimensional problems involving polygons and polyhedra. In this paper a heuristic for the problem where a given set of items, each with a weight, must be placed within a given container, such that overall balance and inertia moment are optimized. The paper was motivated by recent methods for this problem (see [E] for more details).

Although it is not a packing problem in the conventional sense, i.e., as described in Section 2.1, since the set of items to be placed and the container size are given, the purpose of the heuristic is to use it as a post-processing step of another packing algorithm which determines a feasible subset of items or container dimension.

The method used in the paper minimizes an objective function consisting of weighted linear combination of balance, inertia moment and overlap, using the same translational moves as in the papers [A] and [B]. As the procedure progresses the significance of balance and inertia moment is decreased, so that the overlap will have a higher impact on the solution process. This continues until a feasible solution is found at which point it is increased again and the heuristic allows overlapping solutions again.

A similar procedure was used by Faroe et al. [52] for the VLSI layout problem where the total wire-length of interconnected rectangles must be minimized, and the main contribution of the present paper is a demonstration that the same principle can be used to handle the relatively simple objective function involving balance and inertia.

It would also be interesting to investigate if the same principle can be used for three-dimensional layout problems with wire-connections which were considered by Yin et al. [160].

4.1.4 Relation to FFT Algorithms

An important aspect of the minimal translation algorithm is that it can also be seen as a maximal overlap translation algorithm. In this context it seems related to well-known convolution based methods used for e.g. protein docking problems using raster models introduced by Katchalski-Katzir et al. [89]. These methods compare two raster models (three-dimensional grids) with a Fast Fourier Transform (FFT) algorithm to determine where structural elements fit the best, i.e., which relative three-dimensional translation maximizes overlap of the surface. In other words, the objective is to find the $(x,y,z) \in \mathbb{Z}^3$ which solves:

$$\max_{x,y,z} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} f(i,k,j) \cdot g(i-x,k-y,j-z),$$

where $f : \mathbb{Z}^3 \to \{0,1\}$ and $g : \mathbb{Z}^3 \to \{0,1\}$ are "raster functions" which indicate whether a grid-cell in the three-dimensional $n \times n \times n$-grid representations is occupied by the surface or not (1 means occupied). The FFT makes this possible since the entire three-dimensional convolution h of f and g:

$$h(x,y,z) = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} f(i,k,j) \cdot g(i-x,k-y,j-z),$$

can be determined in $O(n^3 \log n)$ time using the FFT (see e.g. [36]). If a grid-cell is set to 1 whenever it is occupied by a structure the same procedure can be used to find the translation with maximal

overlap of two structures. In this case, $h(x,y,z)$ is a discrete version of the volume of overlap of the structures represented by f and g. This procedure somehow relates to our minimal overlap translation algorithm.

4.2 Rectangular Knapsack Packing

In the paper [C] a heuristic for the rectangular knapsack packing problem in two- and three-dimensions is presented. The main strategy in the paper is to use the sequence pair representation to represent placements. In addition to introducing a new heuristic for the two-dimensional knapsack packing problem, the paper also demonstrates how versatile the sequence pair representation is, and that it can be used for other problems than the minimal area packing problem, for which it had previously been applied to.

Another contribution in the paper is the introduction of the the sequence triple for three-dimensional representation of placements. To the author's knowledge this the only truly abstract representation of placement of boxes other than the graph- and naive sequence-based representations, i.e., constraint graphs, packing classes, and sequential placement. However, there are two draw-backs of the representation. Firstly, it is only capable of representing robot-packable placements. This set excludes mainly interlocking placements, which, fortunately, may have little relevance in practical applications. The second drawback is that the asymptotic running time of the decoding algorithm is $O(n^2)$ for n boxes. It would therefore be interesting to examine faster placement strategies or alternative representation for three-dimensional box placement.

The two-dimensional representation and heuristic are powerful enough to return results which are on a par with the current methods of the literature. The three-dimensional variants cannot be compared with other methods from the literature, but returns results which are close to the upper-bounds.

Initial experiments revealed that the three-dimensional representation and heuristic are not capable of handling container loading problem which contain far more items, but it is possible that the representation could be used if the small items were combined to larger building blocks, or a different heuristic principle was used.

4.3 Knapsack Packing of furniture

The paper [D] is the most interesting of the papers from a practical point of view and, in it, we present a heuristic for knapsack packing of pieces of furniture within a container. The procedure consists of a number of different steps including: A tree-search method for finding a overall good solution for large items, a local search heuristic for refining the solution, a local search heuristic for ensuring overall stability of the large items, a greedy heuristic for placement of medium sized items, and a wall-building heuristic for placing small items.

The pieces of furniture are represented by triangle mesh structures and the main strategy of the paper is to determine a large set of possible combinations of furniture to use for the heuristics. The heuristic now has to select both good combinations as well position each combination within the container. When placed, each of the selected combinations of pieces of furniture is aligned with one of the four corners of the width-height plane in the container.

The main contributions of the paper are the combination strategy, the four corner representation which forms the basis of both the tree-search and local search heuristics, and finally the method that ensures that each item is placed in a stable fashion.

The four-corner principle can be viewed as a special type of abstract representation, and the local search method used bears some resemblance to the method from the paper on two- and three-

dimensional knapsack packing ([C]); in both cases the heuristic attempts to exchange items in a sequence and allows placement of items outside the container, and only items inside the container are included when calculating the objective value. The methods of the two papers were developed in parallel and interestingly enough their similarity did not occur to us until late in the development process.

It would be interesting to investigate if elements of the paper can be generalized. It is possible that the combination strategy can be used for rectangular container loading. Here items would have to be combined in larger building blocks that can be managed by the tree-search and local search algorithms. The main problem, however, concerns generation of suitable building blocks. A possibility is to use some form of three-dimensional knapsack packing or minimal volume packing on smaller subsets.

It is also possible that the combination strategy can be used for other types of shapes. In the paper, the geometric analysis which forms the basis of combinations is based on the fact that both items must rest on the floor, and the analysis is made in two dimensions. A general three-dimensional analysis, e.g. based on minkowski-sums or relaxed placement of few items, could form the basis of a generalization to more arbitrary shapes.

The combination strategy could likely be used to improve the performance of the relaxed placement methods by assessing good combinations of items in a preprocessing step and translating combinations instead of individual items. An additional local search neighborhood could then change the combinations used as part of the solution process.

4.4 Cylinder Packing and Placement

The final paper [F] represents unfinished work and concerns a heuristic for placement of capsules which may function as a tool for RNA tertiary structure prediction. Prediction of RNA tertiary structure is related to prediction of protein tertiary structure and concerns prediction of the three-dimensional positions of the atoms of the molecule based on known primary and secondary structures. RNA molecules consist of many helical regions connected by the backbone of the molecule and RNA molecules differ from proteins in that secondary structure prediction can be used to accurately determine helical regions which appear in the tertiary structure.

The paper describes a method where the helical regions of RNA molecules are represented as capsules and geometric considerations are used to predict the tertiary structure. This is a so-called coarse grained method. Since atoms cannot overlap and the helical regions therefore do not overlap, the problem is to generate a non-overlapping placement of capsules. Helices are also connected by the backbone and this property is modeled with proximity constraints that ensures that connected helices are positioned close to each other.

RNA structures are generally compact due to the same hydrophobic forces which appear in proteins. Therefore, it is conjectured that the capsules should be placed somehow compactly. Three different compaction strategies are introduced in the paper and are similar to that of the other papers on relaxed placement ([A] and [B]). The three strategies attempts to minimizes either a box or sphere container which can contain the capsules. The molecular surfaces of the RNA molecules studied in conjunction with the paper are not spherical, and therefore it is unlikely that these compaction strategies are useful for tertiary structure prediction of RNA. However, it is possible that a different compaction strategy used in conjunction with relaxed placement can return useful results. Nevertheless results for a heuristic for the problem of compacting interconnected capsules is presented to demonstrate the potential of the relaxed placement method, and its ability to find placements of capsules with limited freedom (small container dimensions and proximity constraints).

Another problem considered in the paper is the placement problem where the capsules must be

placed within a given molecular surface such that the proximity requirements are met. Here one is given auxiliary information that describes a boundary of the molecule and the objective is to accurately guess the placement of atoms within the molecule. This is modeled as a decision problem where a feasible placement of the capsules within an irregular container must be found. A number of random feasible placements were generated and surprisingly, one of the placements is not far from the known real structure.

The paper represents *work in progress* and a number of aspects are missing from it. Firstly, different compaction strategies need to be investigated to determine if the problem can be considered as a compaction problem. Secondly, more experiments with placements where the molecular surface is given are required. Thirdly, it must be determined if the coarse-grained capsule placement can be successfully used as a starting point for accurate prediction methods.

From a packing problem point of view the main novelty in this paper concerns the overlap translation method. While the overall principle of the papers Egeblad et al. [A, B] has been reused, the translational algorithm used in [F] differs substantially. Instead of volume of overlap, which is hard to determine for capsules, the algorithm deals with directional intersection depth in this paper. This, and the ability to handle both proximity constraints and an irregular container demonstrates how universal the minimal overlap principle is.

5 Conclusion

This thesis presents a number of novel methods for packing problems. Three different types of heuristics are covered for both two- and three-dimensional packing problems. Both the relaxed placement methods and the heuristic for container loading of furniture involves irregular shapes.

The results for the strip-packing problem with irregular shapes with the relaxed placement techniques ([A] and [B]) are among the best in the literature for two- and three-dimensional problems, and the core element of the polygon packing procedure, the minimal overlap translation algorithm, can be implemented in less than one thousand lines of code, which makes it an appealing alternative to NFP based methods.

The heuristics for rectangular knapsack packing ([C]) demonstrate great potential and the sequence triple is a novel abstract representation for three-dimensional placements. Results are on a par with existing methods and the sequence pair and sequence triple representations are simple to implement – Placement methods can be implemented in a few hundred lines of code. The biggest question regarding the three-dimensional heuristic is if it can be scaled to manage container loading problems consisting of many more items.

The techniques used for container loading of furniture ([D]) are specific for this problem. The overall heuristic consists of many relatively simple sub-parts, and an interesting future direction would be to apply some of the principles to other problems. The most impressive part of this work is that the time from start to finish, i.e., being presented with the problem, dealing with the theory, and producing a practical software application, was less than 18 months. At the time when we began this project, there was no obvious way from the literature to deal with irregular shapes to the extent required by our industrial partner. Today, the principles are being used hundreds of times each week within our software.

Stability and balance issues are considered both as part of the heuristic for container loading of furniture and as an individual problem ([E]). The latter is one of several examples of the versatility and potential of the relaxed placement method presented in [A] and [B].

The principles of the relaxed placement method have also been used for the RNA tertiary structure prediction problem ([F]) which occurs in bioinformatics. The problem considered cylinders with capped ends and proximity constraints and the promising results show how universal the relaxed placement methodology is. While the results are promising the draft included in this thesis is incomplete and more experiments are needed in order to understand the full potential of the method.

A common topic throughout the thesis is the relaxed placement method based on the minimal overlap translation. While its ability to tackle several problems has been considered in this thesis and the possibilities of the method seem almost endless, we have yet to successfully use the principle to solve knapsack packing problems. It would be interesting to investigate the possibilities in this domain further as part of future research.

It would also be interesting to investigate generalization of the principles which were used for furniture packing and for there-dimensional rectangular knapsack packing.

Many other future directions have been pointed out in this thesis, both with respect to problem types and improvements of the presented methods. Solution methods for packing problems are slowly maturing, but there are still many interesting possibilities to be explored and I hope that some of the many topics considered in this thesis can form the basis of fruitful future research.

References

[A] J. Egeblad, B. K. Nielsen, and A. Odgaard. Fast neighborhood search for two- and three-dimensional nesting problems. *European Journal of Operational Research*, 183(3):1249–1266, 2007.

[B] J. Egeblad, B. K. Nielsen, and M. Brazil. Translational packing of arbitrary polytopes. *CGTA. Computational Geometry: Theory and Applications*, 2008. accepted for publication.

[C] J. Egeblad and D. Pisinger. Heuristic approaches for the two- and three-dimensional knapsack packing problem. *Computers and Operations Research*, 2007. In press (available online).

[D] J. Egeblad, C. Garavelli, S. Lisi, and D. Pisinger. Heuristics for container loading of furniture. Submitted, 2007.

[E] J. Egeblad. Placement of two- and three-dimensional irregular shapes for inertia moment and balance. Submitted, 2008.

[F] J. Egeblad, L. Guibas, M. Jonikas, and A. Laederach. Three-dimensional constrained capsule placement for coarse grained tertiary rna structure prediction. Working Paper, 2008.

[1] R. Alvarez-Valdes, F. Parreno, and J.M. Tamarit. A tabu search algorithm for two-dimensional non-guillotine cutting problems. Technical Report TR07-2004, Universitat de Valencia, 2004.

[2] R. Alvarez-Valdes, F. Parreno, and J.M. Tamarit. A GRASP algorithm for constrained two-dimensional non-guillotine cutting problems. *Journal of Operational Research Society*, 56:414–425, 2005.

[3] R. C. Art, Jr. An approach to the two dimensional, irregular cutting stock problem. Technical Report 36.Y08, IBM Cambridge Scientific Center, September 1966.

[4] B. S. Baker and J. S. Schwarz. Shelf algorithms for two-dimensional packing problems. *SIAM Journal on Computing*, 12(3):508–525, 1983.

[5] R. Balasubramanian. The pallet loading problem: A survey. *International Journal of Production Economics*, 28(2):217–225, November 1992.

[6] N. Bansal, A. Caprara, and Sviridenko M. Improved approximation algorithms for multidimensional bin packing problems. In *Proceedings of the 47th on Foundations of Computer Science (FOCS'06)*, pages 697–708. IEEE Computer Society, 2006.

[7] N. Bansal, J. Correa, C. Kenyon, and M. Sviridenko. Bin packing in multiple dimensions: Inapproximability results and approximation schemes. *Mathematics of Operations Research*, 31(1):31–49, 2006.

[8] J. E. Beasley. Algorithms for two-dimensional unconstrained guillotine cutting. *Journal of the Operational Research Society*, 36:297–306, 1985.

[9] J. A. Bennell and K. A. Dowsland. Hybridising tabu search with optimisation techniques for irregular stock cutting. *Management Science*, 47(8):1160–1172, 2001.

[10] J. A. Bennell and K. A. Dowsland. A tabu thresholding implementation for the irregular stock cutting problem. *International Journal of Production Research*, 37:4259–4275, 1999.

[11] J. A Bennell and X Song. A comprehensive and robust procedure for obtaining the nofit polygon using minkowski sums. *Computers and Operations Research*, 35:267–281, 2008.

[12] J. O. Berkey and P. Y. Wang. Two-dimensional finite bin-packing algorithms. *The Journal of the Operational Research Society*, 38(5):423–429, 1987.

[13] A. Bettinelli and G. Ceselli, A. Righini. A branch-and-price algorithm for the two-dimensional level strip packing problem. *4OR: A Quarterly Journal of Operations Research*, 2007. Available online.

[14] S. Bhattacharya and R. Bhattacharya. An exact depth-first algorithm for the pallet loading problem. *European Journal of Operational Research*, 110(3):610–625, 1998.

[15] E. E. Bischoff. Three-dimensional packing of items with limited load bearing strength. *European Journal of Operational Research*, 168:952–966, 2006.

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

[17] E. E. Bischoff and M. S. W. Ratcliff. Loading multiple pallets. *Journal of the Operational Research Society*, 46:1322–1336, 1995.

[18] A. Bortfeldt and H. Gehring. Applying tabu search to container loading problems. In *Operations Research Proceedings 1997*, pages 533–538. Springer, Berlin, 1998.

[19] M.A. Boschetti and A. Mingozzi. The two-dimensional finite bin packing problem, Part I: New lower bounds for the oriented case. *4OR*, 1(1):27–42, 2003.

[20] M.A. Boschetti and A. Mingozzi. The two-dimensional finite bin packing problem. Part II: New lower and upper bounds. *4OR*, 1(2):135–147, 2003.

[21] M.A. Boschetti, E. Hadjiconstantinou, and A. Mingozzi. New upper bounds for the two-dimensional orthogonal cutting stock problem. *IMA Journal of Management Mathematics*, 13:95–119, 2002.

[22] V. Bukhvalova and K. Vyatkina. An optimal algorithm for partitioning a set of rectangles with right-angled cuts. In *SIAM Conference on Geometric Design and Computing*, pages 125–136, 2003.

[23] E. K. Burke, R. Hellier, G. Kendall, and G. Whitwell. A new bottom-left-fill heuristic algorithm for the two-dimensional irregular packing problem. *Operations Research*, 54(3):587–601, 2006.

[24] E. K. Burke, R. S. R. Hellier, G. Kendall, and G. Whitwell. Complete and robust no-fit polygon generation for the irregular stock cutting problem. *European Journal of Operational Research*, 179:27—49, 2007.

[25] J. Cagan, D. Degentesh, and S. Yin. A simulated annealing-based algorithm using hierarchical models for general three-dimensional component layout. *Computer Aided Design*, 30(10):781–790, 1998.

[26] A. Caprara. Packing 2-dimensional bins in harmony. In *Proceedings of the 43rd Symposium on Foundations of Computer Science (FOCS'02)*, pages 490–499. IEEE Computer Society, 2002.

[27] A. Caprara and M. Monaci. On the 2-dimensional knapsack problem. *Operations Research Letters*, 1 (32):5–14, 2004.

[28] B. Chazelle. The bottom-left bin-packing heuristic: An efficient implementation. *IEEE Transactions on Computers*, 32(8), 1983.

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

[30] C. H. Cheng, B. R. Feiring, and T. C. E. Cheng. The cutting stock problem – a survey. *International Journal of Production Economics*, 36(3):291–305, October 1994.

[31] F. R. K. Chung, M. R. Garey, and D. S Johnson. On packing two-dimensional bins. *SIAM Journal on Matrix Analysis and Applications*, 3(1):66–76, 1982.

[32] Y. Chung, Y. Chang, G. Wu, and S. Wu. B*-tree: A new representation for non-slicing floorplans. In *Proceedings of Design Automation Conference*, pages 458–463, 2000.

[33] V. Chvatal. *Linear Programming*. W. H. Freeman, 1983.

[34] F. Clautiaux, J. Carlier, and Moukrim A. A new exact method for the two-dimensional bin-packing problem with fixed orientation. *Operations Research Letters*, 35:357–364, 2007.

[35] E. G. Coffman, M. R. Garey, D. S. Johnson, and R. E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9(4):808–826, 1980.

[36] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[37] J. Correa and C. Kenyon. Approximation schemes for multidimensional packing. In *Proceedings of the 15th ACM/SIAM Symposium on Discrete Algorithms*, pages 179–188. ACM/SIAM, 2004.

[38] K. Daniels, Z. Li, and V. Milenkovic. Multiple containment methods. Technical Report TR-12-94, Harvard University, Cambridge, Massachusetts, 1994.

[39] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications (2nd edition)*. Springer, 2000.

[40] E. den Boef, J. Korst, S. Martello, D. Pisinger, and D. Vigo. Erratum to 'The Three-Dimensional Bin Packing Problem': Robot-packable and orthogonal variants of packing problems. *Operations Research*, 53:735–736, 2005.

[41] J. K. Dickinson and G. K. Knopf. A moment based metric for 2-D and 3-D packing. *European Journal of Operational Research*, 122(1):133–144, 2000.

[42] J. K. Dickinson and G. K. Knopf. Packing subsets of 3d parts for layered manufacturing. *International Journal of Smart Engineering System Design*, 4(3):147–161, 2002.

[43] K. A. Dowsland. An exact algorithm for the pallet loading problem. *European Journal of Operational Research*, 31(1):78–84, July 1987.

[44] K. A. Dowsland and W. B. Dowsland. Packing problems. *European Journal of Operational Research*, 56:2–14, 1992.

[45] K. A. Dowsland and W. B. Dowsland. Solution approaches to irregular nesting problems. *European Journal of Operational Research*, 84:506–521, 1995.

[46] K. A. Dowsland, W. B. Dowsland, and J. A. Bennell. Jostling for position: Local improvement for irregular cutting patterns. *Journal of the Operational Research Society*, 49:647–658, 1998.

[47] H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44:145–159, 1990.

[48] J. Egeblad. Placement techniques for VLSI layout using sequence-pair legalization. Master's thesis, DIKU, University of Copenhagen, Denmark, 2003.

[49] F. Eisenbrand, S. Funke, A. Karrenbauer, J. Reichel, and E. Schömer. Packing a trunk: now with a twist! In *SPM '05: Proceedings of the 2005 ACM symposium on Solid and physical modeling*, pages 197–206, New York, NY, USA, 2005. ACM Press.

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

[51] O. Faroe, D. Pisinger, and M. Zachariasen. Guided local search for the three-dimensional bin packing problem. *INFORMS Journal on Computing*, 1999.

[52] O. Faroe, D. Pisinger, and M. Zachariasen. Guided local search for final placement in vlsi design. *Journal of Heuristics*, 9(3):269–295, 2003. ISSN 1381-1231.

[53] 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.

[54] S. Fekete and J Schepers. A general framework for bounds for higher-dimensional orthogonal packing problems. *Mathematical Methods for Operations Research*, 60:311–329, 2004.

[55] S. P. Fekete and J. Schepers. On more-dimensional packing I: Modeling. *Submitted to Discrete Applied Mathematics*, 1997.

[56] S. P. Fekete and J. Schepers. On more-dimensional packing II: Bounds. *Submitted to Discrete Applied Mathematics*, 1997.

[57] S. P. Fekete and J. Schepers. On more-dimensional packing III: Exact algorithms. *Submitted to Discrete Applied Mathematics*, 1997.

[58] S. P. Fekete, J. Schepers, and J. C van der Veen. An exact algorithm for higher-dimensional orthogonal packing. *Operations Research*, 55(3), 2007.

[59] C. E. Ferreira, F. K. Miyazawa, and Y. Wakabayashi. Packing squares into squares. *Pesquisa Operacional*, 19(2):223–237, 1999.

[60] J. B. G. Frenk and G. Galambos. Hybrid next-fit algorithm for the two-dimensional rectangle bin-packing problem. *Computing*, 39(3):201–217, 1987.

[61] M. Fuellerer, K.F. Doerner, R. Hartl, and M. Iori. Ant colony optimization for the two-dimensional loading vehicle routing problem. *Computers and Operations Research*, 2007.

[62] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[63] H. Gehring and A. Bortfeld. A parallel genetic algorithm for solving the container loading problem. *International Transactions in Operational Research*, 9:497–511, 2002.

[64] H. Gehring and A. Bortfeldt. A genetic algorithm for solving the container loading problem. *International Transactions in Operational Research*, 4:401–418, 1997.

[65] M. Gendreau, M. Iori, G. Laporte, and S. Martello. A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. *Networks*, 51(1), 2008.

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

[67] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting stock problem – Part I. *Operations Research*, 9:849–859, 1961.

[68] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting stock problem – Part II. *Operations Research*, 11:863–888, 1963.

[69] P. C. Gilmore and R. E. Gomory. Multistage cutting stock problems of two and more dimensions. *Operations Research*, 13:94–120, 1965.

[70] F. Glover. Tabu search - part 1. *ORSA Journal on computing*, 1(3):190–206, 1989.

[71] B. Golden, S. Raghaven, and E. (editors) Wasil. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer, 2008.

[72] A. M. Gomes and J. F. Oliveira. A 2-exchange heuristic for nesting problems. *European Journal of Operational Research*, 141:359–370, 2002.

[73] A. M. Gomes and J. F. Oliveira. Solving irregular strip packing problems by hybridising simulated annealing and linear programming. *European Journal of Operational Research*, 171(3):811–829, 2006.

[74] E. Hadjiconstantinou and N. Christophides. An exact algorithm for general, orthogonal, two-dimensional knapsack problems. *European Journal of Operational Research*, 83:39–56, 1995.

[75] T. C. Hales. A proof of the kepler conjecture. *Annals of Mathematics*, 162:1065–1185, 2005.

[76] R. Heckmann and T. Lengauer. A simulated annealing approach to the nesting problem in the textile manufacturing industry. *Annals of Operations Research*, 57(1):103–133, 1995.

[77] E. Herbert and K. A. Dowsland. A family of genetic algorithms for the pallet loading problem. *Annals of Operations Research*, 63(3):415–436, 1996.

[78] J. C. Herz. A recursive computing procedure for two-dimensional stock cutting. *IBM Journal of Research and Development*, 16:462–469, 1972.

[79] M. Hifi. Dynamic programming and hill-climbing techniques for constrained two-dimensional cutting stock problems. *Journal of Combinatorial Optimization*, 8(1):65–84, 2004.

[80] X. Hong, G. Huang, Y. Cai, J. Gu, S. Dong, and C. Cheng. Corner block list: An effective and topological representation of non-slicing floorplan. In *Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*, pages 8 – 12, 2000.

[81] I. Ikonen, W. E. Biles, A. Kumar, J. C. Wissel, and R. K. Ragade. A genetic algorithm for packing three-dimensional non-convex objects having cavities and holes. In *Proceedings of the 7th International Conference on Genetic Algortithms*, pages 591–598, East Lansing, Michigan, 1997. Morgan Kaufmann Publishers.

[82] T. Imamichi and N. Hiroshi. *A Multi-sphere Scheme for 2D and 3D Packing Problems*, volume 4638/2007, pages 207–211. 2007.

[83] T. Imamichi, M. Yagiura, and H. Nagamochi. An iterated local search algorithm based on nonlinear programming for the irregular strip packing problem. In *Proceedings of the Third International Symposium on Scheduling, Tokyo Japan*, pages 132–137, 2006.

[84] M. Iori, J. J. Salazar-Gonzalez, and D. Vigo. An exact approach for the vehicle routing problem with two-dimensional loading constraints. *Transportation Science*, 40:342–350, 2006.

[85] S. Jain and H. C. Gea. Two-dimensional packing problems using genetic algorithms. *Engineering with Computers*, 14(3):206–213, 1998.

[86] S. Jakobs. On genetic algorithms for the packing of polygons. *European Journal of Operational Research*, 88:165–181, 1996.

[87] K. Jansen and R. Solis-Oba. An asymptotic approximation algorithm for 3d-strip packing. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm table of contents*, pages 143–152, 2006.

[88] K. Jansen and R. van Stee. On strip packing with rotations. In *Proceedings of the 37-th Annual ACM Symposium on the Theory of Computing (STOC 2005)*, pages 755–761. ACM, 2005.

[89] E. Katchalski-Katzir, I. Shariv, M. Eisenstein, A. A. Friesem, C. Aflalo, and I. A. Vakser. Molecular surface recognition: determination of geometric fit between proteins and their ligands by correlation techniques. In *Proceedings of National Academic Society of the United States of America*, volume 89(6), pages 2195–2199. National Academy of Sciences, 1992.

[90] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, Berlin, Germany, 2004.

[91] M. Kenmochi, T. Imamichi, K. Nnobe, M. Yagiura, and H. Nagamochi. Exact algorithms for the 2-dimensional strip packing problem with and without rotations. Technical Report 2007-005, Department of Applied Mathematics and Physics, Kyoto University, 2007.

[92] C. Kenyon and E. Remila. A near-optimal solution to a two-dimensional cutting stock problem. *Mathematics of Operations Research*, 25(4):645–656, 2000.

[93] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220 (4598):671–680, 1983.

[94] Y. Kohayakawa, F. Miyazawa, P. Raghavan, and Y. Wakabayashi. Multidimensional cube packing. *Electronic Notes of Discrete Mathematics*, 7, 2001.

[95] K. K. Lai and J. W. M. Chan. Developing a simulated annealing algorithm for the cutting stock problem. *Computers and Industrial Engineering*, 32:115–127, 1997.

[96] Z. Li and V. Milenkovic. Compaction and separation algorithms for non-convex polygons and their applications. *European Journal of Operational Research*, 84(3):539–561, 1995.

[97] L. Lins, S. Lins, and R. Morabito. An l-approach for packing (ℓ, w)-rectangles into rectangular and l-shaped pieces. *Journal of the Operational Research Society*, 54(7):777–789, 2003.

[98] D. Liu and T. Teng. An improved bl-algorithm for genetic algorithms of the orthogonal packing of rectangles. *European Journal of Operational Research*, 112:413–420, 1999.

[99] H. Liu and Y. He. Algorithm for 2D irregular-shaped nesting problem based on the nfp algorithm and lowest-gravity-center principle. *Journal of Zhejiang University - Science A*, 7(4):570–576, 2006.

[100] A. Lodi, S Martello, and D. Vigo. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, 11(4):345 – 357, 1999.

[101] A. Lodi, S. Martello, and D. Vigo. Approximation algorithms for the oriented two-dimensional bin packing problem. *European Journal of Operational Research*, 112:158–166, 1999.

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

[103] 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.

[104] 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.

[105] H. Lutfiyya, B. McMillin, P. Poshyanonda, and C. Dagli. Composite stock cutting through simulated annealing. *Journal of Mathematical and Computer Modelling*, 16(2):57–74, 1992.

[106] S. Martello and P. Toth. Lower bounds and reduction procedures for the bin-packing problem. *Discrete Applied Mathematics*, 26:59–70, 1990.

[107] S. Martello and D. Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44:388–399, 1998.

[108] S. Martello, D. Pisinger, and D. Vigo. The three-dimensional bin packing problem. *Operations Research*, 48(2):256–267, 2000. ISSN 0030364X.

[109] S. Martello, M. Monaci, and D. Vigo. An exact approach to the strip packing problem. *INFORMS Journal on Computing*, 3(15):310–319, 2003.

[110] S. Martello, D. Pisinger, D. Vigo, Edgar den Boef, and Jan Korst. Algorithms for general and robot-packable variants of the three-dimensional bin packing problem. *ACM Transactions on Mathematical Software*, 33, 2007.

[111] F. K. Miyazawa and Y Wakabayashi. Packing problems with orthogonal rotations. In *Proceedings of the 6th Latin American Symposium on Theoretical Informatics*, pages 359–368, 2004.

[112] F. K. Miyazawa and Y Wakabayashi. An algorithm for the three-dimensional packing problem with asymptotic performance analysis. *Algorithmica*, 18:122–144, 2000.

[113] F. K. Miyazawa and Y. Wakabayashi. Approximation algorithms for the orthogonal z-oriented 3-d packing problem. *SIAM Journal on Computing*, 29(3):1008 – 1029, 1999.

[114] M. Monaci and P. Toth. A set-covering-based heuristic approach for bin-packing problems. *Informs Journal Computing*, 18:71–85, 2006.

[115] David M. Mount, Ruth Silverman, and Angela Y. Wu. On the area of overlap of translated polygons. *Computer Vision and Image Understanding*, 64(1):53–61, 1996.

[116] A. Moura and J. F. Oliveira. An integrated approach to the vehicle routing and container loading problems. *OR Spectrum*, 2008.

[117] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. VLSI module packing based on rectangle-packing by the sequence pair. *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, 15:1518–1524, 1996.

[118] B. K. A. Ngoi, M. L. Tay, and E. S. Chua. Applying spatial representation techniques to the container packing problem. *International Journal of Production Research*, 32:111–123, 1994.

[119] B. K. Nielsen. An efficient solution method for relaxed variants of the nesting problem. In Joachim Gudmundsson and Barry Jay, editors, *Theory of Computing, Proceedings of the Thirteenth Computing: The Australasian Theory Symposium*, volume 65 of *CRPIT*, pages 123–130, Ballarat, Australia, 2007. ACS.

[120] B. K. Nielsen. *Nesting Problems and Steiner Tree Problems*. PhD thesis, DIKU, University of Copenhagen, Denmark, 2008.

[121] B. K. Nielsen and A. Odgaard. Fast neighborhood search for the nesting problem. Technical Report 03/03, DIKU, Department of Computer Science, University of Copenhagen, 2003.

[122] J. F. Oliveira and J. S. Ferreira. Algorithms for nesting problems. *Applied Simulated Annealing*, pages 255–273, 1993.

[123] J. F. Oliveira, A. M. Gomes, and J. S. Ferreira. TOPOS - a new constructive algorithm for nesting problems. *OR Spektrum*, 22:263–284, 2000.

[124] H. Onodera, Y. Taniguchi, and K Tamaru. Branch-and-bound placement for building block layout. In *Proceedings of the 28th ACM/IEEE Design Automation Conference*, pages 433–439. ACM, 1991.

[125] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1998. Hardback ISBN: 0521640105; Paperback: ISBN 0521649765.

[126] Y. Pang, C. Cheng, K. Lampaert, and W. Xie. Rectilinear block packing using o-tree representation. In *Proceedings of the 2001 international symposium on Physical design*, pages 156 – 161, 2001.

[127] D. Pisinger. Denser packings obtained in O(n log log n) time. *INFORMS Journal on Computing*, 19(3): 395–405, 2007.

[128] D. Pisinger. Heuristics for the container loading problem. *European Journal of Operations Research*, 3 (141):382–392, 2002.

[129] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers and Operations Research*, 34:2403–2435, 2007.

[130] D. Pisinger and M. Sigurd. The two-dimensional bin packing problem with variable bin sizes and costs. *Discrete Optimization*, 2(2):154–167, 2005.

[131] D. Pisinger and M. M. Sigurd. Using decomposition techniques and constraint programming for solving the two-dimensional bin packing problem. *INFORMS Journal on Computing*, 19(1):36–51, 2007.

[132] Y. Pochet and L. A. Wolsey. *Production Planning by Mixed Integer Programming*. Springer, 2006.

[133] G. Scheithauer. Algorithms for the container loading problem. *Operations Research Proceedings 1991*, pages 445–452, 1992.

[134] G. Scheithauer. Equivalence and dominance for problems of optimal packing of rectangles. *Ricerca Operativa*, 83, 1997.

[135] G. Scheithauer. Lp-based bounds for the container and multi-container loading problem. *International Transactions in Operational Research*, pages 199–213, 1999.

[136] G. Scheithauer and U. Sommerweiss. 4-block heuristic for the rectangle packing problem. *European Journal of Operational Research*, 108:509–526, 1998.

[137] G. Scheithauer and J. Terno. The g4-heuristic for the pallet loading problem. *Journal of Operational Research Society*, 47(4):511–522, 1996.

[138] S. S. Seiden and R. van Stee. New bounds for multidimensional packing. *Algorithmica*, 36:261–293, 2003.

[139] S. S. Skiena. Minkowski sum. In *The Algorithm Design Manual*, pages 395–396. Springer-Verlag, New York, 1997.

[140] Y. Stoyan and et al. Packing of various radii solid spheres into a parallelepiped, 2001.

[141] Y. Stoyan and L.D. Ponomarenko. Minkowski sum and hodograph of the dense placement vector function. Technical Report SER. A10, Reports of the SSR Academy of Science., 1977.

[142] Y. Stoyan, J. Terno, G. Scheithauer, N. Gil, and T. Romanova. Phi-functions for primary 2d-objects, 2001.

[143] Y. Stoyan, J. Terno, G. Scheithauer, N. Gil, and T. Romanova. Phi-functions for primary 2d-objects, 2001.

[144] Y. Stoyan, G. Scheithauer, N. Gil, and T. Romanova. Φ-functions for complex 2d-objects. *4OR: Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 2(1):69–84, 2004.

[145] Y. Stoyan, N. I. Gil, G. Scheithauer, A. Pankratov, and I. Magdalina. Packing of convex polytopes into a parallelepiped. *Optimization*, 54(2):215–235, 2005. doi: 10.1080/02331930500050681.

[146] X. Tang and D. F. Wong. FAST-SP: a fast algorithm for block packing based on sequence pair. In *Asia and South Pacific Design Automation Conference*, pages 521–526, 2001.

[147] X. Tang, R. Tian, and D. F. Wong. Fast evaluation of sequence pair in block placement by longest common subsequence computation. In *Proceedings of DATE 2000 (ACM), Paris, France*, pages 106–110, 2000.

[148] A. Tarnowski, J. Terno, and G. Scheithauer. A polynomial time algorithm for the guillotine pallet loading problem. *INFOR*, 32:275–287, 1994.

[149] J. Terno, G. Scheithauer, U. Sommerweiss, and J. Riehme. An efficient approach for the multi-pallet loading problem. *European Journal of Operations Research*, 2(132):371–381, 2000.

[150] V. E. Theodoracatos and J. L. Grimsley. The optimal packing of arbitrarily-shaped polygons using simulated annealing and polynomial-time cooling schedules. *Computer methods in applied mechanics and engineering*, 125:53–70, 1995.

[151] G. T. Toussaint. A simple linear algorithm for intersecting convex polygons. *The Visual Computer*, 1(2): 118–123, 1985.

[152] S. Umetani, T. Yagiura, S. Imahori, K. Nonobe, and T. Ibaraki. A guided local searc algorithm based on a fast neighborhood search for the irregular strip packing problem. In *Proceedings of the Third International Symposium on Scheduling, Tokyo Japan*, 2006.

[153] R. van Stee. An approximation algorithm for square packing. *Operations Research Letters*, 32(6): 535–539, 2004.

[154] G. Varadhan and D. Manocha. Accurate minkowski sum approximation of polyhedral models. *Graphical Models*, 68:343–355, 2006.

[155] C. Voudouris and E. Tsang. Guided local search. Technical Report CSM-147, Department of Computer Science, University of Essex, Colchester, C04 3SQ, UK, August 1995.

[156] C. Voudouris and E. Tsang. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113:469–499, 1999.

[157] G. Wäscher, H. Haussner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183:1109–1130, 2007.

[158] S. Yin and J. Cagan. An extended pattern search algorithm for three-dimensional component layout. *Journal of Mechanical Design*, 122(1):102–108, 2000.

[159] S. Yin and J. Cagan. Exploring the effectiveness of various patterns in an extended pattern search layout algorithm. *Journal of Mechanical Design*, 126(1):22–28, 2004.

[160] S. Yin, J. Cagan, and P. Hodges. Layout optimization of shapeable components with extended pattern search applied to transmission design. *Journal of Mechanical Design*, 126(1):188–191, 2004.

[161] G. Young-Gun and K. Maing-Kyu. A fast algorithm for two-dimensional pallet loading problems of large size. *European Journal of Operational Research*, 127(1):193–202, October 2001.

[162] E. E. Zachariadis, C. D. Tarantilis, and Kiranoudis C. T. A guided tabu search for the vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research*, 2007.

[163] S. Zhong and J. Ghosh. A unified framework for model-based clustering, 2002.

Published in the *European Journal of Operational Research*, 2007

Fast neighborhood search for two- and three-dimensional nesting problems

Jens Egeblad* Benny K. Nielsen* Allan Odgaard*

Abstract

In this paper we present a new heuristic solution method for two-dimensional nesting problems. It is based on a simple local search scheme in which the neighborhood is any horizontal or vertical translation of a given polygon from its current position. To escape local minima we apply the meta-heuristic method *Guided Local Search*.

The strength of our solution method comes from a new algorithm which is capable of searching the neighborhood in polynomial time. More precisely, given a single polygon with m edges and a set of polygons with n edges the algorithm can find a translation with minimum overlap in time $O(mn\log(mn))$. Solutions for standard test instances are generated by an implementation and a comparison is done with recent results from the literature. The solution method is very robust and most of the best solutions found are also the currently best results published.

Our approach to the problem is *very* flexible regarding problem variations and special constraints, and as an example we describe how it can handle materials with quality regions.

Finally, we generalize the algorithm for the fast neighborhood search and present a solution method for three-dimensional nesting problems.

Keywords: Cutting, packing, nesting, 3D nesting, guided local search

1 Introduction

Nesting is a term used for many related problems. The most common problem is strip-packing where a number of irregular shapes must be placed within a rectangular strip such that the strip-length is minimized and no shapes overlap. The clothing industry is a classical example of an application for this problem. Normally, pieces of clothes are cut from a roll of fabric. A high utilization is desirable and it requires that as little of the roll is used as possible. The width of the roll is fixed, hence the problem is to minimize the length of the fabric. Other nesting problem variations exist, but in the following the focus is on the strip-packing variant. Using the typology of Wäscher et al. [36] this is a two-dimensional irregular open dimension problem (ODP).

In the textile industry the shapes of the pieces of clothes are usually referred to as *markers* or *stencils*. In the following we will use the latter term except when the pieces need to be defined more precisely e.g. as polygons.

In order to state the problem formally we first define the associated decision problem:

Nesting Decision Problem *Given a set of stencils $\mathcal{S}$ and a piece of material, position the stencils $\mathcal{S}$ such that*

*Department of Computer Science, University of Copenhagen, DK-2100 Copenhagen Ø, Denmark. E-mail: {jegeblad, benny, duff}@diku.dk.

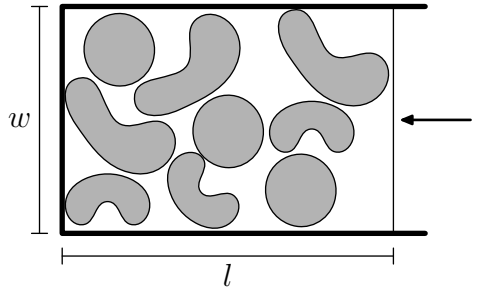


Figure 1: The Strip Nesting Problem. Place a number of stencils on a strip with width *w* such that no two stencils overlap and the length *l* of the strip is minimized.

- *No two stencils overlap.*
- *All stencils are contained within the boundaries of the material.*

The strip-packing variant can now be stated as:

Strip Nesting Problem. *Given a set of stencils $\mathcal{S}$ and a strip (the material) with width w find the minimal length l for which the Nesting Decision Problem can be solved (Figure 1).*

The Strip Nesting Problem is $\mathcal{NP}$-hard [e.g. 28].

In this paper we present a new solution method for the Strip Nesting Problem. After a short analysis of some of the existing approaches to the problem (Section 2) we present a short outline of the new solution method in Section 3. In short, the approach is a local search method (Section 4) using the meta-heuristic *Guided Local Search* (Section 5) to escape local minima. A very efficient search of the neighborhood in the local search is the subject of Section 6.

In the definitions above we have ignored the additional constraints which are often given for a nesting problem e.g. whether rotating and/or flipping the stencils is allowed. In Section 7 a discussion on how we handle such problem variations emphasizes the flexibility of our solution method.

Experiments show that our solution method is very efficient compared with other published methods. Results are presented in Section 8. Finally, in Section 9, it is shown that our solution method is quite easily generalized to three-dimensional nesting problems.

2 Existing Approaches to Nesting Problems

There exists numerous solution methods for nesting problems. A thorough survey by Dowsland and Dowsland [15] exists, but a more recent survey has also been done by Nielsen and Odgaard [28]. Meta-heuristics are one of the most popular tools for solving nesting problems. A detailed discussion of these can be found in the introductory sections of Bennell and Dowsland [6].

The following is a brief discussion of some of the most interesting approaches to nesting problems previously presented in the literature. The discussion is divided into three subsections concerning three different aspects of finding solutions for the problem: The basic solution method, the geometric approach and the use of a meta-heuristic to escape local minima.

2.1 Basic solution methods

Solution methods handling nesting problems generally belong to one of two groups. Those only considering legal placements in the solution process and those allowing overlap to occur during the solution process.

Legal placement methods

These methods never violate the overlap constraint. An immediate consequence is that placement of a stencil must always be done in an empty part of the material.

Most methods for strip packing follow the basic steps below.

1. Determine a sequence of stencils. This can be done randomly or by sorting the stencils according to some measure e.g. the area or the degree of convexity [30].

2. Place the stencils with some first/best fit algorithm. Typically a stencil is placed at the contour of the stencils already placed. Some algorithms also allow hole-filling i.e. placing a stencil in an empty area between already placed stencils [16, 17, 19].

3. Evaluate the length of the solution. Exit with this solution [3] or repeat at step 2 after changing the sequence of stencils [16, 19].

Unfortunately the second step is quite expensive and if repeated these algorithms can easily end up spending time on making almost identical placements.

Legal placement methods not doing a sequential placement do exist. These methods typically construct a legal initial solution and then introduce some set of moves (e.g. swapping two stencils) that can be controlled by a meta-heuristic to e.g. minimize the length of a strip [8, 9, 10, 11, 20].

Relaxed placement methods

The obvious alternative is to allow overlaps to occur as part of the solution process. The objective is then to minimize the amount of overlap. A legal placement has been found when the amount of overlap reaches 0. Numerous papers applying such a scheme exist with varying degrees of success [5, 6, 21, 24, 25, 27, 29, 33]. Especially noteworthy is the work of Heckmann and Lengauer [21].

In this context it is very easy to construct an initial placement. It can simply be a random placement of all of the stencils, although it might be better to start with a better placement.

Searching for a solution can be done by iteratively improving the placement i.e. decrease the total overlap and maybe also the strip-length. This is typically done by moving/rotating stencils.

2.2 Geometric approaches

The first problem encountered when handling nesting problems is how to represent the stencils. If the stencils are not already given as polygons then they can quite easily be approximated by polygons which is also what is done in most cases. A more crude approximation can be done using a *raster model* [12, 21, 27, 29]. This is a discrete model of the stencils created by introducing a grid of some size to represent the material i.e. each stencil covers some set of raster squares. The stencils can then be represented by matrices. An example of a simple polygon and its raster model equivalent is given in Figure 2. A low granularity of the raster model provides fast calculations at the expense of limited precision. Better precision requires higher granularity, but it will also result in slower calculations.

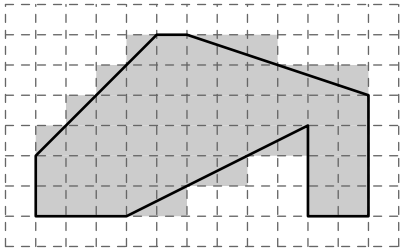


Figure 2: The raster model requires all stencils to be defined by a set of grid squares. The drawing above is an example of a polygon and its equivalent in a raster model.

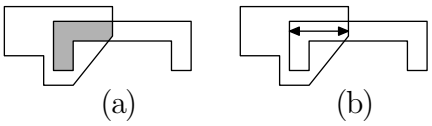


Figure 3: The degree of overlap can be measured in various ways. Here are two examples: (a) The precise area of the overlap. (b) The horizontal intersection depth.

Comparisons between the raster model and the polygonal model were done by Heckmann and Lengauer [21] and they concluded that the polygonal model was the better choice for their purposes.

Assuming polygons are preferred then we need some geometric tools to construct solutions without any overlaps. In the existing literature two basic tools have been the most popular.

Overlap calculations

The area of an overlap between two polygons (see Figure 3a) can be used to determine whether polygons overlap and how much they overlap. This can be an expensive calculation and thus quite a few alternatives have been suggested such as *intersection depth* [6, 14] (see Figure 3b) and the Φ-function [32] which can differentiate between three states of polygon interference: Intersection, disjunction and touching.

Solution methods using overlap calculations most often apply some kind of trial-and-error scheme i.e. they try to place or move a polygon to various positions to see if or how much it overlaps. This can then be used to improve some intermediate solution which might be allowed to contain overlap [5, 6, 7, 21].

No-Fit-Polygons (NFP)

Legal placement methods very often use the concept of the *No-Fit-Polygon* (NFP) [1, 2, 3, 16, 17, 19, 20, 30], although it can also be used in relaxed placement methods as done by Bennell and Dowsland [5].

The NFP is a polygon which describes the legal/illegal placements of one polygon in relation to another polygon, and it was introduced by Art, Jr. [3] (although named *envelope*).

Given two polygons P and Q the construction of the NFP of P in relation to Q can be found in the following way: Choose a reference point for P. Slide P around Q as closely as possible without intersecting. The trace of the reference point is the contour of the NFP. An example can be seen

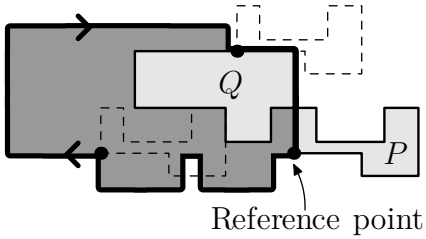


Figure 4: Example of the *No-Fit-Polygon* (thick border) of stencil P in relation to stencil Q. The reference point of P is not allowed inside the NFP if overlap is to be avoided.

in Figure 4. To determine whether P and Q intersect it is only necessary to determine whether the reference point of P is inside or outside their NFP. Placing polygons closely together can be done by placing the reference point of P at one of the edges of the NFP. If P and Q have s and t edges, respectively, then the number of edges in their NFP will be $O(s^2t^2)$ [4].

The NFP has one major weakness. It has to be constructed for all pairs of polygons. If the polygons are not allowed to be rotated it is feasible to do this in a preprocessing step in a reasonable time given that the number of differently shaped polygons is not too large.

2.3 Meta-heuristics

Both legal and relaxed placement methods can make use of meta-heuristics. The most popular one for nesting problems is *Simulated Annealing* (SA) [9, 20, 21, 27, 29, 33]. The most advanced use of it is by Heckmann and Lengauer [21] who implemented SA in 4 stages. The first stage is a rough placement, the second stage eliminates overlaps, the third stage is a fine placement with approximated stencils and the last stage is a fine placement with the original stencils.

Gomes and Oliveira [20] very successfully combine SA with the ideas for compaction and separation by Li and Milenkovic [26]. A very similar approach had previously been attempted by Bennell and Dowsland, Bennell and Dowsland [5, 6], but they combined it with a *Tabu Search* variant.

More exotic approaches are genetic, ant and evolutionary algorithms [10, 11, 25] — all with very limited success.

3 Solution Method Outline

In this section we will give a brief outline of our solution method. Our method is a relaxed placement method and it can handle irregular polygons with holes. A new geometric approach is utilized and the *Guided Local Search* meta-heuristic is used to escape local minima. This approach is inspired by a paper by Faroe et al. [18] which presented a similar approach for the two-dimensional Bin Packing Problem for rectangles.

The following describes the basic algorithm for the Strip Nesting Problem.

1. Finding an initial strip length
   An initial strip length is found by using some fast heuristic e.g. a bottom-left bounding box placement algorithm.

2. Reducing the strip length
   The strip length is reduced by some value. This value could e.g. be based on some percentage

of the current length. After reducing the strip length any polygons no longer contained within the strip are translated appropriately. This potentially causes overlap which is removed during the subsequent optimization.

3. Applying local search to reduce overlap
   The strip length is fixed now and the search for a solution without overlap can begin. The overlap is iteratively reduced by applying local search. More precisely, in each step of the local search a polygon is moved to decrease the total amount of overlap. The local search and its neighborhood are described in Section 4, and a very efficient search of the neighborhood is the focus of Section 6.

   If a placement without overlap is found for the current fixed strip length then we have found a solution, and step 2 can be repeated to find even better solutions. This might not happen though since the local search can be caught in local minima.

4. Escaping local minima
   To escape local minima we have applied the meta-heuristic Guided Local Search. In short, it alters the objective function used in step 3 and then repeats the local search. It will be described in more detail in Section 5.

4 Local Search

4.1 Placement

First we define a *placement* formally. Let $\mathcal{S} = \{s_1, \ldots, s_n\}$ be a set of polygons. A placement of $s \in \mathcal{S}$ can be described by the tuple $(s_x, s_y, s_\theta, s_f) \in \mathbb{R} \times \mathbb{R} \times [0, 2\pi) \times \{false, true\}$ where (s_x, s_y) is the position, s_θ is the rotation angle and s_f states whether s is flipped. Now the map $p : \mathcal{S} \to \mathbb{R} \times \mathbb{R} \times [0, 2\pi) \times \{false, true\}$ is a placement of the polygons $\mathcal{S}$.

4.2 Objective function

Given a set of polygons $\mathcal{S} = \{s_1, \ldots, s_n\}$ and a fixed-length strip with length l and width w, let $\mathcal{P}$ be the space of possible placements. We now wish to minimize the objective function,

$$g(p) = \sum_{i=1}^{n} \sum_{j=1}^{i-1} overlap_{ij}(p), \quad p \in \mathcal{P},$$

where $overlap_{ij}(p)$ is a measure of the overlap between polygons s_i and s_j. A placement p such that $g(p) = 0$ implies that p contains no overlap i.e. p solves the decision problem. We have chosen $overlap_{ij}(p)$ to be the area of intersection of polygons s_i and s_j with respect to the placement described by p.

4.3 Neighborhood

Given a placement p the local search may alter p to create a new placement p' by changing the placement of one polygon $s_i \in \mathcal{S}$. In each iteration the local search may apply one of the following four changes (depending on what is allowed for the given problem instance):

- **Horizontal translation.** Translate s_i horizontally within the strip.

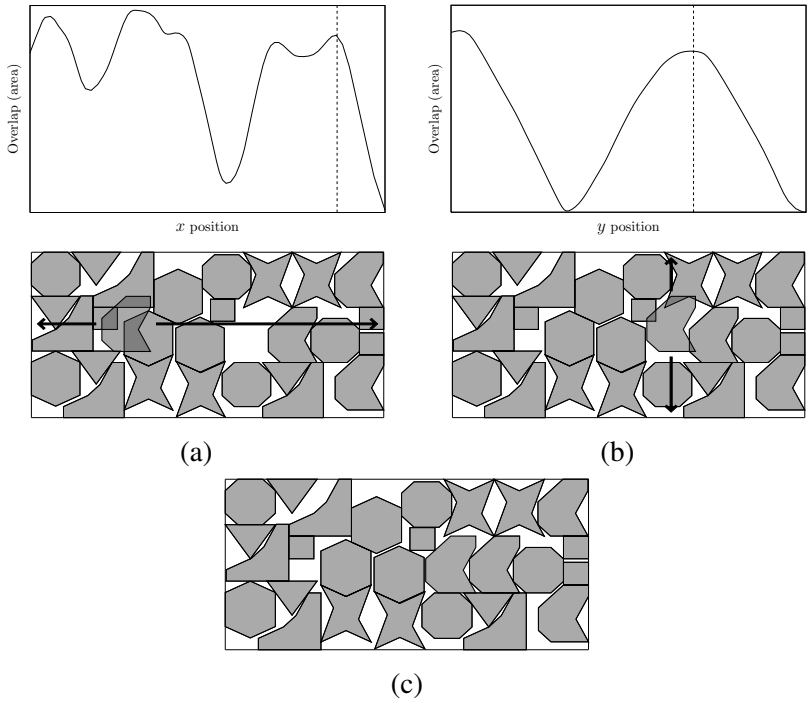


Figure 5: Example of a local search. (a) One polygon overlaps with several other polygons and is selected for optimization. In the top row we have drawn the amount of overlap as a function of the leftmost x-coordinate of the polygon. The positions beyond the dashed line are illegal since the polygon would lie partially beyond the right limit of the strip. Local search translates the polygon to a position with *least* overlap. (b) In the next iteration the local search may continue with vertical translation. The graph of overlap as a function of y-coordinate is shown and again the polygon is translated to the position with *least* overlap. (c) Local search has reached a legal solution.

- **Vertical translation.** Translate s_i vertically within the strip.
- **Rotation.** Select a new angle of rotation for s_i.
- **Flipping.** Choose a new flipping state for s_i.

The new position, angle or flipping state is chosen such that the overlap with all other polygons $\sum_{j \neq i} overlap_{ij}(p')$ is minimized. In other words p' is created from p by reducing the total overlap in a greedy fashion. An example of a local search is shown on Figure 5.

Let $N : \mathcal{P} \rightarrow 2^{\mathcal{P}}$ be the neighborhood function such that $N(p)$ is the set of all neighboring placements of p. We say that the placement p' is a *local minimum* if:

$$\forall p \in N(p') : g(p') \leq g(p),$$

i.e. there exists no neighboring solution with less overlap.

Now the local search proceeds by iteratively creating a new placement p' from the current placement p until p' is a local minimum.

5 Guided Local Search

To escape local minima encountered during local search we apply the meta-heuristic *Guided Local Search (GLS)*. GLS was introduced by Voudouris and Tsang [34] and has previously been successfully applied to e.g. the *Traveling Salesman Problem* [35] and two- and three-dimensional *Bin-Packing Problems* [18].

5.1 Features and penalties

Features are unwanted characteristics of a solution or in our case a placement. We let the features express pairwise overlap of polygons in the placement and define the indicator function:

$$I_{ij}(p) = \begin{cases} 0 & \text{if } overlap_{ij}(p) = 0 \\ 1 & \text{otherwise} \end{cases} \qquad i, j \in 1, \ldots, n, \;\; p \in \mathcal{P},$$

which determines whether polygon s_i and s_j overlap in the placement p.

The key element of GLS is the *penalties*. For each feature we define a penalty count ϕ_{ij} which is initially set to 0. We also define the *utility function*:

$$\mu_{ij}(p) = I_{ij}(p)\frac{overlap_{ij}(p)}{1+\phi_{ij}}.$$

Whenever local search reaches a local minimum p, the feature(s) with highest utility $\mu_{ij}(p)$ are "penalized" by increasing ϕ_{ij}.

5.2 Augmented objective function

The features and penalties are used in an *augmented objective function*,

$$h(p) = g(p) + \lambda \cdot \sum_{i=1}^{n} \sum_{j=1}^{i-1} \phi_{ij} I_{ij}(p),$$

where $\lambda \in]0, \infty[$ is a constant used to *fine-tune* the behavior of the meta-heuristic. Early experiments have shown that a good value for λ is around $1-4\%$ of the area of the largest polygon.

Instead of simply minimizing $g(p)$ we let the local search of Section 4 minimize $h(p)$. An outline of the meta-heuristic and the associated local search is described in Algorithm 1.

5.3 Improvements

The efficiency of GLS can be greatly improved by using *Fast Local Search* (FLS) [34]. FLS divides the local search neighborhood into sub-neighborhoods which are active or inactive depending on whether they should be considered during local search. In our context we let the moves of each polygon be a sub-neighborhood resulting in n sub-neighborhoods. Now it is the responsibility of the GLS algorithm to activate each sub-neighborhood and the responsibility of FLS to inactivate them.

For the nesting problem we have chosen to let GLS activate neighborhoods of polygons involved in penalty increments. When a polygon s is moved we activate all polygons overlapping with s before and after the move. FLS inactivates a neighborhood if it has been searched and no improvement has been found.

Algorithm 1: Guided Local Search for Nesting Decision Problem

```
Input: A set of polygons S;
Generate initial placement p;
foreach pair of polygons s_i, s_j ∈ S do
    Set φ_ij = 0;
while p contains overlap do
    // Local search:;
    while p is not local minimum do
        Select polygon s_i;
        Create p' from p using the best neighborhood
        move of s_i, i.e., such that h(p') is minimized;
        Set p = p'.;
    // Penalize:;
    foreach pair of polygons s_i, s_j ∈ S do
        Compute μ_ij(p);
    foreach pair of polygons s_i, s_j ∈ S such that μ_ij is maximal do
        Set φ_ij = φ_ij + 1;
return p
```

If GLS runs for a long time then the penalties will at some point have grown to a level where the augmented objective function no longer makes any sense in relation to the current placement. Therefore we also need to reset the penalties at some point e.g. after some maximum number of iterations which depends on the number of polygons.

6 Fast Neighborhood Search

To determine a translation of a single polygon which minimizes overlap we have developed a new polynomial-time algorithm. The algorithm itself is very simple and it is presented in Section 6.2, but the correctness of the algorithm is not trivial and a proof is required. The core of the proof is the Intersection Area Theorem which is the subject of the following section.

6.1 Intersection Area Theorem

In this section we will present a special way to determine the area of intersection of two polygons. Nielsen and Odgaard [28] have presented a more general version of the Intersection Area Theorem which dealt with rotation and arbitrary shapes. In this text however we have decided to limit the theory to polygons and horizontal translation since this is all we need for our algorithm to work. It will also make the proof shorter and easier to understand.

In order to state the proof we need to define precisely which polygons we are able to handle. First some definitions of edges and polygons.

Definition 3 (Edges)**.** *An edge e is defined by its end points $e_a, e_b \in \mathbb{R}^2$. Parametrically an edge is denoted $e(t) = e_a + t(e_b - e_a)$ where $t \in [0,1]$. For a point $p = (p_x, p_y) \in \mathbb{R}^2$ and an edge e we say $p \in e$ if and only if $p = e(t_0)$ for some $t_0 \in [0,1]$ and $p_y \neq \min(e_{a_y}, e_{b_y})$.*

The condition, $p_y \neq \min(e_{a_y}, e_{b_y})$, is needed to handle some special cases (see Lemma 1).

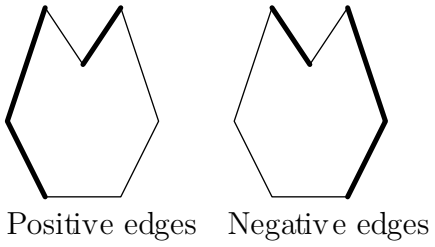


Figure 6: Positive and negative edges of a polygon according to Definition 6.

Definition 4 (Edge Count Functions)**.** *Given a set of edges E we define two edge count functions, $\overleftarrow{f_E}(p), \overrightarrow{f_E}(p) : \mathbb{R}^2 \to \mathbb{N}_0$,*

$$\begin{aligned} \overleftarrow{f_E}(p) &= |\{e \in E \mid \exists\, x' < p_x \,:\, (x', p_y) \in e\}|, \\ \overrightarrow{f_E}(p) &= |\{e \in E \mid \exists\, x' \geq p_x \,:\, (x', p_y) \in e\}|. \end{aligned}$$

Definition 5 (Polygon)**.** *A polygon P is defined by a set of edges E. The edges must form one or more cycles and no pair of edges from E are allowed to intersect. The* interior *of the polygon is defined by the set*

$$\tilde{P} = \{p \in \mathbb{R}^2 | \overleftarrow{f_E}(p) \equiv 1 \ (\mod 2)\}.$$

For a point $p \in \mathbb{R}^2$ we write $p \in P$ if and only if $p \in \tilde{P}$.

Note that this is an extremely general definition of polygons. The polygons are allowed to consist of several unconnected components and cycles can be contained within each other to produce holes in the polygon.

Now, we will also need to divide the edges of a polygon into three groups.

Definition 6 (Sign of Edge)**.** *Given a polygon P defined by an edge set E we say an edge $e \in E$ is* positive *if*

$$\forall t,\ 0 < t < 1 :\ \exists \varepsilon > 0 :\ \forall \delta,\ 0 < \delta < \varepsilon :\ e(t) + (\delta, 0) \in P. \tag{1}$$

Similarly we say e is negative *if Equation 1 is true with the points $e(t) - (\delta, 0)$. Finally we say e is* neutral *if e is neither positive nor negative.*

The sets of positive and negative edges from an edge set E are denoted E^+ and E^-, respectively.

Although we will not prove it here it is true that any non-horizontal edge is either positive or negative, and that any horizontal edge is neutral. Notice that the positive edges are the "left" edges and the negative edges are the "right" edges with respect to the interior of a polygon (see Figure 6).

The following lemma states some important properties of polygons and their positive/negative edges.

Lemma 1. *Given a vertical coordinate y and some interval I, we say that the horizontal line $l_y(t) = (t, y)$, $t \in I$,* crosses *an edge e if there exist t_0 such that $l_y(t_0) \in e$. Now assume that P is a polygon defined by an edge set E then all of the following holds.*

1. *If $I =]-\infty, \infty[$ and we traverse the line from $-\infty$ towards ∞ then the edges crossed alternate between being positive and negative.*

2. *If $I =]-\infty,\infty[$ then the line crosses an even number of edges.*

3. *Assume $p \notin P$ then the infinite half-line $l_{p_y}(t)$ for $I = [p_x,\infty[$ will cross an equal number of positive and negative edges. The same is true for $I =]\infty, p_x[$.*

4. *Assume $p \in P$. If $I = [p_x,\infty[$ and if the line crosses n positive edges then it will also cross precisely $n+1$ negative edges. Similarly, if $I =]-\infty, p_x[$ and if the line crosses n negative edges then it will also cross precisely $n+1$ positive edges.*

Proof. We only sketch the proof. First note that some special cases concerning horizontal edges and the points where edges meet are handled by the inequality in Definition 3.

The first statement easily follows from the definition of positive and negative edges since clearly any positive edge can only be followed by a negative edge and vice-versa when we traverse from left to right. The other statements follow from the first statement and the observation that the first edge must be positive and the last edge must be negative. □

The following definitions are unrelated to polygons. Their purpose is to introduce a precise definition of the area between two edges. Afterwards this will be used to calculate the area of intersection of two polygons based purely on pairs of edges.

Definition 7 (Containment Function)**.** *Given two edges e_1 and e_2 and a point $p \in \mathbb{R}^2$ define the* containment function

$$\mathbf{C}(e_1,e_2,p) = \begin{cases} 1 & \text{if } \exists x_1,x_2 \,:\, x_2 < p_x \leq x_1,\ (x_2,p_y) \in e_2, \text{and } (x_1,p_y) \in e_1 \\ 0 & \text{otherwise.} \end{cases}$$

Given two sets of edges, E and F, we generalize the containment function by summing over all pairs of edges,

$$\mathbf{C}(E,F,p) = \sum_{e_1 \in E} \sum_{e_2 \in F} \mathbf{C}(e_1,e_2,p).$$

Note that given two edges e_1 and e_2 and a point p then $\mathbf{C}(e_1,e_2,p) = 1 \Rightarrow \mathbf{C}(e_2,e_1,p) = 0$.

Definition 8 (Edge Region and Edge Region Area)**.** *Given two edges e_1 and e_2 we define the* edge region $R(e_1,e_2) = \{p \in \mathbb{R}^2 \mid \mathbf{C}(e_1,e_2,p) = 1\}$ *and the area of $R(e_1,e_2)$ as*

$$\mathbf{A}(e_1,e_2) = \int\int_{R(e_1,e_2)} 1 dA = \int\int_{p \in \mathbb{R}^2} \mathbf{C}(e_1,e_2,p) dA,$$

Given two sets of edges, E and F, we will again generalize by summing over all pairs of edges,

$$\mathbf{A}(E,F) = \sum_{e_1 \in E} \sum_{e_2 \in F} \mathbf{A}(e_1,e_2).$$

The edge region of two edges $R(e_1,e_2)$ is the set of points in the plane for which the containment function is 1. This is exactly the points which are both to the right of e_2 *and* to the left of e_1 (see Figure 7).

We will postpone evaluation of $\mathbf{A}(e_1,e_2)$ to Section 6.2 since we do not need it to prove the main theorem of this section. Instead we need to prove a theorem which we can use to break down the intersection of two polygons into regions.

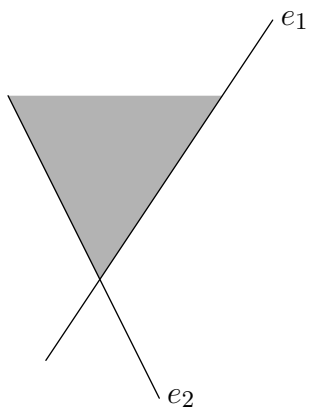


Figure 7: The edge region $R(e_1, e_2)$ of two edges e_1 and e_2 (see Definition 8).

Containment Theorem *Given polygons P and Q defined by edge sets E and F, respectively, then for any point $p \in \mathbb{R}^2$ the following holds:*

$$\begin{aligned} p \in P \cap Q &\Rightarrow w(p) = 1 \\ p \notin P \cap Q &\Rightarrow w(p) = 0, \end{aligned}$$

where

$$w(p) = \mathbf{C}(E^+, F^-, p) + \mathbf{C}(E^-, F^+, p) - \mathbf{C}(E^+, F^+, p) - \mathbf{C}(E^-, F^-, p) \tag{2}$$

Proof. First we note that from the definition of the containment function **C** it is immediately obvious that the only edges affecting $w(p)$ are the edges which intersect with the line $l_{p_y}(t), t \in]-\infty, \infty[$, and only the edges from E which are to the right of p and the edges from F which are to the left of p will contribute to $w(p)$.

Now let $m = \overrightarrow{f_{E^+}}(p)$ and $n = \overleftarrow{f_{F^-}}(p)$. By using Lemma 1 we can prove this theorem by counting.

First assume $p \in P \cap Q$ which implies $p \in P$ *and* $p \in Q$. From Lemma 1 we know that $\overrightarrow{f_{E^-}}(p) = m+1$ and $\overleftarrow{f_{F^+}}(p) = n+1$. Inserting this into Equation 2 reveals:

$$\begin{aligned} w(p) &= (n+1)(m+1) + nm - (n+1)m - n(m+1) \\ &= nm + n + m + 1 + nm - nm - m - nm - n \\ &= 1. \end{aligned}$$

Now for $n \notin P \cap Q$ there are three cases for which we get:

$$\begin{aligned} p \notin P \wedge p \notin Q: \quad & w(p) = nm + nm - nm - nm &&= 0, \\ p \in P \wedge p \notin Q: \quad & w(p) = n(m+1) - nm + nm - n(m+1) &&= 0, \\ p \notin P \wedge p \in Q: \quad & w(p) = (n+1)m - nm + (n+1)m - nm &&= 0. \end{aligned}$$

□

We are now ready to prove the main theorem of this section.

Intersection Area Theorem *Given polygons P and Q defined by edge sets E and F, respectively, then the area of their intersection (denoted* α*) is*

$$\alpha = \mathbf{A}(E^+, F^-) + \mathbf{A}(E^-, F^+) - \mathbf{A}(E^+, F^+) - \mathbf{A}(E^-, F^-). \tag{3}$$

Proof. From the Containment Theorem we know:

$$\alpha = \int\int_{p\in\mathbb{R}^2} w(p)dA.$$

Using Equation 2 we get:

$$\begin{aligned}\int\int_{p\in\mathbb{R}^2} w(p)dA &= \int\int_{p\in\mathbb{R}^2} \mathbf{C}(E^+, F^-, p)dA + \int\int_{p\in\mathbb{R}^2} \mathbf{C}(E^-, F^+, p)dA \\ &- \int\int_{p\in\mathbb{R}^2} \mathbf{C}(E^+, F^+, p)dA - \int\int_{p\in\mathbb{R}^2} \mathbf{C}(E^-, F^-, p)dA\end{aligned}$$

Let us only consider $\int\int_{p\in\mathbb{R}^2} \mathbf{C}(E^+, F^-, p)dA$ which can be rewritten:

$$\begin{aligned}\int\int_{p\in\mathbb{R}^2} \mathbf{C}(E^+, F^-, p)dA &= \int\int_{p\in\mathbb{R}^2} \sum_{e\in E^+}\sum_{f\in F^-} \mathbf{C}(e, f, p)dA \\ &= \sum_{e\in E^+}\sum_{f\in F^-} \int\int_{p\in\mathbb{R}^2} \mathbf{C}(e, f, p)dA \\ &= \sum_{e\in E^+}\sum_{f\in F^-} \mathbf{A}(e, f) \\ &= \mathbf{A}(E^+, F^-).\end{aligned}$$

The other integrals can clearly be rewritten as well and we achieve the required result. □

Note that this theorem implies a very simple algorithm to calculate the area of an intersection without explicitly calculating the intersection itself.

6.2 Translational overlap

The idea behind the fast neighborhood search algorithm is to express the overlap of one polygon P with all other polygons as a function of the horizontal position of P. The key element of this approach is to consider the value of $\mathbf{A}(e, f)$ for each edge-pair in the Intersection Area Theorem and to see how it changes when one of the edges is translated.

Calculating the area of edge regions

Fortunately it is very easy to calculate $\mathbf{A}(e, f)$ for two edges e and f. We only need to consider three different cases.

1. Edge e is completely to the left of edge f (Figure 8a).

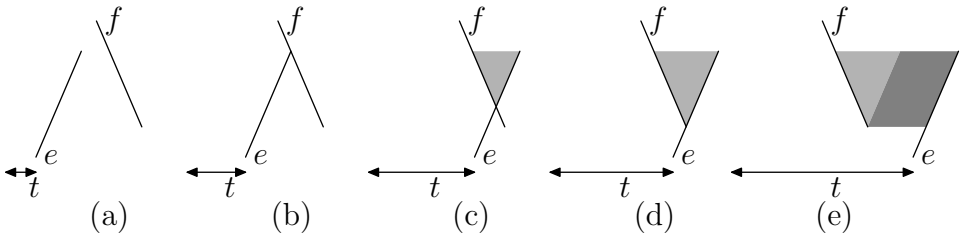


Figure 8: The edge region $R(e,f)$ of two edges as e is translated t units from left to right. (a) e is completely left of f. (b) e adjoins f. (c) e crosses f. (d) e and f are adjoined again. (e) e is to the right of f. Notice that $R(e,f)$ is either $\emptyset$, a triangle or a triangle combined with a parallelogram.

2. Edge e intersects edge f (Figure 8c).

3. Edge e is completely to the right of edge f (Figure 8e).

For the first case the region between the two edges is $\emptyset$. For the second case the region is a triangle and for the third case it is a union of a triangle and a parallelogram.

Now, let the edge e_t be the horizontal translation of e by t units and define the function $a(t) = \mathbf{A}(e_t,f)$. Assume e_t intersects f only when $t \in [t^{\triangle},t^{\square}]$ for appropriate $t^{\triangle}$ and $t^{\square}$ and let us take a closer look at how the area of the region behaves when translating e.

1) Clearly for $t < t^{\triangle}$ we have $a(t) = 0$. 2) It is also easy to see that for $t \in [t^{\triangle},t^{\square}]$ the intersection of the two edges occurs at some point which is linearly depending on t thus the height of the triangle is linearly depending on t. The same goes for the width of the triangle and thereby $a(t)$ must be a quadratic function for this interval. 3) Finally for $t > t^{\square}$, $a(t)$ is the area of the triangle at $t = t^{\square}$ which is $a(t^{\square})$ and the area of some parallelogram. Since the height of the parallelogram is constant and the width is $t - t^{\square}$, $a(t)$ for $t > t^{\square}$ is a linear function.

In other words, $a(t)$ is a piecewise quadratic function.

The next step is to extend the Intersection Area Theorem to edge sets E_t and F where E_t is every edge from E translated by t units, i.e we want to define the function $\alpha(t) = \mathbf{A}(E_t,F)$.

For each pair of edges $e_t \in E_t$ and $f \in F$ the interval of intersection is determined and the function $a_{e,f}(t) = \mathbf{A}(e_t,f)$ is formulated as previously described. All functions $a_{e,f}(t)$ are piecewise quadratic and have the form:

$$a_{e,f}(t) = \begin{cases} 0 & & & & \text{for} & t < t^{\triangle}_{e,f} \\ A^{\triangle}_{e,f}t^2 & + & B^{\triangle}_{e,f}t & + & C^{\triangle}_{e,f} & \text{for} \quad t \in [t^{\triangle}_{e,f}, t^{\square}_{e,f}] \\ & & B^{\square}_{e,f}t & + & C^{\square}_{e,f} & \text{for} \quad t > t^{\square}_{e,f} \end{cases} \tag{4}$$

We denote the constants $t^{\triangle}_{e,f}$ and $t^{\square}_{e,f}$ the *breakpoints* of the edge pair e and f, the values $A^{\triangle}_{e,f}$, $B^{\triangle}_{e,f}$, $C^{\triangle}_{e,f}$ the *triangle coefficients* of $a_{e,f}(t)$, and the values $B^{\square}_{e,f}$ and $C^{\square}_{e,f}$ the *parallelogram coefficients* of $a_{e,f}(t)$.

The total area of intersection between two polygons as a function of the translation of one of the polygons can now be expressed as in Equation 3:

$$\alpha(t) = \mathbf{A}(E^+_t,F^-) + \mathbf{A}(E^-_t,F^+) - \mathbf{A}(E^+_t,F^+) - \mathbf{A}(E^-_t,F^-). \tag{5}$$

The functions $a_{e,f}(t)$ are all piecewise quadratic functions, and thus any sum of these, specifically Equation 5, is also a piecewise quadratic function. In the next section we are going to utilize this result in our algorithm by iteratively constructing $\alpha(t)$ for increasing values of t.

Determining the minimum overlap translation

Given a polygon P defined by an edge set E and a set of polygons $\mathcal{S}$ ($P \notin \mathcal{S}$) defined by an edge set F the local search of Section 4 looks for a translation of P such that the total area of intersection with polygons from $\mathcal{S}$ is minimized. In this section we present an algorithm capable of determining such a translation.

The outline of the algorithm is as follows: For each pair of edges $(e, f) \in E \times F$ use the signs of the edges to evaluate whether $a_{e,f}(t)$ contributes positively or negatively to the sum of Equation 5. Then determine the breakpoints for e and f and compute the triangle and parallelogram coefficients of $a_{e,f}(t)$. Finally traverse the breakpoints of all edge pairs from left to right and at each breakpoint maintain the function

$$\alpha(t) = \tilde{A}t^2 + \tilde{B}t + \tilde{C},$$

where all of the coefficients are initially set to zero. Each breakpoint corresponds to a change for one of the functions $a_{e,f}(t)$. Either we enter the triangle phase at $t^{\triangle}_{e,f}$ or the parallelogram phase at $t^{\square}_{e,f}$ of $a_{e,f}(t)$. Upon entry of the triangle phase at $t^{\triangle}_{e,f}$ we add the triangle coefficients to $\alpha(t)$'s coefficients. Upon entry of the parallelogram phase at $t^{\square}_{e,f}$ we subtract the triangle coefficients and add the parallelogram coefficients to $\alpha(t)$'s coefficients.

To find the minimal value of $\alpha(t)$ we consider the value of $\alpha(t)$ within each interval between subsequent breakpoints. Since $\alpha(t)$ on such an interval is quadratic, determining the minimum of each interval is trivial using second order calculus. The overall minimum can easily be found by considering all interval-minima.

The algorithm is sketched in Algorithm 2. The running time of the algorithm is dominated by the sorting of the breakpoints since the remaining parts of the algorithm runs in time $O(|E| \cdot |F|)$. Thus the algorithm has a worst case running time of $O(|E| \cdot |F| \log(|E| \cdot |F|))$ which in practice can be reduced by only considering polygons from $\mathcal{S}$ which overlap horizontally with P.

Theoretically every pair of edges in $E \times F$ could give rise to a new edge in the intersection $P \cap Q$. Thus a lower bound for the running time of an algorithm which can compute such an intersection must be $\Omega(|E||F|)$. In other words, Algorithm 2 is only a logarithmic factor slower than the lower bound for determining the intersection for simply *one* position of P.

Algorithm 2: Determine Horizontal Translation with Minimal Overlap

```
Input: A set S of polygons and a polygon P ∉ S;
foreach edge e from polygons S \ {P} do
    foreach edge f from P do
        Create breakpoints for edge pair (e, f);
Let B = breakpoints sorted;
Define area-function α(t) = Ãt² + B̃t + C̃;
Set Ã = B̃ = C̃ = 0;
foreach breakpoint b ∈ B do
    Modify α(t) by changing Ã, B̃ and C̃;
    Look for minimum on the next interval of α(t);
return t with smallest α(t)
```

7 Problem Variations

The solution method presented in the previous sections can also be applied to a range of variations of nesting problems. Two of the most interesting are discussed in the following subsections. More details and other variations are described by Nielsen and Odgaard [28].

7.1 Rotation

We have efficiently solved the problem of finding an optimal translation of a polygon. A very similar problem is to find the optimal rotation of a polygon, i.e. how much is a polygon to be rotated to overlap the least with other polygons.

It has been shown by Nielsen and Odgaard [28] that a rotational variant of the Intersection Area Theorem is also possible. They also showed how to calculate the breakpoints needed for an iterative algorithm. It is an open question though whether an efficient iterative algorithm can be constructed.

Nevertheless the breakpoints can be used to limit the number of rotation angles needed to be examined to determine the existence of a rotation resulting in no overlap. This is still quite good since free rotation in existing solution methods is usually handled in a brute-force discrete manner i.e. by calculating overlap for a large set of rotation angles and then select a minimum.

7.2 Quality regions

In e.g. the leather industry the raw material can be divided into regions of quality [22]. Some polygons may be required to be of specific quality and should therefore be confined to these regions. This is easily dealt with by representing each region by a polygon and mark each region-polygon with a positive value describing its quality. Now if an element is required to be of a specific quality, region-polygons of poorer quality are included during overlap calculation with the element, thus disallowing placements with the element within a region with less-than-required quality. Note that the complexity of the translation algorithm is not affected by the number of quality levels.

8 Results

The solution method described in the previous sections has been implemented in C++, and we call the implementation 2DNEST. A good description of the data instances used can be found in Gomes and Oliveira [20]. These data instances are all available on the ESICUP homepage[1]. Some of their characteristica are included in Table 1. For some instances rotation is not allowed and for others 180° rotation or even 90° rotation is allowed. In 2DNEST this is handled by extending the neighborhood to include translations of rotated variants of the stencils. Note that the data instances Dighe1 and Dighe2 are jigsaw puzzles for which other solution methods would clearly be more efficient, but it is still interesting to see how well they are handled since we know the optimal solution.

Most of the data instances have frequently been used in the literature, but with regard to quality the best results are reported by Gomes and Oliveira [20]. They also report average results found when doing 20 runs for each instance. Gomes and Oliveira implemented two variations of their solution method (GLSHA and SAHA) and results for the latter can be found in Table 1 (2-4GHz Pentium 4). Average computation times are included in the table since they vary between instances. More precisely they vary between 22 seconds and 173 minutes. When considering all instances the average

[1] http://www-apdio-pt/esicup

computation time is more than 74 minutes. In total it would have taken more than 15 days to do all of the experiments on a single processor.

SAHA is an abbreviation of "simulated annealing hybrid algorithm". A greedy bottom-left placement heuristic is used to generate an initial solution, and afterwards simulated annealing is used to guide the search of a simple neighborhood (pairwise exchanges of stencils). Linear programming models are used for local optimizations including removing any overlap.

We have chosen to run 2DNEST on each instance 20 times using 10 minutes for each run (3GHz Pentium 4). In Table 1 the quality of the average solution is compared to SAHA followed by comparisons of the standard deviation, the worst solution found and the best solution found. We have also done a single 6 hour long run for each instance. It would take less than 6 days to do these experiments on a single processor.

The best average results, the best standard deviations and the largest minimum results are underlined in the table. Disregarding the 6 hour runs the best of the maximum results are also underlined in the table. Note that the varying computation times of SAHA makes it difficult to compare results, but most of the results (10 out of 15) are obtained using more than the 600 seconds used by 2DNEST.

The quality of a solution is given as a utilization percentage, that is, the percentage of area covered by the stencils in the resulting rectangular strip. Average results by 2DNEST are in general better. The exceptions are Dagli, Shapes2 and Swim for which the average is better for SAHA. The best solutions for these instances are also found by SAHA and this is also the case for Dighe1, Dighe2, Shirts and Trousers. The two latter ones are beaten by the single 6 hour run though. The jigsaw puzzles (Dighe1 and Dighe2) are actually also handled quite well by 2DNEST, but it is not quite able to achieve 100% utilization. Disregarding the jigsaw puzzles we have found the best known solutions for 10 out of 13 instances.

The standard deviations and the minimum results are clearly better for 2DNEST with the exception of Shapes2 and Swim which are instances that are in general handled badly by 2DNEST compared to SAHA. At least for Swim this is likely to be related to the fact that this instance is very complicated with an average of almost 22 vertices per stencil. It probably requires more time or some multilevel approach e-g- using approximated stencils or leaving out small stencils early in the solution process. The latter is the approach taken by SAHA in their multi-stage scheme which is used for the last 3 instances in the table (Shirts, Swim and Trousers).

The best solutions produced by 2DNEST (including 6 hour runs) are presented in Figure 9.

9 Three-dimensional Nesting

Our fast translation method is not restricted to two dimensions. In this section we will describe how the method can be used for three-dimensional nesting, but we will not generalize the proofs from Section 6. Solutions for such problems have applications in the area of *Rapid Prototyping* [37], and Osogami [31] has done a small survey of existing solution methods.

9.1 Generalization to three dimensions

It is straightforward to design algorithms to translate polyhedra in three dimensions. Edges are replaced by *faces*, edge regions (areas) are replaced by *face regions* (volumes) and so forth. Positive and negative faces are also just a natural generalization of their edge counterparts. The only real problem is to efficiently calculate the face region $\mathbf{R}(f,g)$ between two faces f and g.

Data instance			Average		Std. Dev.		Minimum		Maximum		6 hours	Sec.
Name	Size	Deg.	2DNEST	SAHA	2DNEST	SAHA	2DNEST	SAHA	2DNEST	SAHA	2DNEST	SAHA
Albano	24	180°	86.96	84.70	0.32	1.23	86.12	83.27	87.44	87.43	87.88	2257
Dagli	30	180°	85.31	85.38	0.53	1.07	83.97	83.14	85.98	87.15	87.05	5110
Dighe1	16		93.93	82.13	5.16	3.90	86.57	74.68	99.86	100.00	99.84	83
Dighe2	10		93.11	84.17	5.42	6.84	81.81	75.73	99.95	100.00	93.02	22
Fu	12	90°	90.93	87.17	0.62	1.40	90.05	85.08	91.84	90.96	92.03	296
Jakobs1	25	90°	88.90	75.79	0.42	0.88	87.07	75.39	89.07	†*78.89	89.03	332
Jakobs2	25	90°	80.28	74.66	0.18	0.89	79.53	74.23	80.41	77.28	81.07	454
Mao	20	90°	82.67	80.72	0.87	0.87	81.07	78.93	85.15	82.54	85.15	8245
Marques	24	90°	88.73	86.88	0.25	0.81	88.08	85.31	89.17	88.14	89.82	7507
Shapes0	43		65.42	63.20	0.78	0.98	64.25	61.39	67.09	66.50	66.42	3914
Shapes1	43	180°	71.74	68.63	0.79	1.41	71.12	65.41	73.84	71.25	73.23	10314
Shapes2	28	180°	79.89	81.41	1.05	0.74	76.71	80.00	81.21	83.60	81.59	*2136
Shirts	99	180°	85.73	85.67	0.41	0.49	85.14	84.91	86.33	†86.79	87.38	10391
Swim	48	180°	70.27	72.28	0.69	0.97	69.41	70.63	71.53	74.37	72.49	6937
Trousers	64	180°	89.29	89.02	0.28	0.57	88.77	87.74	89.84	89.96	90.46	8588

Table 1: Comparison of our implementation 2DNEST and SAHA by Gomes and Oliveira [20]. For each data instance the number of stencils to be nested and the allowed rotation are given. Both algorithms have been run 20 times. Average, minimum and maximum utilization are given and it is supplemented by the standard deviation. 2DNEST uses 10 minutes (600 seconds) for each run which can be compared to the varying running times of SAHA in the final column (averages in seconds). The second to last column is the result of running 2DNEST once for 6 hours (3600 seconds).

*These values have been corrected compared to those given in Gomes and Oliveira [20].

†Better results were obtained by a more simple greedy approach (GLSHA) [20]: 81-67% for Jakobs1 and 86-80% for Shirts.

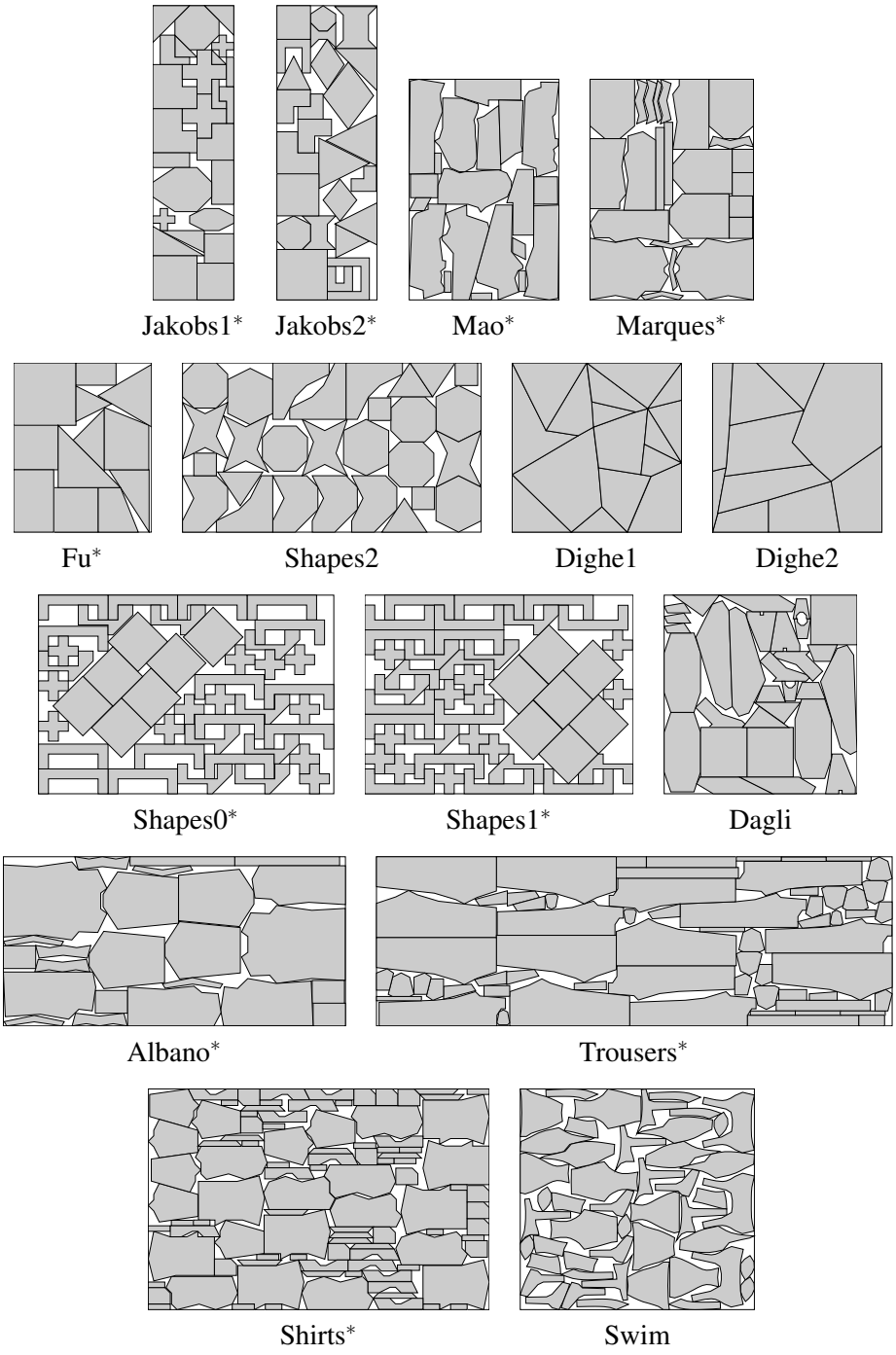


Figure 9: The best solutions found by 2DNEST easily comparable with the ones shown in Gomes and Oliveira [20].

*These solutions are also the currently best known solutions in the literature.

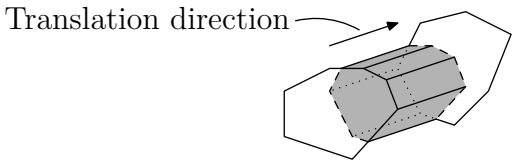


Figure 10: An illustration of the face region between two faces. The faces are not necessarily parallel, but the sides of the face region are parallel with the translation direction. The face region would be more complicated if the two faces were intersecting.

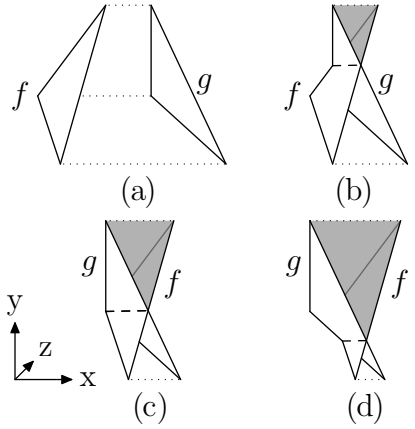


Figure 11: Translation of a triangle f through another triangle g along the x-axis, where the triangles have the same projection onto the yz-plane. The face region $\mathbf{R}(f,g)$ changes shape each time two corner points meet.

Assume that translation is done along the direction of the x-axis. An illustration of a face region is given in Figure 10. Note that the volume will not change if we simplify the two faces to the end faces of the face region. This can be done by projecting the faces onto the yz-plane, find and triangulate the intersection polygon and project this back onto the faces. This reduces the problem to the calculation of the volume of the face region between two triangles in three-dimensional space. We know that the three pairs of corner points will meet under translation. Sorted according to when they meet we will denote these the first, second and third breakpoint.

An illustration of the translation of two such triangles is given in Figure 11. Such a translation will almost always go through the following 4 phases.

1. No volume (Figure 11a).
2. After the first breakpoint the volume becomes a growing tetrahedron (Figure 11b).
3. The second breakpoint stops the tetrahedron (Figure 11c). The growing volume is now a bit harder to describe (Figure 11d) and we will take care of it in a moment.
4. After the third breakpoint the volume is growing linearly. It can be calculated as a constant plus the area of the projected triangle multiplied with the translation distance since the corner points met.

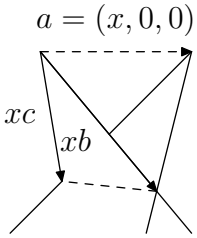


Figure 12: The volume of the above tetrahedron can be calculated from the three vectors **a**, **b** and **c**. In our case **b** and **c** are linearly dependent on x which is the length of **a** (and the translation distance since the tetrahedron started growing).

We have ignored 3 special cases of pairs of corner points meeting at the same time. 1) If the faces are parallel then we can simply skip to phase 4 and use a zero constant. 2) If the two last pairs of corner points meet at the same time then we can simply skip phase 3. 3) Finally, if the first two pairs of corner points meet at the same time we can skip phase 1. The reasoning for this is simple. Figure 11c illustrates that it is possible to cut a triangle into two parts which are easier to handle than the original triangle. The upper triangle is still a growing tetrahedron, but the lower triangle is a bit different. It is a tetrahedron growing from an edge instead of a corner and it can be calculated as a constant minus the area of a shrinking tetrahedron.

The basic function needed is therefore the volume $V(x)$ of a growing tetrahedron (a shrinking tetrahedron then follows easily). This can be done in several different ways, but one of them is especially suited for our purpose. Given three directional vectors $\mathbf{a}, \mathbf{b}, \mathbf{c}$ from one of the corner points of the tetrahedron, the following general formula can be used

$$V = \frac{1}{3!}|\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c})|. \tag{6}$$

In our case one of the vectors is parallel to the x-axis corresponding to the translation direction. An example of three vectors is given in Figure 12.

Since the angles of the tetrahedron are unchanged during translation, the vectors **b** and **c** do not change direction and can simply be scaled to match the current translation by the value x where x is the distance translated. This is indicated in the drawing. Using Equation 6, we can derive the following formula for the change of volume when translating:

$$\begin{aligned} V(x) &= \frac{1}{3!}|\mathbf{a} \cdot (x\mathbf{b} \times x\mathbf{c})| \\ &= \frac{1}{3!}\left| x^3 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \cdot \left(\begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} \times \begin{pmatrix} c_x \\ c_y \\ c_z \end{pmatrix} \right) \right| \\ &= \frac{1}{6}\left|(b_y c_z - b_z c_y)x^3\right|. \end{aligned}$$

However, this function is inadequate for our purpose since it is based on the assumption that the translation is 0 when $x = 0$. We need a translation offset t and by replacing x with $x - t$ we get:

$$V(x) = \frac{1}{6}\left|(b_y c_z - b_z c_y)(x^3 - 3tx^2 + 3t^2 x - t^3)\right|. \tag{7}$$

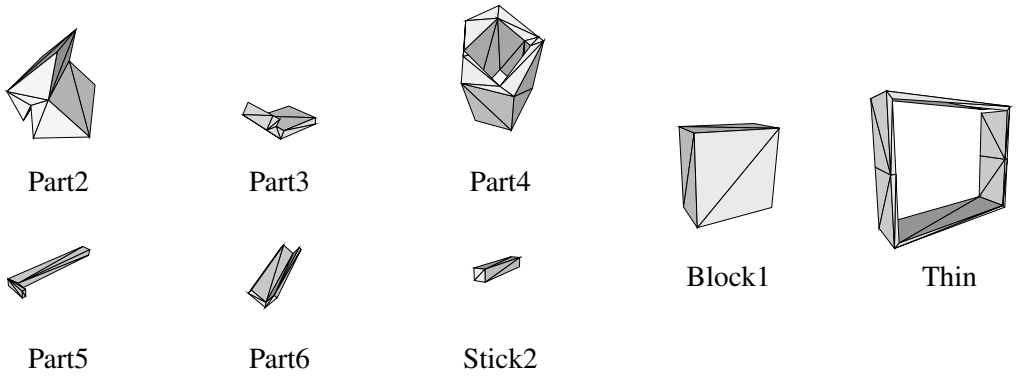


Figure 13: The Ikonen data set.

Now it is a simple matter to use Algorithm 2 in Section 6 for translating polyhedra with Equation 7 as breakpoint polynomials.

The volume function is a cubic polynomial for which addition and finding minimum are constant time operations. Assume we are given two polyhedra with m and n faces respectively (with an upper limit on the number of vertices for each face), then the running time of the three-dimensional variant of Algorithm 2 is exactly the same as for the two-dimensional variant: $O(mn\log(mn))$. However, the constants involved are larger.

9.2 Results for three dimensions

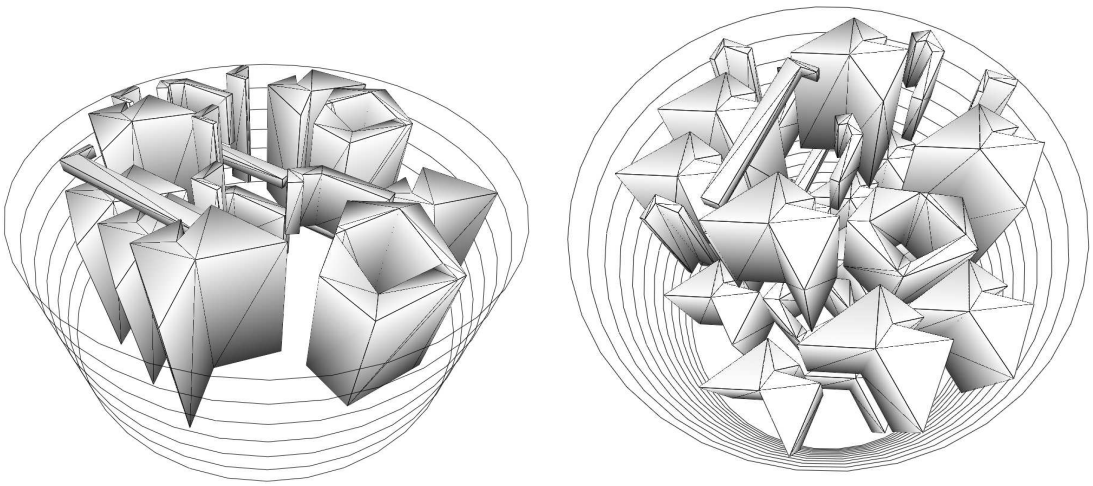
A prototype has been implemented, 3DNEST, and its performance has been compared with the very limited existing results. In the literature only one set of simple data instances has been used. They were originally created by Ilkka Ikonen and later used by Dickinson and Knopf [13] to compare their solution method with Ikonen et al. [23]. Eight objects are available in the set and they are presented in Table 2 and Figure 13. Some of them have holes, but they are generally quite simple. They can all be drawn in two dimensions and then just extended in the third dimension. They have no relation to real-world data instances.

Name	# Faces	Volume	Bounding box
Block1	12	4.00	$1.00 \times 2.00 \times 2.00$
Part2	24	2.88	$1.43 \times 1.70 \times 2.50$
Part3	28	0.30	$1.42 \times 0.62 \times 1.00$
Part4	52	2.22	$1.63 \times 2.00 \times 2.00$
Part5	20	0.16	$2.81 \times 0.56 \times 0.20$
Part6	20	0.24	$0.45 \times 0.51 \times 2.50$
Stick2	12	0.18	$2.00 \times 0.30 \times 0.30$
Thin	48	1.25	$1.00 \times 3.00 \times 3.50$

Table 2: The Ikonen data set.

Based on these objects two test cases were created by Dickinson and Knopf for their experiments.

- Case 1



Case 1 Case 2

Figure 14: The above illustrations contain twice as many objects as originally intended in Ikonens Case 1 and 2. They only took a few seconds to find.

Pack 10 objects into a cylinder of radius 3.4 and height 3.0. The 10 objects were chosen as follows: 3 $\times$ Part2, 1 Part4 and 2 $\times$ Part3, Part5 and Part6. Total number of faces is 260 and 11.3% of the total volume is filled.

- Case 2
  Pack 15 objects into a cylinder of radius 3.5 and height 5.5. The 15 objects were chosen as in case 1, but with 5 more Part2. Total number of faces is 380 and 12.6% of the total volume is filled.

Dickinson and Knopf report execution times for both their own solution method (serial packing) and the one by Ikonen et al. (genetic algorithm) and they ran the benchmarks on a 200 MHz AMD K6 processor. The results are presented in Table 3 in which results from our algorithm are included.

Our initial placement is a random placement which could be a problem since it would quite likely contain almost no overlap and then it would not say much about our algorithm — especially the GLS part. To make the two cases a bit harder we *doubled* the number of objects. Our tests were run on a 733MHz G4. Even considering the difference in processor speeds there is no doubt that our method is the fastest for these instances. Illustrations of the resulting placements can be seen in Figure 14.

Test	Ikonen et al.	Dickinson and Knopf	3DNEST
Case 1	22.13 min.	45.55 sec.	3.2 sec. (162 translations)
Case 2	26.00 min.	81.65 sec.	8.1 sec. (379 translations)

Table 3: Execution times for 3 different heuristic approaches. Note that the number of objects is doubled for 3DNEST.

10 Conclusion

We have presented a new solution method for nesting problems. The solution method uses local search to reduce the amount of overlap in a greedy fashion and it uses Guided Local Search to escape local minima. To find new positions for stencils which decrease the total overlap, we have developed a new algorithm which determines a horizontal or vertical translation of a polygon with least overlap. Furthermore, our solution method can easily be extended to handle otherwise complicated requirements such as free rotation and quality regions.

The solution method has also been implemented and is in most cases able to produce better solutions than those previously published. It is also robust with very good average solutions and small standard deviations compared to previously published solutions methods, and this is within a reasonable time limit of 10 minutes per run.

Finally we have generalized the method to three dimensions which enables us to also solve three-dimensional nesting problems.

Acknowledgments

We would like to thank A. Miguel Gomes and José F. Oliveira for providing additional data on the performance of their solution method [20]. We would also like to thank Martin Zachariasen and the anonymous referees for some valuable remarks.

References

[1] M. Adamowicz and A. Albano. Nesting two-dimensional shapes in rectangular modules. *Computer Aided Design*, 1:27–33, 1976.

[2] A. Albano and G. Sappupo. Optimal allocation of two-dimensional irregular shapes using heuristic search methods. *IEEE Transactions on Systems, Man and Cybernetics*, 5:242–248, 1980.

[3] R. C. Art, Jr. An approach to the two dimensional, irregular cutting stock problem. Technical Report 36.Y08, IBM Cambridge Scientific Center, September 1966.

[4] T. Asano, A. Hernández-Barrera, and S. C. Nandy. Translating a convex polyhedron over monotone polyhedra. *Computational Geometry*, 23(3):257–269, 2002. doi: 10.1016/S0925-7721(02)00098-6.

[5] J. A. Bennell and K. A. Dowsland. Hybridising tabu search with optimisation techniques for irregular stock cutting. *Management Science*, 47(8):1160–1172, 2001.

[6] J. A. Bennell and K. A. Dowsland. A tabu thresholding implementation for the irregular stock cutting problem. *International Journal of Production Research*, 37:4259–4275, 1999.

[7] J. Blazewicz and R. Walkowiak. A local search approach for two-dimensional irregular cutting. *OR Spektrum*, 17:93–98, 1995.

[8] J. Blazewicz, P. Hawryluk, and R. Walkowiak. Using a tabu search approach for solving the two-dimensional irregular cutting problem. *Annals of Operations Research*, 41:313–325, 1993.

[9] E. K. Burke and G. Kendall. Applying simulated annealing and the no fit polygon to the nesting problem. In *Proceedings of the World Manufacturing Congres*, pages 70–76. ICSC Academic Press, 1999.

[10] E. K. Burke and G. Kendall. Applying ant algorithms and the no fit polygon to the nesting problem. In *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence (AI'99)*, volume 1747, pages 454–464. Springer Lecture Notes in Artificial Intelligence, 1999.

[11] E. K. Burke and G. Kendall. Applying evolutionary algorithms and the no fit polygon to the nesting problem. In *Proceedings of the 1999 International Conference on Artificial Intelligence (IC-AI'99)*, volume 1, pages 51–57. CSREA Press, 1999.

[12] P. Chen, Z. Fu, A. Lim, and B. Rodrigues. The two dimensional packing problem for irregular objects. *International Journal on Artificial Intelligent Tools*, 2004.

[13] J. K. Dickinson and G. K. Knopf. Serial packing of arbitrary 3d objects for optimizing layered manufacturing. In *Intelligent Robots and Computer Vision XVII*, volume 3522, pages 130–138, 1998.

[14] D. Dobkin, J. Hershberger, D. Kirkpatrick, and S. Suri. Computing the intersection-depth of polyhedra. *Algorithmica*, 9:518–533, 1993.

[15] K. A. Dowsland and W. B. Dowsland. Solution approaches to irregular nesting problems. *European Journal of Operational Research*, 84:506–521, 1995.

[16] K. A. Dowsland, W. B. Dowsland, and J. A. Bennell. Jostling for position: Local improvement for irregular cutting patterns. *Journal of the Operational Research Society*, 49:647–658, 1998.

[17] K. A. Dowsland, S. Vaid, and W. B. Dowsland. An algorithm for polygon placement using a bottom-left strategy. *European Journal of Operational Research*, 141:371–381, 2002.

[18] 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.

[19] A. M. Gomes and J. F. Oliveira. A 2-exchange heuristic for nesting problems. *European Journal of Operational Research*, 141:359–370, 2002.

[20] A. M. Gomes and J. F. Oliveira. Solving irregular strip packing problems by hybridising simulated annealing and linear programming. *European Journal of Operational Research*, 171(3):811–829, 2006.

[21] R. Heckmann and T. Lengauer. A simulated annealing approach to the nesting problem in the textile manufacturing industry. *Annals of Operations Research*, 57:103–133, 1995.

[22] J. Heistermann and T. Lengauer. The nesting problem in the leather manufacturing industry. *Annals of Operations Research*, 57:147–173, 1995.

[23] I. Ikonen, W. E. Biles, A. Kumar, J. C. Wissel, and R. K. Ragade. A genetic algorithm for packing three-dimensional non-convex objects having cavities and holes. In *Proceedings of the 7th International Conference on Genetic Algortithms*, pages 591–598, East Lansing, Michigan, 1997. Morgan Kaufmann Publishers.

[24] P. Jain, P. Fenyes, and R. Richter. Optimal blank nesting using simulated annealing. *Journal of mechanical design*, 114:160–165, 1992.

[25] S. Jakobs. On genetic algorithms for the packing of polygons. *European Journal of Operational Research*, 88:165–181, 1996.

[26] Z. Li and V. Milenkovic. Compaction and separation algorithms for non-convex polygons and their applications. *European Journal of Operational Research*, 84:539–561, 1995.

[27] H. Lutfiyya, B. McMillin, P. Poshyanonda, and C. Dagli. Composite stock cutting through simulated annealing. *Journal of Mathematical and Computer Modelling*, 16(2):57–74, 1992.

[28] B. K. Nielsen and A. Odgaard. Fast neighborhood search for the nesting problem. Technical Report 03/03, DIKU, Department of Computer Science, University of Copenhagen, 2003.

[29] J. F. Oliveira and J. S. Ferreira. Algorithms for nesting problems. *Applied Simulated Annealing*, pages 255–273, 1993.

[30] J. F. Oliveira, A. M. Gomes, and J. S. Ferreira. TOPOS - a new constructive algorithm for nesting problems. *OR Spektrum*, 22:263–284, 2000.

[31] T. Osogami. Approaches to 3D free-form cutting and packing problems and their applications: A survey. Technical Report RT0287, IBM Research, Tokyo Research Laboratory, 1998.

[32] Y. Stoyan, G. Scheithauer, N. Gil, and T. Romanova. Φ-functions for complex 2d-objects. *4OR: Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 2(1):69–84, 2004.

[33] V. E. Theodoracatos and J. L. Grimsley. The optimal packing of arbitrarily-shaped polygons using simulated annealing and polynomial-time cooling schedules. *Computer methods in applied mechanics and engineering*, 125:53–70, 1995.

[34] C. Voudouris and E. Tsang. Guided local search. Technical Report CSM-147, Department of Computer Science, University of Essex, Colchester, C04 3SQ, UK, August 1995.

[35] C. Voudouris and E. Tsang. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113:469–499, 1999.

[36] G. Wäscher, H. Haussner, and H. Schumann. Translating a convex polyhedron over monotone polyhedra. *European Journal of Operational Research*, this issue, 2006.

[37] X. Yan and P. Gu. A review of rapid prototyping technologies and systems. *Computer Aided Design*, 28 (4):307–318, 1996.

Addendum to "Fast neighborhood search for two- and three-dimensional nesting problems"

Jens Egeblad

1 Implemetation Background

A number of implementation details were missing from the paper [A]. In this addendum we present new experiments in Section 2 and elaborate on some of the missing details in Section 3.

The work presented in the paper [A] is based on older work by Egeblad et al. [1] and the first implementation from 2001 already showed results which were competitive with the best results from the literature. The implementation was later completely rewritten as part of the Master's thesis by Nielsen and Odgaard [4] and this implementation was slightly modified and improved for the first paper [A]. The implementation was heavily modified again to improve floating point stability issues, to handle the strip-packing variant of three-dimensional problems of the paper [B], to allow for analysis of overlap measures and handling of free rotation by Nielsen [6], to handle repeated pattern nesting by Nielsen [5], and to manage the objective function of the paper [E]. Several parts of the implementation were made more efficient, especially for the three-dimensional problems.

2 New Experiments

Experiments from the paper [A] were rerun with the new implementation (see also Nielsen [6]) and a comparison of the new implementation (Current NEST2D) and the old implementation (Old 2DNEST) as well as two other state-of-the-art heuristics by Gomes and Oliveira [2] (SAHA) and Imamichi et al. [3] (ILSQN) are shown in Table 1. The results of both (Old 2DNEST) and (Current NEST2D) were over 20 runs and the running times for each run in both implementations were 600 seconds, although on two different processors. Running times vary for SAHA (see [A] for more details) while results for ILSQN are from 10 runs of 600 or 1200 seconds depending on the size of the problem.

Best results and average results over the 20 runs for 2DNEST and NEST2D, and average results for the other heuristics are presented in the table. It can be seen from this table that both the old and in particular the new implementation are still competitive with the other heuristics from the literature, the average overall utilization of the average results of NEST2D are 84.93, while the average results of the best achieved utilization is 85.75. This almost matches ILSQN which has an overall average of averages equal to 82.74 and overall average of best results equal to 85.89 – Only 0.14 percentage points better than NEST2D.

It is important to note that the running times are for the complete optimization phase and not for solving the last decision variant. However, the majority of the running time is spend solving the decision problem on the last few strip-lengths.

3 Implementation Details

A number of interesting details are omitted from the paper [A] and we discuss them briefly here:

			Average results				Best results			
	Size	Deg.	Old 2DNEST	Current NEST2D	ILSQN	SAHA	Old 2DNEST	Current NEST2D	ILSQN	SAHA
Albano	24	180°	86.96	87.50	87.14	84.70	87.44	87.90	88.16	87.43
Dagli	30	180°	85.31	86.17	85.80	85.38	85.98	87.51	87.40	87.15
Dighe1	16		93.93	99.99	90.49	82.13	99.86	100.00	99.89	100.00
Dighe2	10		93.11	99.99	84.21	84.17	99.95	100.00	99.99	100.00
Fu	12	90°	90.93	91.03	87.57	87.17	91.84	91.94	90.67	90.96
Jakobs1	25	90°	88.90	89.05	84.78	75.79	89.07	89.09	86.89	78.89
Jakobs2	25	90°	80.28	80.71	80.50	74.66	80.41	82.44	82.51	77.28
Mao	20	90°	82.67	83.08	81.31	80.72	85.15	84.23	83.44	82.54
Marques	24	90°	88.73	89.12	86.81	86.88	89.17	89.85	89.03	88.14
Shapes0	43		65.42	66.07	66.49	63.20	67.09	66.74	68.44	66.50
Shapes1	43	180°	71.74	72.35	72.83	68.63	73.84	73.83	73.84	71.25
Shapes2	28	180°	79.89	80.69	81.72	81.41	81.21	82.32	84.25	83.60
Shirts	99	180°	85.73	86.61	88.12	85.67	86.33	87.35	88.78	86.79
Swim	48	180°	70.27	72.00	74.62	72.28	71.53	73.04	75.29	74.37
Trousers	64	180°	89.29	89.64	88.69	89.02	89.84	90.04	89.79	89.96
			83.54	84.93	82.74	80.12	85.25	85.75	85.89	84.32

Table 1: Average and best results are compared for the four different solution methods to nesting. The highest utilization values are underlined. The number of shapes and the degree of rotation allowed are also reported. Processors are 3.0 GHz Intel Pentium IV (Old 2DNEST) 2.16 GHz Intel Core Duo (Current NEST2D), 2.8GHz Intel Xeon (ILSQN), and 2.4GHz Pentium IV (SAHA).

Resetting penalties From time to time penalties are reset to 0 for two reasons. Firstly, because this ensures that the augmented objective function is never too far from the actual objective function. Secondly, because it 'kicks' the heuristic out of the current solution state and allows for masively new changes. A reset is conducted every $5 \cdot n^2$ iterations where n is the number of items. This value was found through parameter tuning.

Penalties in the overlap algorithm Penalties must be included when overlap is calculated in the algorithm which finds the minimal overlap translation between a polygon p and a set of polygons $P \setminus \{p\}$. To do this, overlap between p and the individual polygons is maintained as the algorithm traverses the list of breakpoints. If the overlap for an individual polygon q increases beyond zero the penalty of the overlap of p and q is added to the objective function. When it decreases to zero again it is subtracted. This can be done without increasing the asymptotic running time since each breakpoint comes from one edge of exactly one polygon from $P \setminus \{p\}$, and therefore, each breakpoint only requires update of the overlap of one polygon, which can be done in constant time.

Rotations In the original work by Egeblad et al. [1] rotations were handled differently than translations. A normal overlap algorithm was implemented to return the overlap of a pair of polygons. The overlap of each rotation angle considered was measured using this algorithm, while the minimal overlap translation algorithm was used to determine overlap of translations. Because of small floating point errors the two algorithms could return slightly different overlap values for the same position and rotation. This could cause the heuristic to cycle through the same two placements infinitely since one placement would seem to reduce the overlap when calculated with one of the algorithms and another with respect to the other algorithm. To remedy this problem the implementation used by Nielsen and Odgaard [4] and in later papers use the same algorithm for rotation and translation. This is done by considering a full horizontal translation for each rotation angle considered.

Iterations The paper reports running times but does not detail the number of iterations. However, for the instances tested in the paper, the number of translations considered range between 1,000 to 26,000 per second depending on the complexity of the instance. Since two translations are considered for each polygon, and an additionally number of translations for each rotation, the number of iterations per second is roughly between 250 and 13,000, and the full number of iterations ranges between 150,000 and 78,000,000 for a complete 600 second run.

Decreasing the strip-length The strip-length L is decreased by setting it to $L = L' \times (1 - \varepsilon)$, where L' is length of the last solved decision problem. Initially ε is set to 0.01, i.e. the strip-length is decreased by 1% between each decision problem. However, if no solution to a decision problem has been found within $10 \cdot n^2$, then the heuristic updates ε by setting it to $\varepsilon = 0.7 \cdot \varepsilon'$ where ε' is the last used value of ε.

References

[A] J. Egeblad, B. K. Nielsen, and A. Odgaard. Fast neighborhood search for two- and three-dimensional nesting problems. *European Journal of Operational Research*, 183(3):1249–1266, 2007.

[1] J Egeblad, B. K. Nielsen, and A. Odgaard. Metaheuristikken *guided local search* anvendt på pakning af irregulære polygoner. (Project at DIKU), 2001.

[2] A. M. Gomes and J. F. Oliveira. Solving irregular strip packing problems by hybridising simulated annealing and linear programming. *European Journal of Operational Research*, 171(3):811–829, 2006.

[3] T. Imamichi, M. Yagiura, and H. Nagamochi. An iterated local search algorithm based on nonlinear programming for the irregular strip packing problem. In *Proceedings of the Third International Symposium on Scheduling, Tokyo Japan*, pages 132–137, 2006.

[4] B. K. Nielsen and A. Odgaard. Fast neighborhood search for the nesting problem. Technical Report 03/03, DIKU, Department of Computer Science, University of Copenhagen, 2003.

[5] Benny K. Nielsen. An efficient solution method for relaxed variants of the nesting problem. In Joachim Gudmundsson and Barry Jay, editors, *Theory of Computing, Proceedings of the Thirteenth Computing: The Australasian Theory Symposium*, volume 65 of *CRPIT*, pages 123–130, Ballarat, Australia, 2007. ACS.

[6] Benny Kjær Nielsen. *Nesting Problems and Steiner Tree Problems*. PhD thesis, DIKU, University of Copenhagen, Denmark, 2008.

Accepted for publication in the *Computational Geometry: Theory and Applications*, 2008

Translational packing of arbitrary polytopes

Jens Egeblad* Benny K. Nielsen* Marcus Brazil†

Abstract

We present an efficient solution method for packing d-dimensional polytopes within the bounds of a polytope container. The central geometric operation of the method is an exact one-dimensional translation of a given polytope to a position which minimizes its volume of overlap with all other polytopes. We give a detailed description and a proof of a simple algorithm for this operation in which one only needs to know the set of $(d-1)$-dimensional facets in each polytope. Handling non-convex polytopes or even interior holes is a natural part of this algorithm. The translation algorithm is used as part of a local search heuristic and a meta-heuristic technique, guided local search, is used to escape local minima. Additional details are given for the three-dimensional case and results are reported for the problem of packing polyhedra in a rectangular parallelepiped. Utilization of container space is improved by an average of more than 14 percentage points compared to previous methods.

The translation algorithm can also be used to solve the problem of maximizing the volume of intersection of two polytopes given a fixed translation direction. For two polytopes with complexity $O(n)$ and $O(m)$ and a fixed dimension, the running time is $O(nm\log(nm))$ for both the minimization and maximization variants of the translation algorithm.

Keywords: Packing, heuristics, translational packing, packing polytopes, minimizing overlap, maximizing overlap, strip-packing, guided local search

1 Introduction

Three-dimensional packing problems have applications in various industries, e.g., when items must be loaded and transported in shipping containers. The three main problems are bin-packing, knapsack packing, and container loading. In bin-packing the minimum number of equally-sized containers sufficient to pack a set of items must be determined. In knapsack packing one is given a container with fixed dimensions and a set of items, each with a profit value; one must select a maximum profit subset of the items which may be packed within the container. The container loading problem is a special case of the knapsack problem where the profit value of each item is set to its volume. Bin-packing, knapsack packing, and container loading problems involving boxes are classified as orthogonal packing problems and are well-studied in the literature.

In general, three-dimensional packing problems can also involve more complicated shapes; it is not only boxes that are packed in shipping containers. An interesting example is *rapid prototyping* which is a term originally used for the production of physical prototypes of 3D computer aided design (CAD) models needed in the early design or test phases of new products. Nowadays, rapid prototyping

*Department of Computer Science, University of Copenhagen, DK-2100 Copenhagen Ø, Denmark. E-mail: {jegeblad, benny}@diku.dk.

†ARC Special Research Centre for Ultra-Broadband Information Networks (CUBIN) an affiliated program of National ICT Australia, Department of Electrical and Electronic Engineering, The University of Melbourne, Victoria 3010, Australia. E-mail: brazil@ee.unimelb.edu.au.

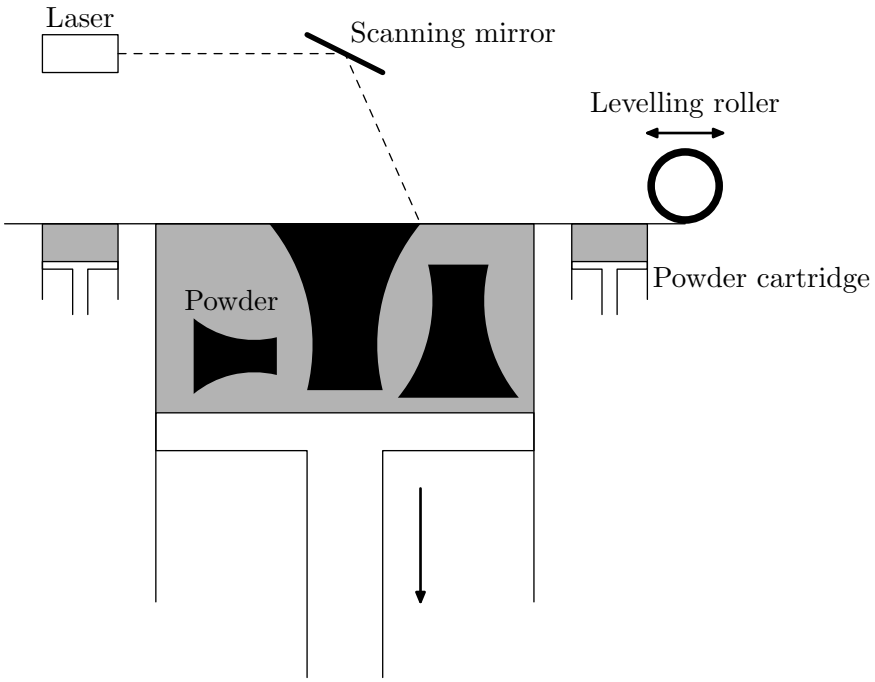


Figure 1: An illustration of a typical machine for rapid prototyping. The powder is added one layer at a time and the laser is used to sinter what should be solidified to produce the desired objects.

technologies are also used for manufacturing purposes. One of these technologies, *selective laser sintering process*, is depicted in Figure 1. The idea is to build up the object(s) by adding one very thin layer at a time. This is done by rolling out a thin layer of powder and then sintering (heating) the areas/lines which should be solid by the use of a laser. The unsintered powder supports the objects built and therefore no pillars or bridges have to be made to account for gravitational effects. This procedure takes hours ("rapid" when related to weeks) and since the time required for the laser is significantly less than the time required for preparing a layer of powder, it will be an advantage to have as many objects as possible built in one run of the machine. A survey of rapid prototyping technologies is given by Yan and Gu [31].

In order to minimize the time used by the rapid prototype machine items must be placed as densely as possible and the number of layers must be minimized. The problem of minimizing layers may therefore be formulated as a strip-packing problem: A number of items must be placed within a container such that the container height is minimized.

In this paper we present a solution method for the multidimensional strip-packing problem. However, our techniques may be applied to some of the other problem variants, e.g., bin-packing. Specifically, for three dimensions, we pack a number of arbitrary (both convex and non-convex) polyhedra in a parallelepiped such that one of the parallelepiped's dimensions is minimized. No rotation is allowed and gravity is not considered. A formal description of the problem is given in Section 2 and a review of related work is given in Section 3.

The solution method described in this paper generalizes previous work by Egeblad et al. [A]. This earlier paper focused on the two-dimensional variant of this problem which is generally known as the nesting problem (packing polygons in a rectangle), but also included a short description and some results for a three-dimensional generalization. In both cases overlap is iteratively reduced by a

central algorithm which determines a one-dimensional translation of a given polygon/polyhedra to a minimum overlap position.

Egeblad et al. only prove the correctness of the two-dimensional variant. In this paper, we prove the correctness of the translation algorithm in three and higher dimensions (Section 4), essentially describing a solution method for packing polytopes in d-dimensional space. We also give a more detailed description of the translation algorithm in three dimensions. The complete solution method is described in Section 5.

Because applications for $d > 3$ are not obvious, an implementation has only been done for the three-dimensional case. Experimental results are presented in Section 6 and compared with existing results from the literature. Finally, some concluding remarks are given in Section 7.

2 Problem description

The main problem considered in this paper is as follows:

The 3D Decision Packing Problem (3DDPP). *Given a set of polyhedra $\mathcal{S}$ and a polyhedral container C, determine whether a non-overlapping translational placement of the polyhedra within the bounds of the container exists.*

This problem is $\mathcal{NP}$-complete even if all polyhedra in $\mathcal{S}$ are cubes [16]. If $\nu(P)$ denotes the volume of a polyhedron P and this is generalized for sets such that $\nu(\mathcal{S}) = \sum_{P\in\mathcal{S}} \nu(P)$ then a non-overlapping placement for the 3DDPP has a *utilization* (of the container) of $\nu(\mathcal{S})/\nu(C)$. Based on the decision problem we can define the following optimization problem.

The 3D Strip Packing Problem (3DSPP). *Given a set of polyhedra $\mathcal{S}$ and a rectangular parallelepiped C (the container) with fixed width w and length l, find the minimum height h of the container for which the answer to the 3D decision packing problem is positive.*

An optimal solution to 3DSPP has a utilization of $\nu(\mathcal{S})/\nu(C) = \nu(\mathcal{S})/(w \cdot l \cdot h)$, i.e., the utilization only depends on the height of the parallelepiped and not on a particular placement corresponding to this height. The word *strip* is based on the terminology used for the 2-dimensional variant of the problem.

While the solution method discussed in the following sections could be applied to the bin-packing problem or other variants of multi-dimensional packing problems, we limit our description to the strip-packing problem. The strip-packing variant has been chosen mainly because it allows a comparison with results from the existing literature. In the typology of Wäscher et al. [30], the problem we consider, 3DSPP, is a three-dimensional irregular open dimension problem (ODP) with fixed orientations of the polyhedra.

The polyhedra handled in this paper are very general. Informally, a polyhedron can be described as a solid whose boundary consists of a finite number of polygonal faces. Note that every face must separate the exterior and the interior of the polyhedron, but convexity is not required and holes and interior voids are allowed. A polyhedron is even allowed to consist of several disconnected parts and holes may contain smaller individual parts.

The problem formulations above are easily generalized to higher dimensions. Simply replace polyhedron with polytope and consider a rectangular d-dimensional parallelepiped for the strip-packing problem. We denote the corresponding problems dDDPP and dDSPP, where d is the number of dimensions. For simplicity, the faces are required to be convex, but this is not a restriction on the types

of polytopes allowed, as a non-convex face can be partitioned into a finite number of convex faces. The polytopes themselves can be non-convex and contain holes.

Since dDDPP is $\mathcal{NP}$-complete dDSPP is an $\mathcal{NP}$-hard problem. Our solution method for dDSPP is heuristic and therefore not guaranteed to find the optimal solution of dDSPP.

When solving a problem in 3D for physical applications such as rapid prototyping, one should be aware that some feasible solutions are not very useful in practice since objects may be interlocked. Avoiding this is a very difficult constraint which is not considered in this paper.

3 Related work

Cutting and packing problems have received a lot of attention in the literature, but focus has mainly been on one or two dimensions and also often restricted to simple shapes such as boxes. A survey of the extensive 2D packing literature is given by Sweeney and Paternoster [28] and a survey of the 2D packing literature concerning irregular shapes (nesting) is given by Dowsland and Dowsland [12]. Recent heuristic methods for orthogonal packing problems include the work of Lodi et al. [22] and Faroe et al. [15] for the bin-packing problem, and Bortfeldt et al. [2] and Eley [14] for the container loading problem. The meta-heuristic approach utilized in this paper is based on the ideas presented by Faroe et al.

In the following, we review solution methods presented for packing problems in more than two dimensions which also involve shapes more general than boxes. A survey is given by Cagan et al. [4] in the broader context of *three-dimensional layout problems* for which maximum utilization may not be the only objective. Their focus is mainly on various meta-heuristic approaches to the problems, but a section is also dedicated to approaches for determining intersections of shapes. A survey on *3D free form packing and cutting problems* is given by Osogami [26]. This covers applications in both rapid prototyping in which maximum utilization is the primary objective and applications in product layout in which other objectives, e.g., involving electrical wire routing length and gravity are more important.

Ikonen et al. [20] have developed one of the earliest approaches to a non-rectangular 3D packing problem. Using a genetic algorithm they can handle non-convex shapes with holes and a fixed number of orientations (45° increments on all three axes). To evaluate if two shapes overlap, their bounding boxes (the smallest axis-aligned circumscribing box) are first tested for intersection and, if they intersect, triangles are subsequently tested for intersection. For each pair of intersecting triangles it is calculated how much each edge of each triangle intersects the opposite triangle.

Cagan et al. [3] use the meta-heuristic *simulated annealing* and they allow rotation. They can also handle various additional optimization objectives such as routing lengths. Intersection checks are done using *octree decompositions* of shapes. As the annealing progresses the highest resolution is increased to improve accuracy. Improvements of this work using variants of the meta-heuristic *pattern search* instead of simulated annealing are later described by Yin and Cagan, Yin and Cagan [32, 33].

Dickinson and Knopf [10] focus on maximizing utilization, but they introduce an alternative metric to determine the compactness of a given placement of shapes. In short, this metric measures the compactness of the remaining free space. The best free space, in three dimensions, is in the form of a sphere. The metric is later used by Dickinson and Knopf [11] with a sequential placement algorithm for three-dimensional shapes. Items are placed one-by-one according to a predetermined sequence and each item is placed at the best position as determined by the free-space metric. To evaluate if two shapes overlap they use depth-maps. For each of the six sides of the bounding box of each shape, they divide the box-side into a uniform two-dimensional grid and store the distance perpendicular

to the box-side from each grid cell to the shape's surface. Determining if two shapes overlap now amounts to testing the distance at all overlapping grid points of bounding box sides, when the sides are projected to two dimensions. For each shape up to 10 orientations around each of its rotational axes are allowed. Note that the free-space metric is also generalized for higher dimensions and thus the packing algorithm could potentially work in higher dimensions.

Hur et al. [19] use voxels, a three-dimensional uniform grid structure, to represent shapes. As for the octree decomposition technique, each grid-cell is marked as full if a part of the associated shape is contained within the cell. The use of voxels allows for simple evaluation of overlap of two shapes since overlap only occurs if one or more overlapping grid cells from both shapes are marked as full. Hur et al. [19] also use a sequential placement algorithm and a modified bottom-left strategy which always tries to place the next item of the sequence close to the center of the container. A genetic algorithm is used to iteratively modify the sequence and reposition the shapes.

Eisenbrand et al. [13] investigate a special packing problem where the maximum number of uniform boxes that can be placed in the trunk of a car must be determined. This includes free orientation of the boxes. For any placement of boxes they define a potential function that describes the total overlap and penetration depth between boxes and trunk sides and of pairs of boxes. Boxes are now created, destroyed, and moved randomly, and simulated annealing is used to decide if new placements should be accepted.

Recently, Stoyan et al. [27] presented a solution method for 3DSPP handling convex polyhedra only (without rotation). The solution method is based on a mathematical model and it is shown how locally optimal solutions can be found. Stoyan et al. [27] use Φ-functions to model non-intersection requirements. A Φ-function for a pair of shapes is defined as a real-value calculated from their relative placement. If the shapes overlap, abut or do not overlap the value of the Φ-function is larger than, equal or less than 0, respectively. A tree-search is proposed to solve the problem to optimality, but due to the size of the solution space Stoyan et al. opt for a method that finds locally optimal solutions instead. Computational results are presented for three problem instances with up to 25 polyhedra. A comparison with the results of the solution method presented in this paper can be found in Section 6.

4 Axis-Aligned Translation

As mentioned in the introduction, our solution method for packing polytopes is based on an algorithm for translating a given polytope to a minimum volume of overlap position in an axis-aligned direction. This problem is polynomial-time solvable and we present an efficient algorithm for it here. The algorithm can easily be modified to determine a maximum overlap translation. Note that the position of a polytope is specified by the position of a given reference point on the polytope; hence its position corresponds to a single point.

Without loss of generality, we assume that the translation under consideration is an x-axis-aligned translation. In three dimensions, the problem we are solving can be stated as follows:

1-Dimensional Translation Problem in 3D (1D3DTP). *Given a fixed polyhedral container C, a polyhedron Q with fixed position, and a polyhedron P with fixed position with respect to its y and z coordinates, find a horizontal offset x for P such that the volume of overlap between P and Q is minimized (and P is within the bounds of the container C).*

By replacing the term polyhedron with polytope this definition can easily be generalized to higher dimensions, in which case we denote it 1DdDTP. In Section 4.1 we give a more formal definition and present a number of properties of polytopes which we use in Section 4.2 to prove the correctness of an

algorithm for 1DdDTP. Since the algorithm solves the problem of finding a minimum overlap position of P we refer to it in the following as the translation algorithm. In Section 4.3 we provide additional details for the three dimensional case.

Egeblad et al. [A] have proved the correctness of the two-dimensional special case of the algorithm described in the following and they have also sketched how the ideas can be generalized to 3D. Here we flesh out the approach sketched by Egeblad et al., and generalize it to d dimensions.

4.1 Polytopes and their Intersections

In the literature, the term *polytope* is usually synonymous with *convex polytope*, which can be thought of as the convex hull of a finite set of points in d-dimensional space, or, equivalently, a bounded intersection of a finite number of half-spaces. In this paper we use the term *polytope* to refer to a more general class of regions in d-dimensional space, which may be non-convex and can be formed from a finite union of convex polytopes.

By definition, we assume that the boundary of a polytope P is composed of *faces*, each of which is a convex polytope of dimension less than d. Following standard notation, we refer to a one-dimensional face of P as an *edge*, and a zero-dimensional face as a *vertex*. The faces must satisfy the following properties:

1. The $(d-1)$-dimensional faces of P (which we refer to as *facets*) have the property that two facets do not intersect in their interiors.

2. The facets must be simple, i.e., on the boundary of a facet each vertex must be adjacent to exactly $d-1$ edges. Note that this only affects polytopes of dimension 4 or more.

3. Each face of P of dimension $k < d-1$ lies on the boundary of at least two faces of P of dimension $k+1$ (and hence, by induction, on the boundary of at least two facets).

Note that our definition of a polytope allows two adjacent facets to lie in the same hyperplane. This allows our polytopes to be as general as possible, while imposing the condition that all faces are convex (by partitioning any non-convex facets into convex $(d-1)$-dimensional polytopes). Most importantly, our definition of polytopes does not require boundaries to be triangulated in 3D. Note that such a requirement would only allow minor simplifications in the proofs and the algorithm described later in this paper.

Given a polytope P, we write $p \in P$ if and only if p is a point of P including the boundary. More importantly, we write $p \in \text{int}(P)$ if and only if p is an interior point of P, i.e., $\exists \varepsilon > 0 : \forall p' \in \mathbb{R}^d, \Rightarrow ||p - p'|| < \varepsilon,\ p' \in P$.

We next introduce some new definitions and concepts required to prove the correctness of the translation algorithm in d dimensions.

Let e_i be the ith coordinate system basis vector, e.g., $e_1 = (1,0,\ldots,0)^T$. As stated earlier we only consider translations in the direction of the x-axis (that is, with direction $\pm e_1$). This helps to simplify the definitions and theorems of this section without loss of generality since translations along other axes work in a similar fashion. In the remainder of this section it is convenient to refer to the direction $-e_1$ as *left* and e_1 as *right*.

Given a polytope P, we divide the points of the boundary of P into three groups, *positive*, *negative*, and *neutral*.

Definition 9 (Signs of a Boundary Point)**.** *Suppose p is a point of the boundary of a polytope P. We say that the* sign *of p is*

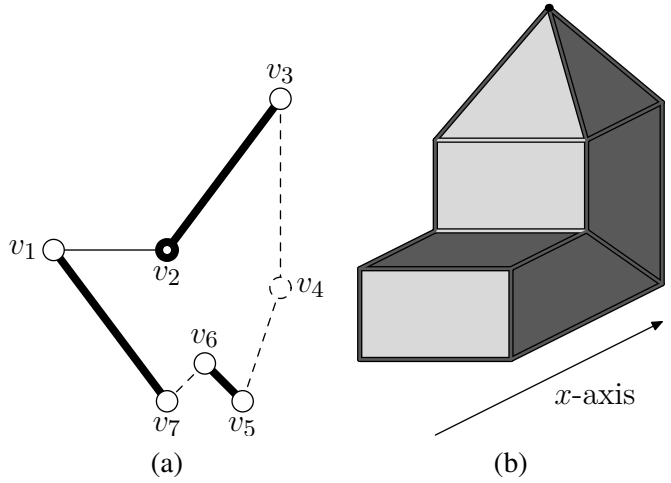


Figure 2: (a) A polygon with three positive (thick) edges, (v_1, v_7), (v_2, v_3), and (v_5, v_6), three negative (dashed) edges, (v_3, v_4), (v_4, v_5), and (v_6, v_7), and one neutral (thin) edge, (v_1, v_2). Also note that the end-points v_1, v_3, v_5, v_6, and v_7 are neutral (thin), v_2 is positive (thick) and v_4 is negative (dashed). (b) A polyhedron for which only positive (bright) and neutral (dark) faces are visible. Most of the edges are neutral since the interior of the polyhedron is neither to the left nor the right of the edges. Two edges are positive since the interior is only to the right of them.

- *positive if* $\exists \varepsilon > 0 : \forall \delta \in (0, \varepsilon) : p + \delta \cdot e_1 \in \text{int}(P)$ *and* $p - \delta \cdot e_1 \notin \text{int}(P)$,
- *negative if* $\exists \varepsilon > 0 : \forall \delta \in (0, \varepsilon) : p - \delta \cdot e_1 \in \text{int}(P)$ *and* $p + \delta \cdot e_1 \notin \text{int}(P)$,
- *and neutral if it is neither positive nor negative.*

In other words, a point is positive if the interior of the polytope is only on the right side of the point, it is negative if the interior of the polytope is only on the left side of the point, and it is neutral if the interior of the polytope is on both the left and the right side of the point or on neither the left nor the right side.

Clearly, each point on the boundary is covered by one and only one of the above cases. Furthermore, all points in the interior of a given face have the same sign. Therefore, any set of facets F can be partitioned into three sets F^+, F^-, and F^0 consisting of respectively positive, negative, and neutral facets from F. Examples in two and three dimensions are given in Figure 2.

In order to handle some special cases in the proofs, we need to be very specific as to which facet a given boundary point belongs. Every positive or negative point p on the boundary is *assigned* to exactly one facet as follows. If p belongs to the interior of a facet f then it cannot belong to the interior of any other facet and thus it is simply assigned to f. If p does not belong to the interior of a facet then it must be on the boundary of two or more facets. If p is positive, then it follows easily that this set of facets contains at least one positive facet to which it can be assigned. Analogously, if p is negative it is assigned to a negative facet. Such an assignment of the boundary will be referred to as a *balanced assignment*. Neutral points are not assigned to any facets. Given a (positive or negative) facet f, we write $p \in f$ if and only if p is a point assigned to f. Note that since all points in the interior of a face have the same sign, it follows that the assignment of all boundary points of the polytope can be done in bounded time; one only needs to determine the sign of one interior point of a face (of dimension 1 or more) to assign the whole interior of the face to a facet.

It follows from the definition of balanced assignment that a point moving in the direction of e_1, that passes through an assigned point of a facet of P, either moves from the exterior of P to the interior of P or vice versa. To determine when a point is inside a polytope we need the following definition.

Definition 10 (Facet Count Functions)**.** *Given a set of facets F we define the* facet count functions *for all points $p \in \mathbb{R}^d$ as follows:*

$$\begin{aligned}
\overleftarrow{C}_{F^+}(p) &= |\{f \in F^+ \mid \exists t > 0 : p - te_1 \in f\}|, \\
\overleftarrow{C}_{F^-}(p) &= |\{f \in F^- \mid \exists t \geq 0 : p - te_1 \in f\}|, \\
\overrightarrow{C}_{F^+}(p) &= |\{f \in F^+ \mid \exists t \geq 0 : p + te_1 \in f\}|, \\
\overrightarrow{C}_{F^-}(p) &= |\{f \in F^- \mid \exists t > 0 : p + te_1 \in f\}|.
\end{aligned}$$

The facet count functions $\overleftarrow{C}_{F^+}(p)$ and $\overleftarrow{C}_{F^-}(p)$ represent the number of times the ray from p with directional vector $-e_1$ intersects a facet from F^+ and F^-, respectively. Equivalently, $\overrightarrow{C}_{F^+}(p)$ and $\overrightarrow{C}_{F^-}(p)$ represent the number of times the ray from p with directional vector e_1 intersects a facet from F^+ and F^-, respectively.

The following lemma states some other important properties of polytopes and their positive/negative facets based on the facet count functions above.

Lemma 2. *Let P be a polytope with facet set F. Given a point p and interval $I \subseteq \mathbb{R}$, we say that the line segment $l_p(t) = p + t \cdot e_1$, $t \in I$,* intersects *a facet $f \in F$ if there exist $t_0 \in I$ such that $l_p(t_0) \in f$.*

Given a balanced assignment of the boundary points of P, then all of the following statements hold.

1. *If $I = (-\infty, \infty)$ then, as t increases from $-\infty$, the facets intersected by $l_p(t)$ alternate between positive and negative.*

2. *If $p \notin \text{int}(P)$ then $\overrightarrow{C}_{F^+}(p) = \overrightarrow{C}_{F^-}(p)$, i.e, the ray from p in direction e_1 intersects an equal number of positive and negative facets. Similarly, $\overleftarrow{C}_{F^+}(p) = \overleftarrow{C}_{F^-}(p)$.*

3. *If $p \in \text{int}(P)$ then $\overrightarrow{C}_{F^-}(p) - \overrightarrow{C}_{F^+}(p) = 1$, i.e, the number of positive facets intersected by the ray from p in direction e_1 is one less than the number of negative facets. Similarly, $\overleftarrow{C}_{F^+}(p) - \overleftarrow{C}_{F^-}(p) = 1$.*

The proof is straightforward, and is omitted.

As a corollary, the facet count functions provide an easy way of determining whether or not a given point is in the interior of P.

Corollary 1. *Let P be a polytope with facet set F. Given a balanced assignment of the boundary points, then for every point $p \in \mathbb{R}^d$ we have that p lies in the interior of P if and only if $\overrightarrow{C}_{F^-}(p) - \overrightarrow{C}_{F^+}(p) = 1$. Similarly, p lies in the interior of P if and only if $\overleftarrow{C}_{F^+}(p) - \overleftarrow{C}_{F^-}(p) = 1$.*

Proof. Follows directly from Lemma 2. □

The following definitions relate only to facets. Their purpose is to introduce a precise definition of the overlap between two polytopes in terms of their facets.

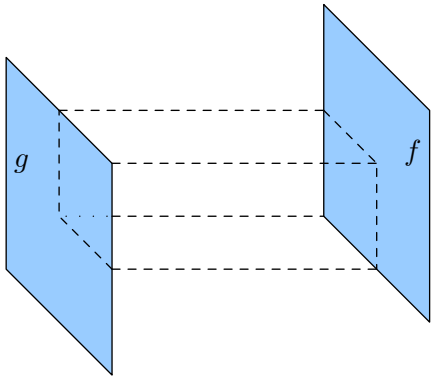


Figure 3: A simple example of the inter-facet region $R(f,g)$ of two vertical faces f and g in $\mathbb{R}^3$. The dashed lines delimit the region. Note that the inter-facet region $R(g,f)$ is empty.

Definition 11 (Containment Function, Inter-Facet Region)**.** *Given two facets f and g and a point $p' \in \mathbb{R}^d$ define the* containment function

$$\mathbf{C}(f,g,p') = \begin{cases} 1 & \text{if } \exists t_1,t_2 \in \mathbb{R} \,:\, t_2 < 0 < t_1,\ p'+t_2e_1 \in g, \text{ and } p'+t_1e_1 \in f \\ 0 & \text{otherwise.} \end{cases}$$

Also define the inter-facet region $R(f,g)$ *as the set of points which are both to the right of g and to the left of f; that is, $R(f,g) = \{p \in \mathbb{R}^d \mid \mathbf{C}(f,g,p) = 1\}$.*

Given two facet sets F and G, we generalize the containment function by summing over all pairs of facets (one from each set):

$$\mathbf{C}(F,G,p) = \sum_{f\in F}\sum_{g\in G} \mathbf{C}(f,g,p).$$

If f and g do not intersect and f lies to the right of g then in three dimensions the inter-facet region $R(f,g)$ is a *tube*, with the projection (in direction e_1) of f onto g and the projection of g onto f as ends. A simple example is given in Figure 3.

We now state a theorem which uses the containment function to determine whether or not a given point lies in the intersection of two polytopes.

Theorem 2. *Let P and Q be polytopes with facet sets F and G, respectively. Then for any point $p \in \mathbb{R}^d$ the following holds:*

$$\begin{aligned} p \in \operatorname{int}(P \cap Q) &\quad\Leftrightarrow\quad w(p) = 1 \\ p \notin \operatorname{int}(P \cap Q) &\quad\Leftrightarrow\quad w(p) = 0, \end{aligned}$$

where

$$w(p) = \mathbf{C}(F^+,G^-,p) + \mathbf{C}(F^-,G^+,p) - \mathbf{C}(F^+,G^+,p) - \mathbf{C}(F^-,G^-,p) \tag{1}$$

Proof. $\mathbf{C}(F^+,G^-,p)$ is equal to $\overrightarrow{C}_{F^+}(p) \cdot \overleftarrow{C}_{G^-}(p)$, since it is equal to the number of facets from F^+ which are to the right of p, times the number of facets from G^- which are to the left of p. Similar observations allows us to deduce that $\mathbf{C}(F^-,G^+,p) = \overrightarrow{C}_{F^-}(p) \cdot \overleftarrow{C}_{G^+}(p)$, $\mathbf{C}(F^+,G^+,p) = \overrightarrow{C}_{F^+}(p) \cdot \overleftarrow{C}_{G^+}(p)$, and $\mathbf{C}(F^-,G^-,p) = \overrightarrow{C}_{F^-}(p) \cdot \overleftarrow{C}_{G^-}(p)$.

If we assume $p \in \text{int}(P \cap Q)$ then from Corollary 1, we know that $\overrightarrow{C}_{F^-}(p) - \overrightarrow{C}_{F^+}(p) = 1$ and $\overleftarrow{C}_{G^+}(p) - \overleftarrow{C}_{G^-}(p) = 1$ and we get:

$$\begin{aligned} w(p) &= \overrightarrow{C}_{F^+}(p) \cdot \overleftarrow{C}_{G^-}(p) + \overrightarrow{C}_{F^-}(p) \cdot \overleftarrow{C}_{G^+}(p) - \overrightarrow{C}_{F^+}(p) \cdot \overleftarrow{C}_{G^+}(p) - \overrightarrow{C}_{F^-}(p) \cdot \overleftarrow{C}_{G^-}(p) \\ &= -\overrightarrow{C}_{F^+}(p) \cdot (\overleftarrow{C}_{G^+}(p) - \overleftarrow{C}_{G^-}(p)) + \overrightarrow{C}_{F^-}(p) \cdot (\overleftarrow{C}_{G^+}(p) - \overleftarrow{C}_{G^-}(p)) \\ &= \overrightarrow{C}_{F^-}(p) - \overrightarrow{C}_{F^+}(p) = 1. \end{aligned}$$

When $p \notin \text{int}(P \cap Q)$ the three cases where $p \notin P$ and/or $p \notin Q$ can be evaluated in similar fashion. □

In order to solve 1D*d*DTP we need to define a measure of the overlap between two polytopes.

Definition 12 (Overlap Measures)**.** *An* overlap measure *is a real-valued function μ such that, for any bounded region R_0, $\mu(R_0) = 0$ if $\text{int}(R_0) = \emptyset$ and $\mu(R_0) > 0$ otherwise.*

Preferably, an overlap measure should be computationally efficient and give a reasonable estimate of the degree of overlap of two polytopes. A general discussion of overlap measures in the context of translational packing algorithms can be found in Nielsen [25].

For the remainder of this paper we restrict our attention to the standard Euclidean volume measure V^d. Given a bounded region of space R we write $V^d(R)$ for its volume:

$$V^d(R) = \int_R dV^d.$$

In particular $V^d(R(f,g))$ is the volume of the inter-facet region of facets f and g. For convenience, we let $V^d(f,g) = V^d(R(f,g))$ and for sets of facets we use

$$V^d(F,G) = \sum_{f \in F} \sum_{g \in G} V^d(f,g).$$

The following theorem states that V^d is an overlap measure with a simple decomposition into volumes of inter-facet regions.

Theorem 3. *Let R_0 be a bounded region in $\mathbb{R}^d$, and let P and Q be polytopes in $\mathbb{R}^d$, with facet sets F and G respectively, such that $R_0 = P \cap Q$. Then V^d is an overlap measure, and it satisfies the following relation:*

$$V^d(R_0) = V^d(F^+,G^-) + V^d(F^-,G^+) - V^d(F^+,G^+) - V^d(F^-,G^-).$$

Proof. The theorem essentially follows from Theorem 2 and the proof given for the Intersection Area Theorem in Egeblad et al. [A], with area integrals replaced by d-dimensional volume integrals. □

Note that a balanced assignment is not required in order to get the correct overlap value using the facet decomposition of Theorem 3. This is due to the fact that the d-dimensional volume of all points in $P \cap Q$ which require the balanced assignment of boundary points to get the correct value in Theorem 2 is 0 and therefore will have no impact on the resulting volume.

In order to simplify notation in the following sections, for a d-dimensional region R we write $V(R)$ for $V^d(R)$; and for given facets f and g we write $V(f,g)$ for $V^d(f,g)$.

4.2 Minimum area translations

In the following we describe an efficient algorithm for solving 1D*d*DTP with respect to volume measure using Theorem 3. We continue to assume that the translation direction is e_1, i.e., parallel to the x-axis, and we will use terms such as *left*, *right* and *horizontal* as natural references to this direction.

4.2.1 Volume Calculations

Here we describe a method for computing the volume of overlap between two polytopes by expressing it in terms of the volumes of a collection of inter-facet regions, and then using the decomposition of volume measure given in Theorem 3.

First we introduce some basic notation. Given a point $p \in \mathbb{R}^d$ and a translation value $t \in \mathbb{R}$, we use $p(t)$ to denote the point p translated by t units to the right, i.e., $p(t) = p + te_1$. Similarly, given a facet $f \in F$, we use $f(t)$ to denote the facet translated t units to the right, i.e., $f(t) = \{p(t) \in \mathbb{R}^d | p \in f\}$. Finally, given a polytope P with facet set F, we let $P(t)$ denote P translated t units to the right, i.e., $P(t)$ has facet set $\{f(t) | f \in F\}$.

Now, consider two polytopes P and Q with facet sets F and G, respectively. For any two facets $f \in F$ and $g \in G$, we will show how to express the volume function $V(f(t), g)$ as a piecewise polynomial function in t with degree d. Combined with Theorem 3, this will allow us to express the full overlap of $P(t)$ and Q as a function in t by iteratively adding and subtracting volume functions of inter-facet regions. In order to express volumes in higher dimensions, we use the concept of a *simplex*. A simplex is the d-dimensional analogue of a triangle and it can be defined as the convex hull of a set of $d+1$ affinely independent points.

Before describing how to calculate $V(f(t), g)$, we first make a few general observations on $R(f(t), g)$; recall that $V(f(t), g) = V(R(f(t), g))$.

Definition 13 (Hyperplanes and Projections)**.** *Let f and g be facets of polytopes in $\mathbb{R}^d$. We denote by $\overline{f}$ the unique hyperplane of $\mathbb{R}^d$ containing f. Also, we define the* projection *of g onto f as*

$$\text{proj}_f(g) = \{p \in f | \exists t \in \mathbb{R} : \ p(t) \in g\} \tag{2}$$

In other words, $\text{proj}_f(g)$ is the horizontal projection of the points of g onto f.

Using these definitions, there are a number of different ways to express the inter-facet region $R(f(t), g)$. Let $f' = \text{proj}_g(f)$ and $g' = \text{proj}_f(g)$; then it is easy to see that $R(f(t), g) = R(f'(t), g') = R(f'(t), \overline{g}) = R(\overline{f(t)}, g')$ for any value of t. Clearly, the corresponding volumes are also all identical. To compute f' and g', it is useful to first project f and g onto a hyperplane $\overline{h}$ orthogonal to the translation direction. For a horizontal translation this can be done by simply setting the first coordinate to 0 (which, in 3D, corresponds to a projection onto the yz-plane). We denote the $d-1$ dimensional intersection of these two projections by $h = \text{proj}_{\overline{h}}(f) \cap \text{proj}_{\overline{h}}(g)$. The region h can then be projected back onto f and g (or their hyperplanes) to obtain f' and g'. This also means that the projections of f' and g' onto $\overline{h}$ are identical. In the case of three dimensions, when f' is to the right of g', h is a polygonal cross-section of the tube $R(f, g)$ (perpendicular to e_1) and f' and g' are the end-faces of this tube. See Figures 4a and 4b for an example.

Finding the intersection of the projections of f and g is relatively straightforward. Since we require the facets to be convex, the projections are also convex and can therefore be represented using half-spaces. The intersection can then easily be represented by the intersection of all of the half-spaces from the two sets and the main problem is then to find the corresponding vertex representation.

If the intersection h is empty or has dimension smaller than $d-1$ then the volume $V(f(t), g)$ is 0 for any t value. So, assume we have a $(d-1)$-dimensional intersection h, i.e., a non-degenerate intersection. Let $p_1, \ldots, p_n$ be the vertices of g'. For each point p_i, there exists a translation value t_i such that $p_i(-t_i) \in f$. Each point $p_i(-t_i)$ is a vertex of f', and each t_i value represents the horizontal distance of p_i from the hyperplane $\overline{f}$. Assume that the points p_i are sorted such that $t_1 \leq \cdots \leq t_n$. We refer to these points as *breakpoints* and the t_i values as their corresponding distances.

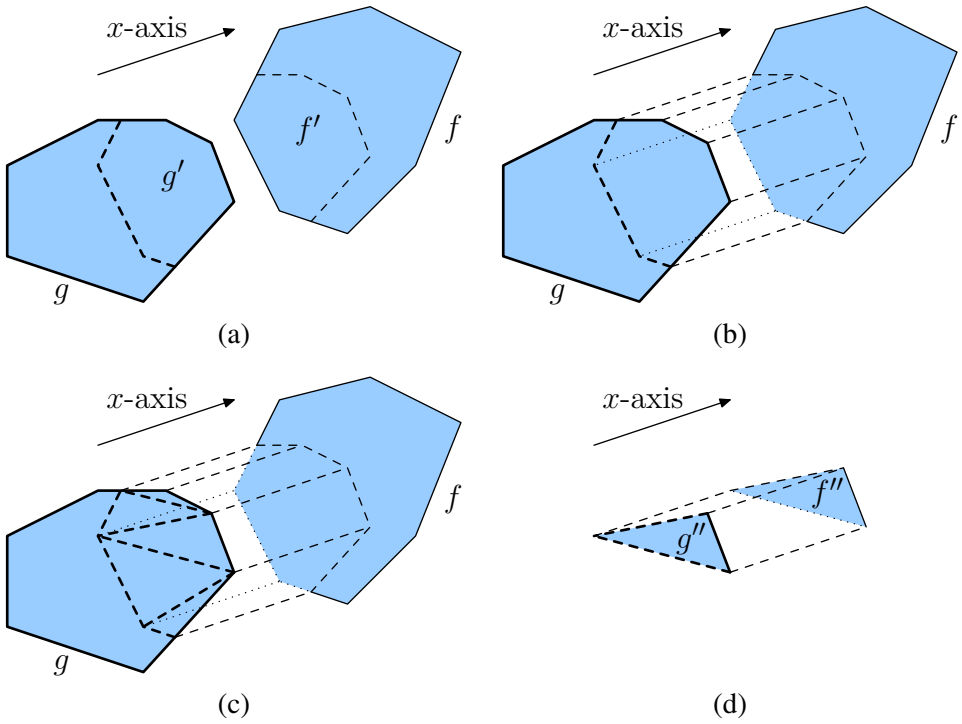


Figure 4: An example of the inter-facet region $R(f,g)$ between two faces, f and g, in three dimensions, where g is in front of f in the x-axis direction. In general the two facets are not necessarily in parallel planes and they can also intersect. (a) The dashed lines indicate the boundary of the projections of f on g and g on f. The projections are denoted f' and g'. (b) The region $R(f,g)$ which is identical to $R(f',g')$. (c) To simplify matters, the projections can be triangulated. (d) One of the resulting regions with triangular faces.

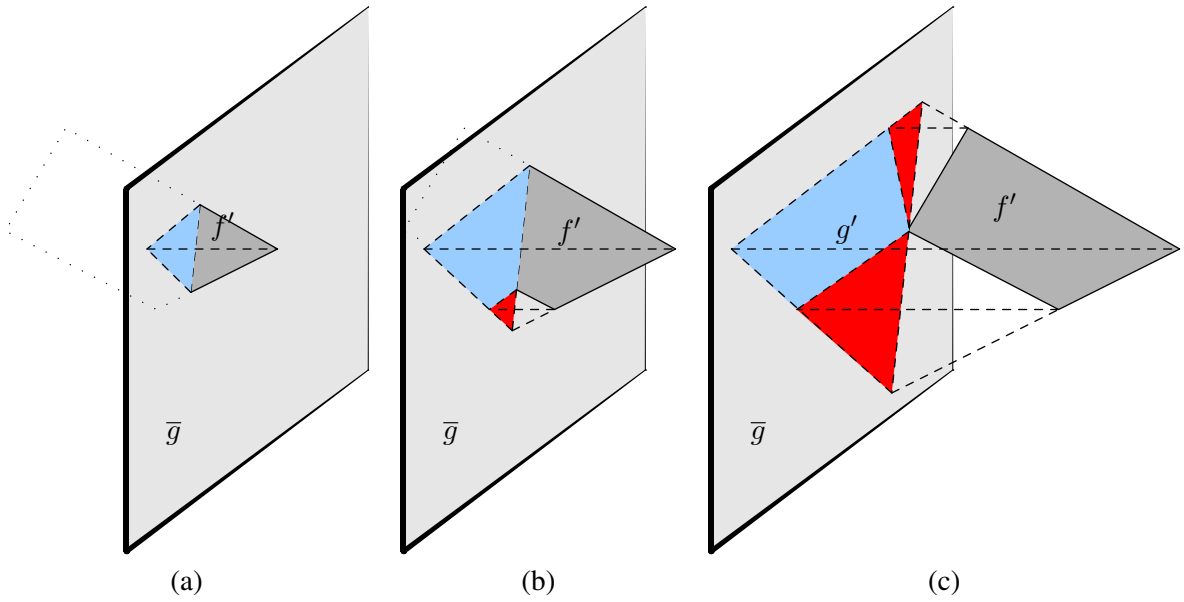


Figure 5: An illustration of the growing tetrahedron volumes needed when a face f' passes through a face g, or equivalently its plane $\overline{g}$, in order to calculate the volume of $R(f',g')$. (a) The initial growing tetrahedron after the first breakpoint. (b) After the second breakpoint, at growing tetrahedral volume based on the red area needs to be subtracted. (c) After the third breakpoint, a second growing tetrahedral needs to be subtracted.

We now consider how to compute $V(f(t),g)$. If $t \leq t_1$ then the region $R(f(t),g)$ is clearly empty since $f'(t)$ is entirely to the left of g'. It follows in this case that $V(f(t),g) = 0$. A bit less trivially, if $t \geq t_n$ then $V(f(t),g) = V(f(t_n),g) + (t-t_n) \cdot V^{(d-1)}(h)$ which is a linear function in t. In this case $f'(t)$ is to the right of g' in its entirety. Note that $V^{(d-1)}(h)$ is the volume of h in $(d-1)$-dimensional space. In $\mathbb{R}^3$, $V^2(h)$ is the area of the polygonal cross-section of the tube between f' and g'. The volume $V^{(d-1)}(h)$ can be computed by partitioning h into simplices. Hence the only remaining difficulty lies in determining $V(f(t),g)$ for $t \in (t_1,t_n]$. For any value of t in this interval, $f'(t)$ and g' intersect in at least one point.

Figure 5 is an illustration of what happens when a face in 3D is translated through another face. This is a useful reference when reading the following. First note that the illustration emphasizes that we can also view this as the face f' passing through the plane $\overline{g}$.

An easy special case, for computing $V(f(t),g)$, occurs when $t_1 = t_n$. This corresponds to the two facets being parallel and thus $V(f(t_n),g) = 0$.

Now, assume all the t_i are distinct. The case where two or more t_i are equal is discussed in Section 4.3. Each facet is required to be simple by definition and thus each vertex p_i of g' has exactly $d-1$ neighboring points, i.e., vertices of g' connected to p_i by edges. Denote these points $p_i^1,\ldots,p_i^{d-1}$ and denote the corresponding breakpoint distances $t_i^1,\ldots,t_i^{d-1}$.

Consider the value of $V(f(t),g)$ in the first interval $(t_1,t_2]$. To simplify matters, we change this to the equivalent problem of determining the function $V_0(f(t),g) = V(f(t+t_1),g)$ for $t \in (0,t_2-t_1]$. $V_0(f(t),g)$ can be described as the volume of a growing simplex in t with vertex set $\{tv_j | j = 0,\ldots,d\}$ where $v_0 = (0,\ldots,0)$, $v_d = (1,0,\ldots,0)$ and $v_j = (p_1^j - p_1)/(t_1^j - t_1)$ for $1 \leq j \leq d-1$. The volume of this simplex is:

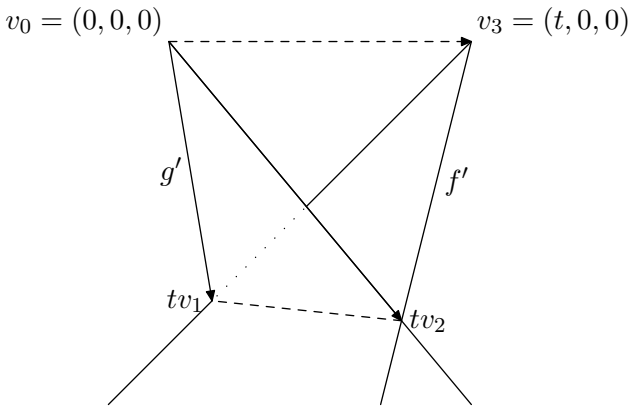


Figure 6: In 3D, it is necessary to calculate the volume function for a growing tetrahedron based on two points, v_1 and v_2, and an overlap distance t. The coordinates of the points change linearly with respect to a change of the distance t. In d dimensions $d-1$ neighboring points are used.

$$V_0(f(t),g) = \frac{1}{d!}|\det([tv_1,...,tv_d])|.$$

This is illustrated in Figure 6.

Since $v_d = (1,0,\ldots,0)$, we can simplify the above expression to

$$V_0(f(t),g) = \frac{1}{d!}|\det([v_1,v_2,...,v_d])t^d| = \frac{1}{d!}|\det([v'_1,v'_2,...,v'_{d-1}])t^d|,$$

where v'_i is v_i without the first coordinate. This results in very simple expressions in low dimensions:

$$\begin{array}{lrcl} \text{2D}: & \frac{1}{2}|\det(v'_2)t^2| & = & \frac{1}{2}|v_2^y t^2| \\ \text{3D}: & \frac{1}{6}|\det([v'_2v'_3])t^3| & = & \frac{1}{6}|(v_2^y v_3^z - v_2^z v_3^y)t^3| \end{array}$$

In general $V_0(f(t),g)$ is a degree d polynomial in t.
To calculate the original volume $V(f(t),g)$ we offset the function by setting $V(f(t),g) = V_0(f(t-t_1),g)$ for $t \in (t_1,t_2]$.

The above accounts for the interval between the two first breakpoints. To handle the remaining breakpoints we utilize a result of Lawrence [21] concerning the calculation of the volume of a convex polytope. Lawrence shows that the volume can be calculated as the sum of volumes of a set of simplices. Each simplex is based on the neighboring edges of each vertex of the polytope.

Given $t \in (t_1,t_n]$, we are interested in the polytope $R(f(t),g)$. It can be constructed by taking the part of f' which is to the right of the hyperplane $\overline{g}$, then projecting this back onto $\overline{g}$ and connecting the corresponding vertices horizontally. See Figure 5 for some examples. This is a convex polytope, and its volume can be calculated as follows.

Let $\Delta(p_1,t)$ denote the initial growing simplex described above, that is $V(\Delta(p_1,t)) = V_0(f(t_1-t),g)$. Similarly, let $\Delta(p_i,t), i \in \{2,...,n-1\}$ denote simplices based on the other points p_i and their neighboring points (we do not need to include p_n). For each such simplex, the vertices are given by tv_j where $v_j = (p_i^j - p_i)/(t_i^j - t_i)$. Note that whenever $t_i^j < t_i$ the direction of the vector from p_i to p_i^j is reversed. By an argument of Lawrence [21], we then have

$$V(f(t),g) = V(\Delta(p_1,t)) - \sum_{i=2}^{k} V(\Delta(p_i,t)) \text{ where } k = \max_{1\leq i\leq n}\{i|t_i < t\}. \quad (3)$$

In other words, the volume can be calculated by taking the growing volume of the first simplex and then subtracting a simplex for each of the remaining breakpoints with a breakpoint distance less than t. The volume of each simplex can be described as a degree d polynomial in t similar to the description of V_0 of the previous paragraph. In Figure 5a, there is only the first growing simplex (tetrahedron). After that, in Figure 5b, another growing simplex needs to be subtracted from the first, and finally, in Figure 5c, a third growing simplex needs to be subtracted. Between each pair of breakpoints, the total volume between g' and $f'(t)$ in the horizontal direction can therefore be described as a sum of polynomials in t of degree d which itself is a polynomial of degree d.

4.2.2 The Main Algorithm

An algorithm for determining a minimum overlap translation in d dimensions can now be established. Pseudo-code is given in Algorithm 3. Given polytopes P and Q and a polytope container C, we begin by determining all breakpoints between facets from P and facets from polytopes Q and C and the coefficients of their d-dimensional volume polynomials. Signs of all volume polynomials calculated with regard to the container C should be negated. This corresponds to viewing the container as an infinite polytope with an internal cavity.

Algorithm 3: Determine minimum overlap translation along x-axis in d dimensions

Input: Polytopes P and Q and a container C;

foreach facet f from P **do**
  foreach facet g from $Q \cup C$ **do**
    Create all breakpoints for the facet pair (f,g).;
    Each breakpoint (f,g) has a distance $t_{f,g}$ and a volume polynomial $V_{f,g}(t)$;
    (negate the sign of $V_{f,g}(t)$ if $g \in C$).;

Let **B** be all breakpoints sorted in ascending order with respect to t.;
Let $\nu(t)$ and $\nu_C(t)$ be polynomials with maximum degree d and initial value $\nu(P)$.;

for $i = 1$ to $|\mathbf{B}| - 1$ **do**
  Let t_i be the distance value of breakpoint i.;
  Let f and g be the facets of breakpoint i.;
  Let $V_i(t)$ be the volume function of breakpoint i.;
  Modify $\nu(t)$ by adding the coefficients of $V_i(t)$.;
  if $g \in C$ **then**
    Modify $\nu_C(t)$ by adding the coefficients of $V_i(t)$.;
  if $\nu_C(t) = 0$ for $t \in [t_i, t_{i+1}]$ **then**
    Find the minimum value t_i' for which $\nu(t)$ is minimized in $[t_i, t_{i+1}]$.;
return t_i' with smallest $\nu(t_i')$

The breakpoints are sorted such that $t_1 \leq t_2 \leq \ldots \leq t_n$. The algorithm traverses the breakpoints in this order while maintaining a total volume function $\nu(t)$ which describes the volume of the total overlap between P and $Q \cup C$. The volume function $\nu(t)$ is a linear combination of the volumes of

simplices for each breakpoint encountered so far (based on Equation 3) and may be represented as a degree d polynomial in t.

Initially $\nu(t) = \nu(P)$ for $t \leq t_1$, corresponding to P being completely outside the container (overlapping C). As each breakpoint t_i is reached, the algorithm adds an appropriate volume polynomial to $\nu(t)$. Since $\nu(t)$ is a sum of polynomials of at most degree d, $\nu(t)$ can itself be represented as $d+1$ coefficients of a polynomial with degree at most d. When a breakpoint is encountered its volume polynomial is added to $\nu(t)$ by simply adding its coefficients to the coefficients of $\nu(t)$.

The subset of volume polynomials which comes from breakpoints related to the container may also be added to a volume polynomial $\nu_C(t)$. Whenever this function is 0 for a given t-value, it means that $P(t)$ is inside the container.

For each interval $(t_i, t_{i+1}]$ between succeeding breakpoints for which $\nu_C(t) = 0$, $\nu(t)$ can be analyzed to determine local minima. Among all these local minima, we select the smallest distance value t_{min} for which $\nu(t_{min})$ is a global minimum value. This minimum corresponds to the leftmost x-translation where the overlap between P and $Q \cup C$ is as small as possible. Therefore t_{min} is a solution to 1DdDTP.

Analyzing each interval amounts to determining the minimum value of a polynomial of degree d. This is done by finding the roots of the derivative of the polynomial and checking interval end-points.

While finding the exact minimum for $d \leq 5$ is easy, it is problematic for higher dimensions, due to the Abel-Ruffini Theorem, since one must find the roots of the derivative which itself is polynomial of degree 5 or higher. However, it is possible to find the minimum to any desired degree of accuracy, e.g., using the Newton-Raphson method. Approximations are needed in any case when using floating point arithmetics.

The following lemma is a simple but important observation.

Lemma 3. *Given two polytopes P and Q, assume that the 1-dimensional translation problem in n dimensions has a solution (a translation distance t') where P does not overlap Q then there also exists a breakpoint distance t_b for which P does not overlap Q.*

Proof. Assume that t' is not a breakpoint distance. First assume t' is in an interval (t_b^1, t_b^2) of breakpoint distances. Assume that it is the smallest such interval. Since the overlap value, $\nu(t')$, is 0 and t' is not a breakpoint distance then, specifically, no facets from P and Q can intersect for $t = t'$. Since there are no other breakpoint distances in this interval, and facets can only begin to intersect at a breakpoint distance, no facets can intersect in the entire interval (t_b^1, t_b^2). From the discussion in Section 4.2.1, $\nu(t)$ must be a sum of constants or linear functions for $t \in (t_b^1, t_b^2)$ since no facets intersect in this interval. $\nu(t)$ cannot be linear since t' is not an interval end-point and this would imply a negative overlap at one of the interval end-points which is impossible. Therefore $\nu(t) = 0$ for $t \in (t_b^1, t_b^2)$. By continuity $\nu(t_b^1) = 0$ and $\nu(t_b^2) = 0$ and we may choose either of these breakpoints as t_b.

Now assume t' is within the half-open infinite interval either before the first breakpoint or after the last breakpoint. Again, since $\nu(t)$ is linear on that entire interval and $\nu(t)$ cannot be negative, one can select the breakpoint of the infinite interval as t_b. □

If a non-overlapping position of P exists for a given translation direction, then P is non-overlapping at one of the breakpoint distances. Our solution method for dDDPP, as described in Section 5, repeatedly solves 1DdDTP problems using Algorithm 3. Since our aim is to find a non-overlapping position for each polytope we may actually limit our analysis in each translation to testing the interval end-points. This way one may avoid the computationally inefficient task of finding roots even though one does not find the true minimum for 1DdDTP (though it might be at the expense of increasing the number of translations required to find a solution).

To analyze the asymptotic running time of Algorithm 3, we assume that either one does not find minima between breakpoints or one considers this a constant time operation (which is true for 5 dimensions or less). For a fixed dimension d, the time needed for finding the intersection of $(d-1)$-simplices can also be considered a constant, and the same is true for the calculation of the determinants used in the volume functions. Given two polytopes with complexity $O(n)$ and $O(m)$, the number of breakpoints generated is at most a constant times nm. Computing the volume function for each pair of breakpoints takes constant time and thus the running time is dominated by the sorting of breakpoints revealing a running time of $O(nm\log(nm))$. In most cases, the number of breakpoints is likely to be much smaller than $O(nm)$ since the worst case scenario requires very unusual non-convex polytopes — the vertical extents must overlap for all pairs of edges. If the complexity of the facets is bounded (e.g. triangles for $d = 3$), then the number of breakpoints generated for two polytopes with n and m *facets*, respectively, is at most $O(nm)$, and the asymptotic running time is $O(nm\log(nm))$.

In some settings it may be desirable to allow for overlap with the region outside the container. This is easily accommodated by ignoring the condition that $\nu_C(t) = 0$ in Algorithm 3.

4.3 Special cases in three dimensions

In the previous subsection, it was assumed that either all breakpoints had unique distance values or that all breakpoints had the same distance value. Here we describe how to handle a subset of breakpoints with the same distance value for the case where $d = 3$. In particular, we focus on the special case of the two first breakpoint distances being equal (and different than the subsequent ones).

Given two convex faces f and g, we know that h, the intersection of their 2D projections onto the yz-plane, is also a convex polygon. As before, let f' and g' denote the projections of h onto the given faces. When $\overline{f}$ and $\overline{g}$ are not parallel, then for any given translation of f', the intersection between f' and g' contains at most two corner points. Furthermore, Equation 3 still applies if these two corner points are not neighbors; and they can only be neighbors if they are the two first breakpoints or the two last breakpoints. The latter case is not a problem since at that point, the volume function can be changed to a linear expression based on the area of h.

The important special case is when the two first breakpoint distances are equal. This problem varies depending on the shape of h, but in each case the solution is almost the same. Figure 7a illustrates the standard case with only a single initial breakpoint while Figures 7b-d illustrate three possible variants of having two initial breakpoints. They differ with respect to the direction of the neighboring edges, but in all three cases it is possible to introduce a third point p' which emulates the standard case in Figure 7a. Essentially, the calculations need to be based on a tetrahedron with fixed size, a linearly increasing volume and the usual cubic volume function of a growing tetrahedron. A more detailed illustration and description of the situation in Figure 7b is given in Figure 8, where v'_2 corresponds to p_1, v'_3 to p_2, and v_0 to p_0.

Note that quite often polyhedrons have triangulated surfaces. Given triangular faces, the special case in Figure 7d cannot occur. The special case in Figure 7b can also be avoided if the intersection polygon is triangulated and each triangle is handled separately (see Figure 4d).

It is still an open question how to handle identical breakpoint distances in higher dimensions. Perturbation techniques could be applied, but it would be more gratifying if the above approach could be generalized to higher dimensions.

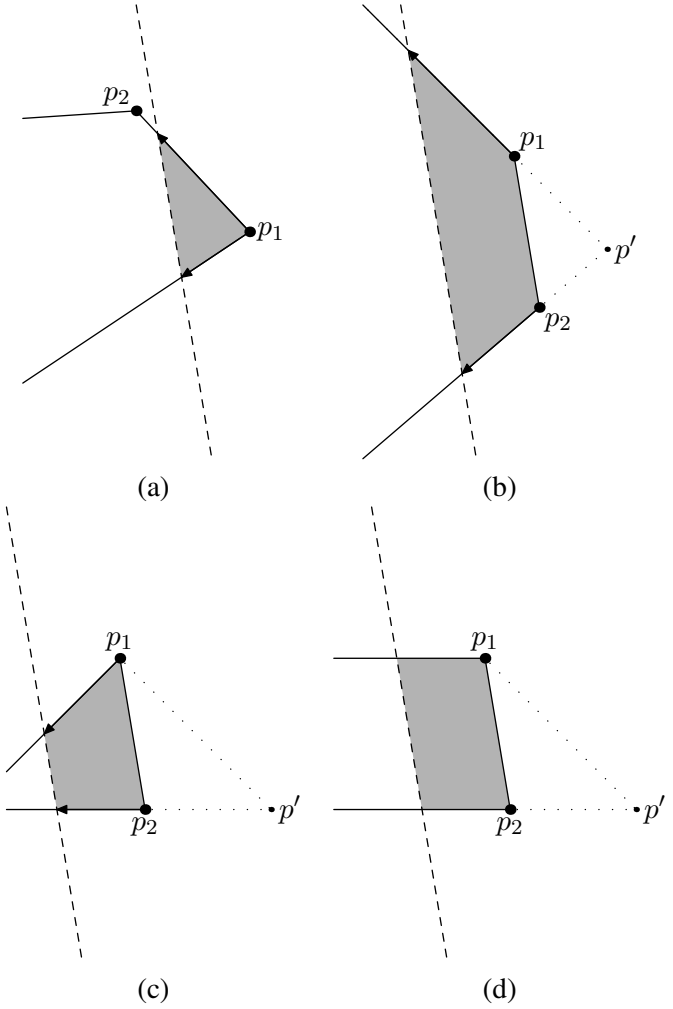


Figure 7: Various special cases can occur when the two first breakpoint distances are equal. Here it is illustrated in 2D using a dashed line to illustrate the intersection line of the two faces during the translation. (a) The standard case in which the first breakpoint distance is unique. (b) The two first breakpoint distances are equal (points p_1 and p_2) which also means that the dashed line is parallel to the line between p_1 and p_2. The sum of the angles at p_1 and p_2 is greater than 180°. This can be handled by introducing a third point p' at the intersection of the lines through the edges from p_1 and p_2. This point is going to have a smaller breakpoint distance and it almost reduces the problem to be identical with the first case. More details can be seen in Figure 8. (c) The sum of the angles at p_1 and p_2 is less than 180°, but we can still find a natural candidate for the additional point p' based on the edges from p_1 and p_2. (d) The sum of angles is exactly 180°. In this case p' is just chosen at an appropriate distance from p_2, e.g., such that the angle at p' is 45°. This case does not occur if the input polyhedra have triangulated faces.

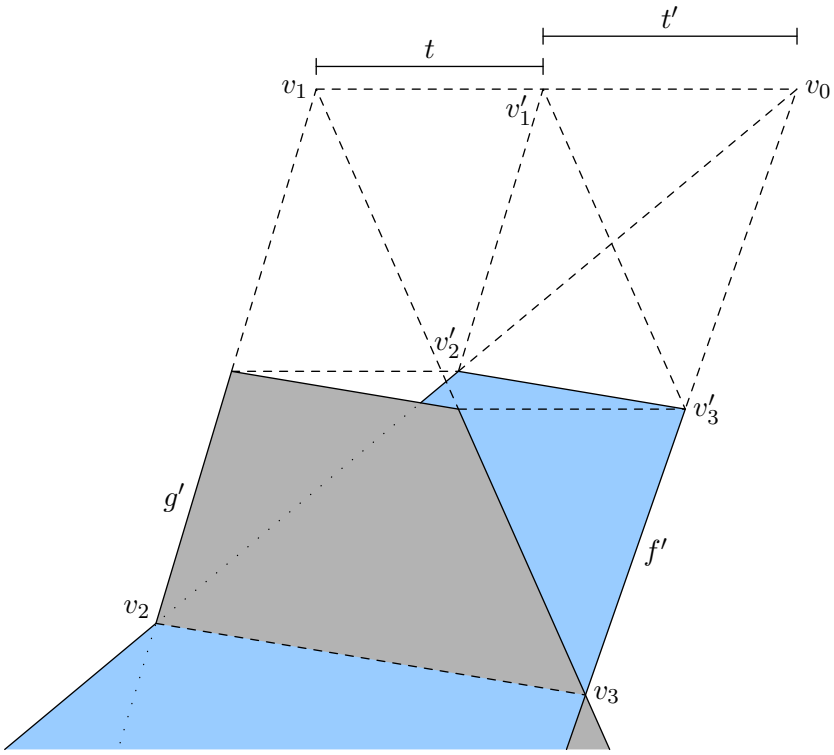


Figure 8: To calculate the volume between the faces f' and g', one can base the calculations on the extended faces illustrated with dashed lines. The volume is a growing tetrahedron (v_0, v_1, v_2, v_3) subtracted by a constant volume (the tetrahedron (v_0, v_1', v_2', v_3')) and subtracted by a linearly growing volume based on the triangle (v_1', v_2', v_3') and the translation distance t.

5 Solution method for dDSPP

In this section we describe our solution method for the d-dimensional strip packing problem (dDSPP), i.e., the problem of packing a set $\mathcal{S}$ of n given polytopes inside a d-dimensional rectangular parallelepiped C with minimal height. In short, we do this by solving the decision problem dDDPP for repeatedly smaller heights using a local search and a meta-heuristic technique. This stops when a fixed time limit is reached. The height of the last solved dDDPP is reported as a solution to the dDSPP. Each instance of dDSPP in turn is solved by repositioning polytopes to minimal overlapping positions using Algorithm 3. In the following, we first review the local search and the meta-heuristic technique used and described by Egeblad et al. [A]. We then describe how one can obtain an initial height (Section 5.2) for which a non-overlapping placement is known to exist.

5.1 Local search and guided local search

To solve dDDPP (for a container with given height) we apply a local search method in conjunction with *guided local search* (GLS); a meta-heuristic technique introduced by Voudouris and Tsang [29]. Given a set of polytopes $\mathcal{S} = \{Q_1, \ldots, Q_n\}$, the local search starts with a placement of the polytopes which may contain overlap. Overlap is then iteratively reduced by translating one polytope at a time in axis-aligned directions to a minimum overlap position. When overlap can no longer be reduced by translating a single polytope the local search stops. If overlap is still present in the placement then GLS is used to escape this constrained local minimum. Otherwise a non-overlapping placement has been found for dDDPP and the strip-height is decreased and all polyhedrons are moved inside the smaller container.

Minimum overlap translations are found using the axis-aligned translation algorithm described in the previous section. For each polytope each of the possible d axis-aligned translations are used and the direction which reveals the position with least overlap is chosen. Note that if $P = Q_i$ is the polytope undergoing translation then the polytope Q (in the translation algorithm in the previous section) is actually the union of all other polytopes, i.e., $Q = \cup_{j=1,\ j\neq i}^{n} Q_j$. The container C in the previous section is assumed to be a simple rectangular parallelepiped with some initial height h. (The solution method can, however, be adapted to other containers.)

Let $V_{i\cap j}(\mathbf{P})$ be the volume of the pairwise overlap of Q_i and Q_j in a placement $\mathbf{P}$. The local search minimizes the objective function

$$f(\mathbf{P}) = \sum_{1\leq i<j\leq n} V_{i\cap j}(\mathbf{P}). \tag{4}$$

In other words, $f(\mathbf{P})$ is the total sum of volumes of pairwise overlaps in placement $\mathbf{P}$. If $f(\mathbf{P}) = 0$ then $\mathbf{P}$ is a solution to the current instance of dDDPP.

The local search uses the axis-aligned translation algorithm from the previous section to iteratively decrease the amount of overlap in the current placement. A local minimum is reached if no polytope can be translated in an axis-aligned direction to a position with less overlap. To escape local minima the objective function is augmented using the principles of GLS to give

$$g(\mathbf{P}) = f(\mathbf{P}) + \lambda \sum_{1\leq i<j\leq n} \phi_{i,j} I_{i,j}(\mathbf{P}), \tag{5}$$

where λ is a penalty constant used to fine-tune the heuristic, $\phi_{i,j}$ is a penalty term associated with Q_i and Q_j (which is described in more detail below), and $I_{i,j}(\mathbf{P}) \in \{0,1\}$ is 1 if and only if the interiors of polytopes Q_i and Q_j overlap in placement $\mathbf{P}$. Due to the fact that the augmented terms are larger than

zero if and only if $\mathbf{P}$ contains overlap, the augmented objective function, $g(\mathbf{P})$, retains the property that a value of 0 is reached if and only if there is no overlap in the placement $\mathbf{P}$. Therefore, a placement $\mathbf{P}$ is a solution to the dDDPP if and only if $g(\mathbf{P}) = 0$.

Initially, $\phi_{i,j} = 0$ for $1 \leq i < j \leq n$. Whenever the search reaches a local minimum with $g(\mathbf{P}) > 0$, the value of $\phi_{i,j}$ is increased for the pair Q_i and Q_j with highest $\mu_{i,j}(\mathbf{P})$ where $\mu_{i,j}(\mathbf{P}) = \frac{V_{i\cap j}(\mathbf{P})}{1+\phi_{i,j}}$. We refer to this as *penalizing* the pair Q_i and Q_j. Intuitively, this change of the objective function $g(\mathbf{P})$ (if large enough), allows the local search to move Q_i and Q_j away from each other, even if it results in greater overlap. Ideally, this move causes the local search to reach a different part of the solution space. To avoid large discrepancy between the real and penalized solution space, the penalties are reset from time to time. To avoid searching the entire neighborhood in each iteration of GLS, we also apply *fast local search* [29].

The translation algorithm in the previous section needs to be able to handle the penalties introduced here. This subject was not fully covered by Egeblad et al. [A] although it is quite straightforward. First of all an augmented volume polynomial $\nu'(t)$ is defined by adding the penalties between the polytope $P = Q_i$ to be translated and all other polytopes. This is done by maintaining an array of volume functions $\nu_{Q_j}(t), Q_j \in \mathcal{S} \setminus Q_i$. Whenever the overlap between P and a given polytope Q_j changes from 0 or to 0, the penalty for the pair of polytopes P, Q_j is, respectively, added to or subtracted from the augmented volume function $\nu'(t)$. Note that this does not increase the asymptotic running time, since the volume polynomial of a breakpoint arising from a face of Q_j is only added to $\nu'(t)$ and $\nu_{Q_j}(t)$ and only $\nu_{Q_j}(t)$ needs to be checked for a change to or from 0.

With regard to the usefulness of the local search neighborhood in relation to the number of dimensions d, we note that a 1-dimensional translation becomes a less efficient move as d increases, since up to d axis-aligned translations may be required to move a polytope from one arbitrary point to another. However, it should also be noted that in general fewer polytopes would be involved in each translation. If the polytopes are placed compactly in a grid-like fashion with little overlap, then there are likely to be in the order of $\sqrt[d]{|(S)|}$ polytopes to be considered in each of the coordinate system axes directions.

5.2 Initial solution

The solution method described above can start with a parallelepiped of any height since the initial placement is allowed to contain overlaps. However, it makes more sense to set the initial height to one for which a solution is known to exist.

In any dimension, a naive initial height can be based on the sum of heights of all polytopes, but in the following, we describe a more ambitious strategy. In short, we use a greedy bounding box based algorithm in which the polytopes are placed one by one inside the container in an order of decreasing bounding box volume. This algorithm is based on residual volumes and is related to the approach used by Eley [14] for the container loading problem in three dimensions. Although the algorithm could be generalized to higher dimensions, we are only going to describe its three-dimensional variant.

The algorithm maintains a set of empty box-spaces. Each box-space s consists of the volume $[\underline{x}_s, \overline{x}_s] \times [\underline{y}_s, \overline{y}_s] \times [\underline{z}_s, \overline{z}_s]$. Initially, the entire container is the only empty space.

Whenever a new shape i with bounding box $B_i = [\underline{x}_i, \overline{x}_i] \times [\underline{y}_i, \overline{y}_i] \times [\underline{z}_i, \overline{z}_i]$ is to be placed inside the container, the list of empty spaces is searched. Let s' be the empty space with lexicographical least $\underline{z}'_s$, $\underline{y}'_s$, and $\underline{x}'_s$ (lower-left-back corner), which is large enough to contain B_i. Shape i is now positioned in s with offset $(x_i, y_i, z_i)^T$ such that B_i's lower-left-back corner is coincident with the lower-left-back corner of s; $(\underline{x}_i, \underline{y}_i, \underline{z}_i)^T + (x_i, y_i, z_i)^T = (\underline{x}'_s, \underline{y}'_s, \underline{z}'_s)^T$.

Next all residual spaces that overlap with the bounding box of the positioned shape i, $B'_i = [\underline{x}_i + x_i, \overline{x}_i + x_i] \times [\underline{y}_i + y_i, \overline{y}_i + y_i] \times [\underline{z}_i + z_i, \overline{z}_i + z_i]$, are split into six new box-spaces and removed from the list of empty box-spaces. For each overlapping space s, we generate the following six new box-spaces:

$$[\underline{x}_s, \underline{x}_i + x_i] \times [\underline{y}_s, \overline{y}_s] \times [\underline{z}_s, \overline{z}_s], \quad [\underline{x}_s, \overline{x}_s] \times [\underline{y}_s, \underline{y}_i + y_i] \times [\underline{z}_s, \overline{z}_s], \quad [\underline{x}_s, \overline{x}_s] \times [\underline{y}_s, \overline{y}_s] \times [\underline{z}_s, \underline{z}_i + z_i]$$
$$[\overline{x}_i + x_i, \overline{x}_s] \times [\underline{y}_s, \overline{y}_s] \times [\underline{z}_s, \overline{z}_s], \quad [\underline{x}_s, \overline{x}_s] \times [\overline{y}_i + y_i, \overline{y}_s] \times [\underline{z}_s, \overline{z}_s], \quad [\underline{x}_s, \overline{x}_s] \times [\underline{y}_s, \overline{y}_s] \times [\overline{z}_i + z_i, \overline{z}_s].$$

These represent the volumes left, below, behind, right, above, and in-front of B_i, respectively. If any of the intervals are empty then the new space is empty. Each of the new non-empty spaces are added to the list of empty spaces and may be used for the remaining bounding boxes. To reduce the number of empty spaces generated throughout this process, spaces which are contained within or are equal to other empty spaces are discarded whenever a new bounding box is placed.

The resulting placement is a non-overlapping placement and the maximum $\overline{z}$ value of any placed bounding box B'_i may be used as a basis for the initial strip-height. To diversify solutions to 3DSPP we place shapes randomly within a container with this strip-height. This is the only random element of the solution method.

6 Computational experiments

The solution method described in this paper was implemented for the three dimensional problem using the C++ programming language and the GNU C++ 4.0 compiler. We denote this implementation 3DNEST. Although similar in functionality, this implementation is not identical to the one used by Egeblad et al. [A]. In particular, the new implementation can handle convex faces without triangulating them and it can handle the strip packing problem. Another noteworthy feature of the new implementation is that it is possible to do almost all calculations with rational numbers — the only exception is the computation of minimum values between breakpoints in the translation algorithm since this requires solving quadratic equations. This is primarily convenient for debugging purposes, and is currently not very useful in practice since it is much slower than using standard floating point precision.

Due to the limited precision of floating point calculations, the correctness of all solutions found are verified using CGAL [17], i.e., it is verified that no polyhedron is involved in any significant overlap with other polyhedra or the container. In the experiments presented in this section, the largest total volume of overlap allowed in a solution corresponds to 0.01% of the total volume of all polyhedrons for the given problem.

All experiments were performed on a system with a 2.16 GHz Intel Core Duo processor with 2 MB of level 2 cache and 1 GB of RAM. Note that the implementation only uses one core of the processor.

6.1 Problem instances

The literature on the subject of three-dimensional packing contains only few useful problem instances with regard to a comparison of results. We have found two appropriate data sets for our experiments. The first one was introduced by Ikonen et al. [20] and the second one was introduced by Stoyan et al. [27]. The sets contain 8 and 7 polyhedra, respectively. Characteristics of these data sets are presented in Table 1. The Stoyan polyhedra are all convex and relatively simple with a maximum of 18 faces, while some of the Ikonen polyhedra are non-convex and feature up to 52 faces. Real world instances, e.g., from the rapid prototyping industry, could easily contain more than 100,000 faces,

but in most cases it would also be possible to simplify these polyhedra considerably without making substantial changes to the basic shape, e.g., Cohen et al. [6] has an example of a model of a phone handset which is reduced from 165,936 to 412 triangles without changing its basic shape.

					Ikonen		Stoyan		
Name	Faces	Volume	Bounding box	Type	1	2	1	2	3
Block1	12	4.00	$1.00 \times 2.00 \times 2.00$	Convex					
Part2	24	2.88	$1.43 \times 1.70 \times 2.50$	Non-convex	3	8			
Part3	28	0.30	$1.42 \times 0.62 \times 1.00$	Non-convex	2	2			
Part4	52	2.22	$1.63 \times 2.00 \times 2.00$	Non-convex	1	1			
Part5	20	0.16	$2.81 \times 0.56 \times 0.20$	Non-convex	2	2			
Part6	20	0.24	$0.45 \times 0.51 \times 2.50$	Non-convex	2	2			
Stick2	12	0.18	$2.00 \times 0.30 \times 0.30$	Convex					
Thin	48	1.25	$1.00 \times 3.00 \times 3.50$	Non-convex					
Convex1	14	176.00	$5.00 \times 6.00 \times 8.00$	Convex			1	1	2
Convex2	4	74.67	$11.00 \times 4.00 \times 14.00$	Convex			1	1	4
Convex3	10	120.00	$3.00 \times 4.00 \times 12.00$	Convex			1	1	6
Convex4	16	124.67	$3.00 \times 4.00 \times 16.00$	Convex			1	1	4
Convex5	18	133.33	$4.00 \times 8.00 \times 10.00$	Convex			1	3	4
Convex6	8	147.00	$6.00 \times 7.00 \times 7.00$	Convex			1	2	3
Convex7	16	192.50	$6.00 \times 10.00 \times 9.00$	Convex			1	3	2
				Number of polyhedra:	10	15	7	12	25

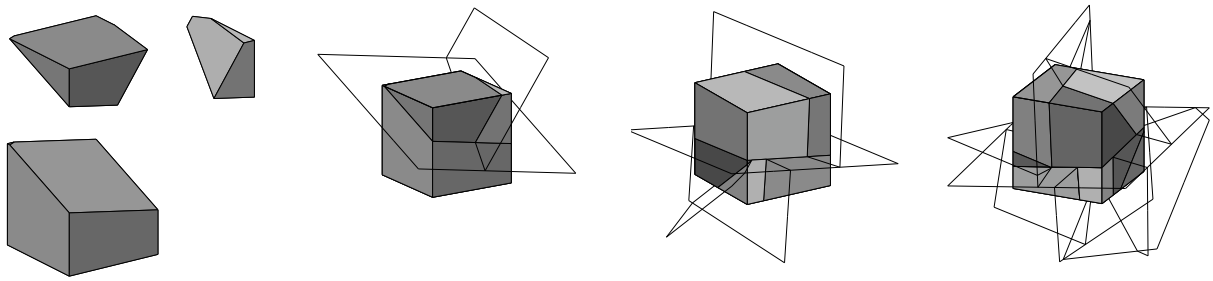
Table 1: Characteristics of the three-dimensional polyhedra from the literature used in the experiments.The rightmost 5 columns describe the sets of polyhedra used in the problems presented in the originating papers. A number in one of these columns is the number of copies of the polyhedra in the corresponding problem instance, e.g., 6 copies of the polyhedron named Convex3 is present in the problem instance Stoyan3.

6.2 Puzzles

To further test the capabilities of our solution method, we devised and implemented a generator for random problem instances. The generator creates a problem instance by splitting a three-dimensional cube into smaller pieces. The pieces along with container dimensions matching the width and height of the cube constitute a problem instance for which the optimal utilization is known to be 100%.

A set of half-spaces $\mathcal{H}$ can be used to define a convex polyhedron as the set of points which is contained in all of the half-spaces. This polyhedron can be found by generating the set, I, of all intersection points of distinct planes p,q,r with $p,q,r \in \mathcal{H}$, and then generate the convex hull $\mathcal{C}$ of the subset of points from I which are contained in all of the half-spaces. An elegant way to find the points of the convex hull is by using the concept of dualisation (see de Berg et al. [9]). Let $\mathcal{C}^d$ be the convex hull of the dual points of $\mathcal{H}$, then the planes of the facets of $\mathcal{C}^d$ are duals of the corner points of $\mathcal{C}$. This allows one to find $\mathcal{C}$ in time $O(n \log n)$ (de Berg et al. [9]) using just a convex hull algorithm.

Given a positive integer n, the construction of an n-piece puzzle commences as follows. Initially, a set of 6 half-spaces, H_0, is generated such that they correspond to a cube. Now let $\mathcal{P}_1 = \{H_0\}$ then we will iteratively construct a sequence of half-space sets $\mathcal{P}_i$. To do this, we select the smallest cardinality half-space set $H \in \mathcal{P}_i$ for each i and generate a random plane which can be used to split H into two new



(a) (b) (c) (d)

Figure 9: Examples of three different puzzles and the cutting planes used to generate them. (a) Three convex polyhedra. (b) The corresponding cube and cutting planes. (c) A puzzle with 5 pieces. (d) A puzzle with 10 pieces.

sets H' and H''. We then let $\mathcal{P}_{i+1} = (\mathcal{P}_i \setminus \{H\}) \cup \{H', H''\}$, i.e., the set of half-space sets containing H' and H'' as well as all sets from $\mathcal{P}_i$ except $\mathcal{H}$. Since the cardinality $|\mathcal{P}_i| = i$, it follows that $|\mathcal{P}_n| = n$. If the random plane used to split each half-space set has been selected appropriately we may generate n non-empty convex polyhedrons from the half-space sets in $\mathcal{P}_n$. In Figure 9, three examples of various sizes are visualized including the cutting planes used to generate them.

It is important to emphasize that a solution method specifically designed with this type of instances in mind may be able to find better solutions more efficiently than our general method. However, since the optimal utilization for these instances is 100%, we may use them to evaluate the quality of the solutions produced by the heuristic. It is interesting to see if we can actually solve some of them even though the solution method is obviously not ideal for puzzle-solving.

6.3 Benchmarks

The two problems given by Ikonen et al. [20] (see Table 1) are decision problems with a cylindrical container and they have already been shown to be easily solved by Egeblad et al. [A]. The only previous results and thus also the best results for the Stoyan instances are reported by Stoyan et al. [27] and their results are repeated in the first two columns of Table 2.

	Stoyan		Bounding box		3DNEST		
Problem	Height	Util. (%)	Height	Util. (%)	Height	Util. (%)	Improv.
Stoyan1	27.0	29.88	46	17.54	19.31	42.05 (3.4)	12.17
Stoyan2	30.92	27.21	34	24.75	19.83	42.45 (1.0)	15.24
Stoyan3	45.86	29.33	45	29.90	29.82	45.12 (0.8)	15.79

Table 2: The results from Stoyan et al. [27] are compared to the average results of 10 runs of 10 minutes with 3DNEST. Results for the initial solution found by 3DNEST is also reported. The second last column includes the standard deviation over the 10 runs. The last column emphasizes the difference between 3DNEST and the approach by Stoyan et al.

In Table 2, we also report the results of the initial solution found using the algorithm described in Section 5.2 and the average results found by running 3DNEST with 10 different random seeds and 10 minutes for each seed. Note that the initial solution found is actually slightly better than the solution found by Stoyan et al. [27] for the largest problem instance. The last column of Table 2 emphasizes the percentage of material saved (on average) when 3DNEST is compared to the results from Stoyan et al. The utilization of 3DNest is on an average about 14 percentage points higher than that of Stoyan et al., and the average improvement over the utilization of Stoyan et al. is 50.2%. This demonstrates that 3DNest performs very well in comparison to existing methods.

In order to make some additional experiments, a new problem instance called Merged1 was created by combining the polyhedra from Stoyan and Ikonen. It contains one copy of each of the polyhedra in Table 1 and the Ikonen polyhedra have been scaled with a factor of 4 to better match the size of the Stoyan polyhedra. Larger versions of this problem instance called Merged*i* are simply created by making *i* number of copies of each polyhedron. The dimensions of the container for these instances are chosen such that a solution with 50% utilization is a cube. Furthermore, we have used the puzzle generator described above to generate 40 puzzles with 5, 10, 20, and 40 pieces. Rather than testing each puzzle with 10 different random seeds, 10 different puzzles are tested for each of the four cardinalities. The purpose of this is to illustrate the heuristic's capabilities independently of the input data. Each shape of these puzzles has an average of about 11-13 facets which is very similar to the Stoyan instances.

Results are presented in Table 3. The utilization of the initial solution (0 seconds) and the utilization after 10, 60, 300, and 600 seconds are reported. All values are averages over 10 runs with different seeds for 3DNEST, except in the case of the puzzles where the seed is used to vary the problem instance. The best solution after 10 minutes, the standard deviation, and the average number of translations done per second are reported for each instance.

		Utilization after number of seconds					Max.	Std.	Translations
Problem	Size	0	10	60	300	600	util.	dev.	per second
Stoyan1	7	17.54	39.76	41.60	42.05	42.05	46.38	3.4	1468
Stoyan2	12	24.75	38.25	39.90	41.79	42.45	44.27	1.0	887
Stoyan3	25	29.90	39.19	42.49	44.58	45.12	46.67	0.8	756
Merged1	15	23.44	37.29	39.68	42.38	42.97	44.12	1.0	462
Merged2	30	23.62	30.23	39.77	42.80	42.92	42.99	0.1	295
Merged3	45	24.58	27.02	35.49	42.23	43.32	44.99	1.0	265
Merged4	60	24.80	26.09	31.61	40.06	41.99	42.81	0.5	233
Merged5	75	26.17	26.66	29.63	37.99	40.96	42.56	0.7	199
Puzzle5	5	28.85	98.30	98.89	98.89	99.22	100.00	2.2	-
Puzzle10	10	20.90	72.68	84.96	93.74	94.30	100.00	14.4	353
Puzzle20	20	15.77	42.27	50.05	72.20	82.54	95.16	12.2	205
Puzzle40	40	13.62	26.40	34.56	45.68	49.59	70.85	7.8	145

Table 3: Average results obtained by running 3DNEST 10 times for at most 10 minutes in each run. Results include the utilization obtained with the initial solution, within 10 seconds and within 1, 5, and 10 minutes. The maximum utilization and standard deviation is also included for the results after 10 minutes. Finally, the average number of translations per second is presented except in the case of Puzzle5, for which an optimal solution was often found within a second. Note that the results for the Puzzle problems are on 10 different instances rather than with 10 different random seeds.

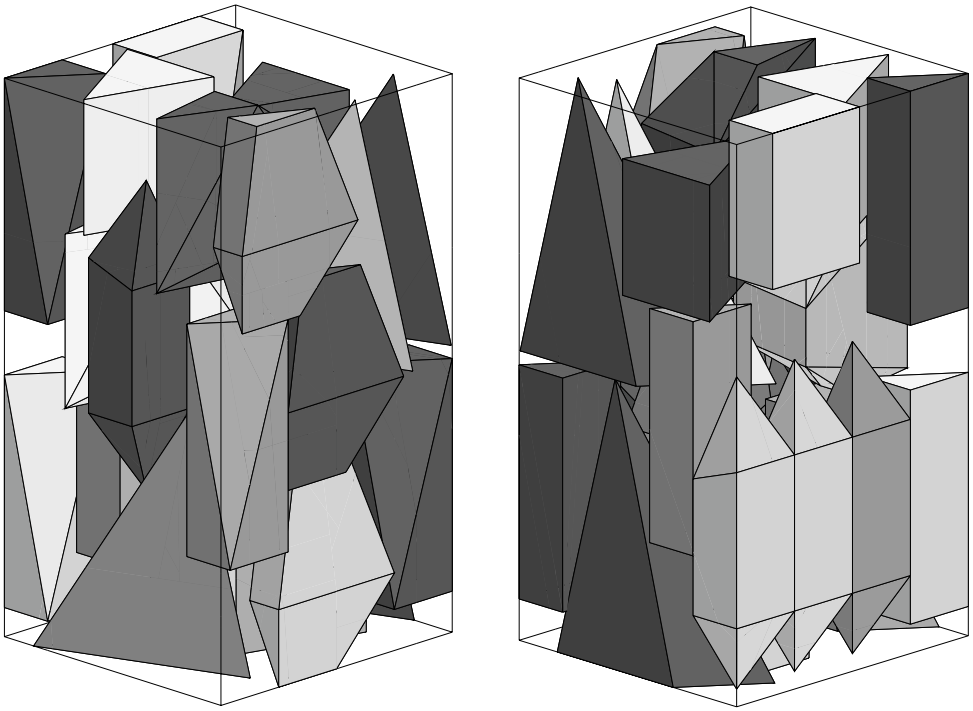


Figure 10: The best solution found for Stoyan3 without using rotation (from two different angles). The utilization is 46.67%.

After 10 seconds the results are already better than those of Stoyan et al. for all of the Stoyan instances with an average utilization close to 40%. The heuristic continues to improve solutions but utilization is only improved by less than 1 percentage point after 300 seconds. Solutions for the Merged instances appear to be quite good with utilizations matching the smaller Stoyan instances even with as many as 60 and 75 shapes. Also here, solutions are generally only improved by one percentage point after the first 300 seconds with the exception of Merged5. The optimal utilization for Merged5 is probably higher than it is for Merged1 which is also indicated by the initial solutions, but the slow decline in the number of translations performed is also a strong indication that large problem instances are solved efficiently by 3DNEST. Puzzles with 5 or 10 pieces are most often solved to optimality, and even puzzles with 20 pieces are handled quite well within the time limit of 10 minutes. The average utilization of these instances is only 50% after 10 minutes, and the best found utilization is less than 71% which is far from the optimal 100%. The best solutions found for Stoyan3 and Merged5 are shown in Figure 10 and Figure 11.

A simple strategy for handling rotation has also been implemented in 3DNEST. The local search neighborhood was expanded, so that in addition to trying translations in three directions, 24 different orientations (90° increments for each axis) are also tried. In each iteration the translation or orientation which results in least overlap is chosen. This was mainly done to get an indication of the improvement possible in the utilization when allowing rotation. The results are presented in Table 4 and better results are indeed obtained for the Stoyan instances while some of the large Merged instances are not handled very well, most likely because of the increased amount of computations needed and the increased size of the solution space.

A lower bound on the height of the Stoyan instances and Merged1 is 16 since these instances

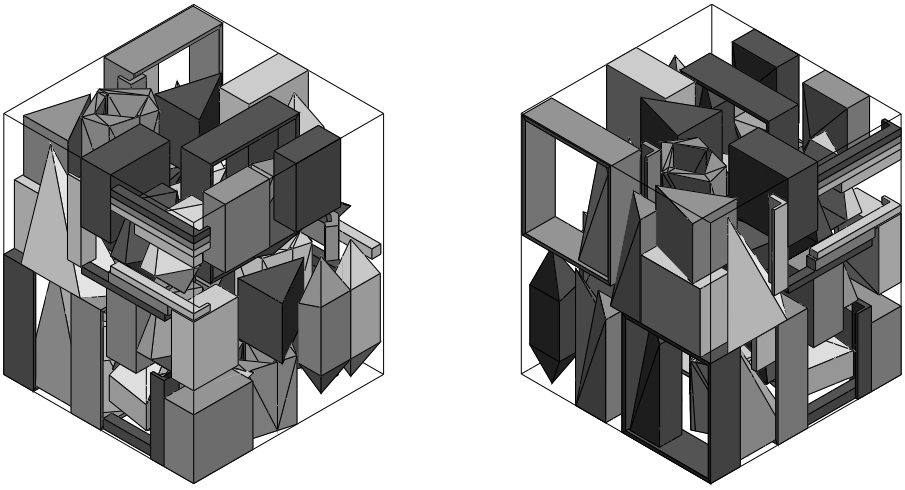


Figure 11: The best solution found for Merged5 without using rotation (from two different angles). The utilization is 42.12%.

contain a shape (see Convex4 in Table 1) with height 16 which cannot be rotated and still be within the bounds of the container. In all runs on Stoyan1 and Merged1 as well as most runs on Stoyan2, 3DNest is able to find solutions matching this bound and therefore these solutions are optimal.

7 Conclusion

In this paper we have presented a solution method for the multi-dimensional strip-packing problem. An earlier version of this method was previously tested by Egeblad et al. [A] and proved very successful for two dimensions. Three problem instances in three dimensions, by Stoyan et al. [27], were used to show that the presented solution method is able to reach far better results than those by Stoyan et al. [27]. The heuristic has also been tested on problems where the optimal value is known, and has proven able to find the optimal solution for instances with 10 items and close to optimal solutions for instances with 20 items. A simple rotation scheme shows that increased utilization may be achieved by allowing rotation, and optimal solutions are found for instances with 7 and even 15 items.

The translation algorithm presented in Section 4 is strongly connected to packing problems, but it is important to emphasize that the algorithm could also be used to maximize the volume of intersection of polytopes with an axis-aligned translation. Also, the restriction to axis-aligned translations is imposed only in order to keep the mathematical details as simple as possible. It is, of course, possible to alter the algorithm for translation in an arbitrary direction: a trivial approach would be to rotate the input data.

It is also important to note the simplicity of the translation algorithm which is able to work directly with the faces of the polyhedrons. Unlike many other methods, as presented in Section 3, we do not rely on additional approximating data-structures such as octrees, depth maps or voxels. Even though intersection volumes of non-convex polytopes are calculated, the intersections are never explicitly

Problem	Size	Average Height	Utilization Avg.	Min.	Max.	Translations per second
Stoyan1	7	16.00	50.42 (0.0)	50.42	50.43	1325.21
Stoyan2	12	16.13	52.17 (0.5)	51.15	52.58	682.27
Stoyan3	25	25.60	52.57 (1.2)	50.41	54.02	673.50
Merged1	15	16.00	46.87 (0.0)	46.87	46.87	440.18
Merged2	30	20.97	45.09 (1.3)	43.50	47.72	305.21
Merged3	45	26.50	40.83 (0.9)	39.75	42.95	299.31
Merged4	60	32.93	36.25 (1.9)	33.44	39.26	275.15
Merged5	75	40.27	31.89 (1.2)	29.31	33.52	242.22

Table 4: Results obtained when allowing rotation.Average utilization, standard deviation, minimum and maximum utilization are reported. The last column is the average number of translations per second.

constructed. Non-convex polytopes are handled as easily as the convex ones and even holes are handled without any changes to the algorithm. Other problem variants might include non-rectangular containers [18], quality regions [18], repeated patterns [24] and more. Although these references are for the 2D problem, the generalized constraints can be handled by our solution method in any dimension, essentially without affecting the running time.

If one does not calculate the minimum between each set of breakpoints in the translation algorithm, then only rational numbers are needed for the solution method described (given that the input problem only uses rational numbers). This property permits the use of the method if exact calculations are needed for some reason. In two dimensions, a minimum between breakpoints can also be found using rational numbers since one only needs to solve linear equations.

Mount et al. [23] described what is essentially a 2D translation algorithm in 2D space (solving 2D2DTP). Given polygons with n and m edges, the worst case running time is $O((mn)^2)$. Their approach is based on an *arrangement* of line segments. Decomposition techniques for $d \geq 3$ have been studied by several authors (see de Berg et al. [7]), and it would be interesting if these methods can be used to generalize the solution method for 2D2dTP to dDdDTP. It is also an open question how to make an algorithm for 2D3DTP or more generally d'DdDTP for any $d \geq 3$ and $d' \in \{2,\ldots,d-1\}$. For the sake of completion, de Berg et al. [8] solve the maximization variant of 2D2DTP with two convex polygons in time $O((n+m)\log(n+m))$. Ahn et al. [1] consider a generalization of the same problem in which they allow rotation.

Finally, Cheong et al. [5] present an approximation algorithm, that finds a translation for two general polygons where the area of overlap is at least $\mu_{opt} - \varepsilon$, for some given value ε and μ_{opt} is the maximal overlap of any translation. If the polygons have complexity n and m, the running time of their algorithm is $O(m + (n^2/\varepsilon^4)\log^2 n)$ and $O(m + (n^3/\varepsilon^4)\log^5 n)$, if rotations are allowed.

Free orientation of shapes is one of the most important directions for future research. Especially when considering that most applications of packing 3D shapes, e.g., rapid prototyping, do allow free orientation. Another important direction for future research is how to also handle some of the constraints which are typically part of more general layout problems, e.g., constraints concerning gravity or wire length [4].

References

[A] J. Egeblad, B. K. Nielsen, and A. Odgaard. Fast neighborhood search for two- and three-dimensional nesting problems. *European Journal of Operational Research*, 183(3):1249–1266, 2007.

[1] H.-K. Ahn, O. Cheong, C.-D. Park, C.-S. Shin, and A. Vigneron. Maximizing the overlap of two planar convex sets under rigid motions. *Computational Geometry*, 37(1):3–15, 2006.

[2] A. Bortfeldt, H. Gehring, and D. Mack. A parallel tabu search algorithm for solving the container loading problem. *Parallel Computing*, 29:641–662, 2002.

[3] J. Cagan, D. Degentesh, and S. Yin. A simulated annealing-based algorithm using hierarchical models for general three-dimensional component layout. *Computer Aided Design*, 30(10):781–790, 1998.

[4] J. Cagan, K. Shimada, and S. Yin. A survey of computational approaches to three-dimensional layout problems. *Computer-Aided Design*, 34(8):597–611, 2002.

[5] O. Cheong, A. Efrat, and S. Har-Peled. Finding a guard that sees most and a shop that sells most. *Discrete & Computational Geometry*, 37(4):545–563, 2007.

[6] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright. Simplification envelopes. *Computer Graphics*, 30(Annual Conference Series):119–128, 1996.

[7] M. de Berg, L. J. Guibas, and D. Halperin. Vertical decompositions for triangles in 3-space. *Discrete & Computational Geometry*, 15(1):35–61, 1996.

[8] M. de Berg, O. Cheong, O. Devillers, and M. van Kreveld. Computing the maximum overlap of two convex polygons under translations. *Theory of Computing Systems*, 31(5):613–628, 1998.

[9] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications (2nd edition)*. Springer, 2000.

[10] J. K. Dickinson and G. K. Knopf. A moment based metric for 2-D and 3-D packing. *European Journal of Operational Research*, 122(1):133–144, 2000.

[11] J. K. Dickinson and G. K. Knopf. Packing subsets of 3d parts for layered manufacturing. *International Journal of Smart Engineering System Design*, 4(3):147–161, 2002.

[12] K. A. Dowsland and W. B. Dowsland. Solution approaches to irregular nesting problems. *European Journal of Operational Research*, 84:506–521, 1995.

[13] F. Eisenbrand, S. Funke, A. Karrenbauer, J. Reichel, and E. Schömer. Packing a trunk: now with a twist! In *SPM '05: Proceedings of the 2005 ACM symposium on Solid and physical modeling*, pages 197–206, New York, NY, USA, 2005. ACM Press.

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

[15] 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.

[16] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Information Processing Letters*, 12(3):133–137, 1981.

[17] P. Hachenberger and L. Kettner. 3D Boolean Operations on Nef Polyhedra. In C. E. Board, editor, *CGAL-3.2 User and Reference Manual*. 2006.

[18] J. Heistermann and T. Lengauer. The nesting problem in the leather manufacturing industry. *Annals of Operations Research*, 57:147–173, 1995.

[19] S.-M. Hur, K.-H. Choi, S.-H. Lee, and P.-K. Chang. Determination of fabricating orientation and packing in sls process. *Journal of Materials Processing Technology*, 112(2-3):236–243, 2001.

[20] I. Ikonen, W. E. Biles, A. Kumar, J. C. Wissel, and R. K. Ragade. A genetic algorithm for packing three-dimensional non-convex objects having cavities and holes. In *Proceedings of the 7th International Conference on Genetic Algortithms*, pages 591–598, East Lansing, Michigan, 1997. Morgan Kaufmann Publishers.

[21] J. Lawrence. Polytope volume computation. *Mathematics of Computation*, 57(195):259–271, 1991.

[22] 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.

[23] D. M. Mount, R. Silverman, and A. Y. Wu. On the area of overlap of translated polygons. *Computer Vision and Image Understanding*, 64(1):53–61, 1996.

[24] B. K. Nielsen. An efficient solution method for relaxed variants of the nesting problem. In J. Gudmundsson and B. Jay, editors, *Theory of Computing, Proceedings of the Thirteenth Computing: The Australasian Theory Symposium*, volume 65 of *CRPIT*, pages 123–130, Ballarat, Australia, 2007. ACS.

[25] B. K. Nielsen. *Nesting Problems and Steiner Tree Problems*. PhD thesis, DIKU, University of Copenhagen, Denmark, 2008.

[26] T. Osogami. Approaches to 3D free-form cutting and packing problems and their applications: A survey. Technical Report RT0287, IBM Research, Tokyo Research Laboratory, 1998.

[27] Y. G. Stoyan, N. I. Gil, G. Scheithauer, A. Pankratov, and I. Magdalina. Packing of convex polytopes into a parallelepiped. *Optimization*, 54(2):215–235, 2005.

[28] P. E. Sweeney and E. R. Paternoster. Cutting and packing problems: A categorized, application-orientated research bibliography. *Journal of the Operational Research Society*, 43(7):691–706, 1992.

[29] C. Voudouris and E. Tsang. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113:469–499, 1999.

[30] G. Wäscher, H. Haussner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 2006.

[31] X. Yan and P. Gu. A review of rapid prototyping technologies and systems. *Computer Aided Design*, 28(4):307–318, 1996.

[32] S. Yin and J. Cagan. An extended pattern search algorithm for three-dimensional component layout. *Journal of Mechanical Design*, 122(1):102–108, 2000.

[33] S. Yin and J. Cagan. Exploring the effectiveness of various patterns in an extended pattern search layout algorithm. *Journal of Mechanical Design*, 126(1):22–28, 2004.

In press (available online). *Computers and Operations Research*, 2007

Heuristic approaches for the two- and three-dimensional knapsack packing problem

Jens Egeblad* and David Pisinger

Abstract

The maximum profit two- or three-dimensional knapsack packing problem packs a maximum profit subset of some given rectangles or boxes into a larger rectangle or box of fixed dimensions. Items must be orthogonally packed, but no other restriction is imposed to the problem. We present a new iterative heuristic for the two-dimensional knapsack problem based on the sequence pair representation proposed by Murata et al. (1996) using a semi-normalized packing algorithm by Pisinger (2006). Solutions are represented as a pair of sequences. In each iteration, the sequence pair is modified and transformed to a packing in order to evaluate the objective value. Simulated annealing is used to control the heuristic. A novel abstract representation of box placements, called sequence triple, is used with a similar technique for the three-dimensional knapsack problem . The heuristic is able to handle problem instances where rotation is allowed. Comprehensive computational experiments which compare the developed heuristics with previous approaches indicate very promising results for both two- and three-dimensional problems.

1 Introduction

Assume that we are given a set of n rectangles $j = 1,\ldots,n$, each having a width w_j, height h_j and profit p_j and a rectangular plate having width W and height H. The *maximum profit two-dimensional knapsack packing* problem (2DKP) assigns a subset of the rectangles onto the plate such that the associated profit sum is maximized. All coefficients are assumed to be nonnegative integers, and the rectangles may not be rotated. A packing of rectangles on the plate is feasible if no two rectangles overlap, and if no part of any rectangle exceeds the plate.

The *maximum profit three-dimensional knapsack packing* problem (3DKP) assigns a subset of boxes each with dimensions w_j, h_j, d_j into a larger box with dimensions W, H and D.

The problem has direct applications in various packing and cutting problems where the task is to use the space or material in an optimal way. The 2DKP problem also appears as pricing problem when solving the two-dimensional bin-packing problem [11, 31, 32]. 2DKP and 3DKP are NP-hard in the strong sense, which can be shown by reduction from the one-dimensional bin packing problem. An extensive survey on cutting and packing as well as a useful classification of these problems was developed by Wascher, Haussner and Schumann [33].

The problems we consider in this paper can be classified as two- and three-dimensional rectangular single knapsack problems (SKP) according to the typology of Wascher, Haussner and Schumann [33]. The items considered are strongly heterogeneous and we consider problems with and without rotation.

A related problem is the constrained two-dimensional orthogonal non-guillotine cutting problem. Here equal items are grouped in types and for each item-type there are both a lower bound and an

*Corresponding Author: Tel.: +45 35 32 14 00; fax: +45 35 32 14 01. E-mail addresses: jegeblad@diku.dk (J. Egeblad), pisinger@diku.dk (D. Pisinger)

upper bound on the number of that type required in the solution. Therefore the constrained non-guillotine cutting problem may be seen as a generalization of the orthogonal knapsack packing problem. Instances for the constrained non-guillotine cutting problem are often weakly heterogeneous, and solution methods commonly take advantage here of.

Integer Programming formulations of the 2DKP have been presented by Beasley [5], Hadjiconstantinou and Christofides [15], and Boschetti, Hadjiconstantinou, Mingozzi [7] among others.

Fekete and Schepers [11, 12, 14] solved the 2- and 3DKP through a branch-and-bound algorithm which assigns items to the knapsack without specifying the position of the rectangles. For each assignment of items a two-dimensional packing problem is solved, deciding whether a feasible assignment of coordinates to the items is possible such that they all fit into the knapsack without overlaps. An advanced graph representation was used for solving the latter problem. Baldacci and Boschetti [3] used a similar approach but introduced new reduction-tests and a cutting-plane approach to compute more effective bounds. Pisinger and Sigurd [32] solved the 2DKP through a branch-and-cut approach in which an ordinary one-dimensional knapsack problem is used to select the most profitable items whose overall area does not exceed the area of the plate. Having selected the most profitable items, a two-dimensional packing problem in decision form is solved, through constraint programming. If all items can be placed in the knapsack the algorithm terminates, otherwise an inequality is added to the one-dimensional knapsack stating that not all the current items can be selected simultaneously, and the process is repeated. Finally, Caprara and Monaci [8] developed a branch-and-bound algorithm for the 2DKP. The algorithm is based on a branch-and-bound scheme which assigns items to the knapsack without specifying the position of each item, followed by a feasibility check. The latter is done using an enumeration scheme from Martello, Monaci, Vigo [25].

Several authors have also applied heuristics to the constrained non-guillotine packing variant of the problem. Lai and Chan [20, 21] use simulated annealing and genetic algorithms. Solutions are represented as a sequence of items that are transformed into a placement. Only limited computational results are reported. The work by Leung et al. [22, 23] is based on that of Lan and Chan and the bottom-left placement procedure introduced by Jakobs [18]. Beasley [4] described a heuristic based on genetic algorithms capable of efficiently generating good solutions for instances with up to 4000 pieces. However, the heuristic is unable to reproduce known optimal solutions for smaller instances. Intermediate solutions explicitly state coordinates of rectangles, and overlap of items are allowed during the solution process. A penalty in the objective function ensures that overlap is minimized. More recently Alvarez-Valdes et al. applied both of the meta-heuristics GRASP and Tabu-search [1, 2] to the problem and were able to achieve very impressive results on the data used by Beasley.

In the present paper we first present an IP formulation of the 2- and 3DKP. In Section 3 we describe the sequence pair representation, which we use in Section 4 with a simple local search neighborhood controlled by Simulated Annealing to solve 2DKP. In Section 5 we introduce a novel abstract representation of box placements in three dimensions and use the same methods as for two dimensions to solve 3DKP. Finally in Section 6 we present our result on existing and new benchmarks instances for 2- and 3DKP.

2 Integer programming formulation of the problem

In the following we show an integer programming formulation of the 3DKP. A formulation of 2DKP easily follows by removing variables and constraints for the third dimension.

We will introduce the decision variable s_i to indicate whether box i is packed within the knapsack box. The coordinates of box i are (x_i, y_i, z_i), meaning that the lower left back corner of the box

is located at this position. If a rectangle is not packed within the knapsack we may assume that $(x_i, y_i, z_i) = (0,0,0)$. As no part of a packed box may exceed the knapsack, we have the obvious constraints

$$0 \le x_i \le W - w_i, \qquad 0 \le y_i \le H - h_i, \qquad 0 \le z_i \le D - d_i. \tag{1}$$

We introduce the binary decision variables ℓ_{ij} (left), r_{ij} (right), u_{ij} (under), o_{ij} (over), b_{ij} (behind) and f_{ij} (in-front), to indicate the relative position of boxes i, j where $i < j$. To ensure that no two packed boxes i, j overlap we will demand that

$$\ell_{ij} + r_{ij} + u_{ij} + o_{ij} + b_{ij} + f_{ij} \ge 1, \tag{2}$$

whenever $s_i = s_j = 1$. Depending on the relative position of two rectangles the coordinates must satisfy the following inequalities

$$\begin{array}{llll} \ell_{ij} = 1 & \Rightarrow & x_i + w_i \le x_j, & r_{ij} = 1 \;\Rightarrow\; x_j + w_j \le x_i, \\ u_{ij} = 1 & \Rightarrow & y_i + h_i \le y_j, & o_{ij} = 1 \;\Rightarrow\; y_j + h_j \le y_i, \\ b_{ij} = 1 & \Rightarrow & z_i + d_i \le z_j, & f_{ij} = 1 \;\Rightarrow\; z_j + d_j \le z_i. \end{array} \tag{3}$$

The problem may now be formulated as

$$\begin{array}{lll} \max & \displaystyle\sum_{i=1}^{n} p_i s_i & \\ \text{s.t.} & \ell_{ij} + r_{ij} + u_{ij} + o_{ij} + b_{ij} + f_{ij} \ge s_i + s_j - 1 & i,j = 1,\ldots,n \\ & x_i - x_j + W\ell_{ij} \le W - w_i & i,j = 1,\ldots,n \\ & x_j - x_i + W r_{ij} \le W - w_j & i,j = 1,\ldots,n \\ & y_i - y_j + H u_{ij} \le H - h_i & i,j = 1,\ldots,n \\ & y_j - y_i + H o_{ij} \le H - h_j & i,j = 1,\ldots,n \\ & z_i - z_j + D b_{ij} \le D - d_i & i,j = 1,\ldots,n \\ & z_j - z_i + D f_{ij} \le D - d_j & i,j = 1,\ldots,n \\ & 0 \le x_i \le W - w_i & i = 1,\ldots,n \\ & 0 \le y_i \le H - h_i & i = 1,\ldots,n \\ & 0 \le z_i \le D - d_i & i = 1,\ldots,n \\ & \ell_{ij}, r_{ij}, u_{ij}, o_{ij}, b_{ij}, f_{ij} \in \{0,1\} & i,j = 1,\ldots,n \\ & s_i \in \{0,1\} & i = 1,\ldots,n \\ & x_i, y_i, z_i \ge 0 & i = 1,\ldots,n \end{array} \tag{4}$$

The first constraint ensures that if boxes i and j are packed, then they must be located left, right, under, over, behind or in-front of each other as stated in (2). The next six constraints are just linear versions of the constraints (3). The last three inequalities correspond to the constraints (1).

The MIP-model has $6n^2 + n$ binary decision variables and $3n$ continuous variables. Although the size of $O(n^2)$ binary variables is not alarming, the problem is difficult to solve. This is mainly due to the use of conditional constraints (3), as these will loose their effect when solving the LP-relaxation, and thus bounds from LP-relaxation are in general far from the MIP-optimal solution value.

3 Sequence pairs

Murata et al. [17] presented an abstract representation of two-dimensional rectangle packings based on sequence pairs. The problem they consider is the minimum area enclosing rectangle packing problem.

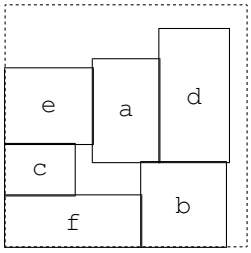


Figure 1: A packing represented by sequence A = <e,c,a,d,f,b> and sequence B = <f,c,b,e,a,d>.

In the abstract representation every compact packing can be represented by two permutations of the numbers $\{1,2,\ldots,n\}$ where each number represents a rectangle in the problem instance. The pair of permutations is called a *sequence pair* (A,B).

For a given packing, the two permutations A and B are found as follows: We use the relation $\mathcal{A}_{ij}$ to denote that item i precedes j in sequence A. We define

$$(x_i + w_i \leq x_j \quad \vee \quad y_i \geq y_j + h_j) \quad \Leftrightarrow \quad \mathcal{A}_{ij} \tag{5}$$

In a similar way we use the relation $\mathcal{B}_{ij}$ to denote that item i precedes j in sequence B, defining

$$(x_i + w_i \leq x_j \quad \vee \quad y_i + h_i \leq y_j) \quad \Leftrightarrow \quad \mathcal{B}_{ij} \tag{6}$$

Each of the two relations $\mathcal{A},\mathcal{B}$ given by (5) and (6) defines a semi-ordering, and hence for a given packing the two permutations A and B can easily be found by repeatedly choosing one (of possibly more) minimum elements. Figure 1 illustrates a packing and a corresponding sequence pair (A,B).

From (5) and (6) we immediately see that if item i precedes item j in both sequences, then i must be placed left of j. If i succeeds j in sequence A but i precedes j in sequence B then i must be placed under j. Formally we have

$$\mathcal{A}_{ij} \wedge \mathcal{B}_{ij} \quad \Rightarrow \quad i \text{ is left of } j \tag{7}$$
$$\neg\mathcal{A}_{ij} \wedge \mathcal{B}_{ij} \quad \Rightarrow \quad i \text{ is under } j \tag{8}$$

where we use the terminology $\neg\mathcal{A}_{ij}$ to denote $\mathcal{A}_{ji}$.

The implications (7) and (8) can be used to derive a pair of constraint graphs as illustrated in Figure 2. In both graphs the nodes correspond to the items and edges indicate which rectangles should be placed left of each other (respectively under each other). In the first graph we have an edge from i

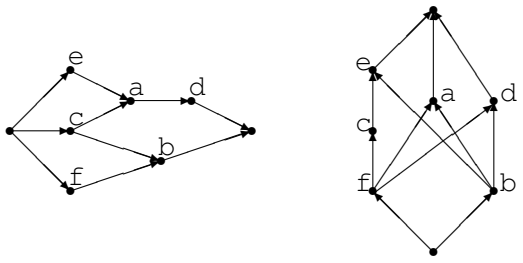


Figure 2: Constraint graphs corresponding to the sequence pair (A,B) = (<e,c,a,d,f,b>,<f,c,b,e,a,d>).

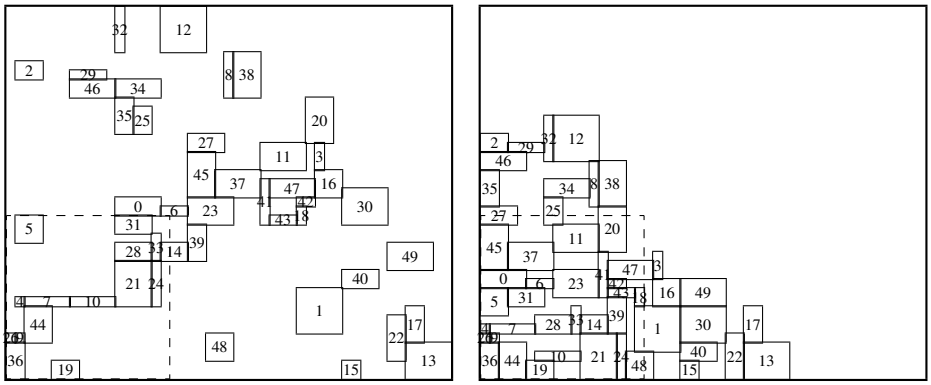


Figure 3: Transformation of a sequence pair to a packing using the ordinary transformation (left) and using the semi-normalized transformation (right).

to j if and only if item i should be placed left of j $(\mathcal{A}_{ij} \wedge \mathcal{B}_{ij})$. In the second graph we have an edge from i to j if and only if item i should be placed under j $(\neg\mathcal{A}_{ij} \wedge \mathcal{B}_{ij})$. Redundant edges are removed from the figure for clarity. Traversing the nodes in topological order while assigning coordinates to the items, a packing (i.e. the coordinates of the items) can be obtained in $O(n^2)$ time. Tang et al. [34, 35] showed how the same packing can be derived without explicitly defining the constraint graph, but by finding weighted longest common subsequences in the sequence pair.

Pisinger [30] further improved the algorithm, by presenting an algorithm which transforms a sequence pair to a *semi-normalized* packing in time $O(n \log\log n)$. A *normalized* packing is a packing where the items are packed according to the sequence B and where each new item is placed such that it touches an already placed item on its left side, and an already placed item on its lower side. A *semi-normalized* packing is a packing where the items are packed according to the sequence B and where each new item is placed such that it touches the *contour* of the already placed items both from left and from below. The difference between a packing based on the ordinary transformation and the semi-normalized packing of the transformation by Pisinger is illustrated in Figure 3.

4 Sequence pairs for two-dimensional knapsack packing

Let any sequence pair represent a feasible solution to the 2DKP. To evaluate the solution we transform the sequence pair into a packing. The solution value is sum of the profit values of those items which are located completely within the knapsack $W \times H$ of the transformed packing. Figure 3 illustrates two such packings which arose from the conventional and semi-normalized transformation of the sequence-pair. The solution values of each packing is the sum of the profits of items within the dashed lines.

The transformation from sequence pair to packing may be stopped as soon as the contour of already placed items is completely outside the knapsack. For problems where only a small fraction of items fit inside the knapsack, this can save substantial time, since generating the packing will take an amount of time which is roughly equal to the time required to place only the subset of items which are inside the knapsack.

This section is organized as follows: In section 4.1 we describe our heuristic in more detail. In Section 4.2 we describe how to accommodate problems where rotation is allowed and in Section 4.3 we describe how problem instances can be simplified during the solution process.

4.1 Simulated annealing

To solve the 2DKP, we use the meta-heuristic Simulated Annealing which works well in cooperation with the sequence pair representation [17, 30, 34, 35].

In this setting we repeatedly make a small modification to the sequence pair, transform the sequence pair into a packing, evaluate the profit of the corresponding packing, and accept the solution depending on the outcome. In Simulated Annealing any non-improving solution is accepted with probability that decreases over time. An outline of the algorithm is found in Figure 4.

```
choose initial incumbent solution s ∈ S
choose initial time t_0
choose time step t_s
a := 0
repeat
    choose s' ∈ N(s)
    if f(s') ≤ f(s) then
        accept := true
    else
        p := rand(0,1)
        T := 1/(t_0 + t_s · a)
        Δ := (f(s') − f(s))/f(s)
        if p < e^(−Δ/T) then
            accept := true
        end
    end
    if accept then
        s := s'
        a := a + 1
    end
until stopping-criteria
return s
```

Figure 4: Simulated Annealing Heuristic

Our variant of Simulated Annealing is as follows: At any given time the temperature is evaluated as $1/(t_0 + t_s \cdot a)$ where t_0 is a start time-value, t_s is a time-step value and a is the number of accepted solutions. The temperature depends on the time, so the higher $t_0 + t_s \cdot a$ is, the lower is the current temperature. The temperature is decreased only when a new solution is accepted.

The neighborhood $N(s)$ of a solution $s = (A, B)$ is defined as one of the following three permutations: Either exchange two items in sequence A; exchange two items in sequence B; or exchange two items in both sequence A and B. The items are selected randomly.

4.2 Rotations

Few papers consider exact algorithms for packing problems where rotation is allowed. A possible explanation could be the increased size of the solution space and the lack of high-quality upper bounds. In our heuristic, rotations are easy to handle as we may represent each packing by the triple (A, B, R). Here (A, B) is the sequence pair and $R = (r_1, \ldots, r_n)$ is a binary vector of length n. In a placement of

(A,B,R) item i is rotated $r_i \cdot 90$ degrees. If rotation is allowed the neighborhood $N(s)$ of our heuristic is extended with a fourth permutation: Change the rotation flag of an item in R.

4.3 Removing items

In instances where only a small fraction of the items can fit inside the knapsack, it can be advantageous to a-priori remove some items which provably will not be selected in an optimal solution. Our approach is based on standard techniques for reducing the number of items in a 0-1 Knapsack Problem [19] and is similar to the one presented in [1].

Let the 1-dimensional relaxation of the (2DKP) be given by the following 0-1 Knapsack Problem (1DKP):

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} p_j x_j \\ \text{s.t.} \quad & \sum_{j=1}^{n} (w_j h_j) x_j \leq WH \\ & x_j \in \{0,1\}, j = 1,\ldots,n \end{aligned} \tag{9}$$

Assume that z^* is the currently best known solution to (2DKP), then clearly z^* is a lower bound for (1DKP), and an optimal solution to (1DKP) is also an upper bound on z^*.

Now, assume that we have an upper bound u_j^1 for (1DKP) with the additional constraint that $x_j = 1$. If $u_j^1 \leq z^*$ then we know that item j will not be chosen in an improved solution of (1DKP), and hence neither in an improved solution of (2DKP).

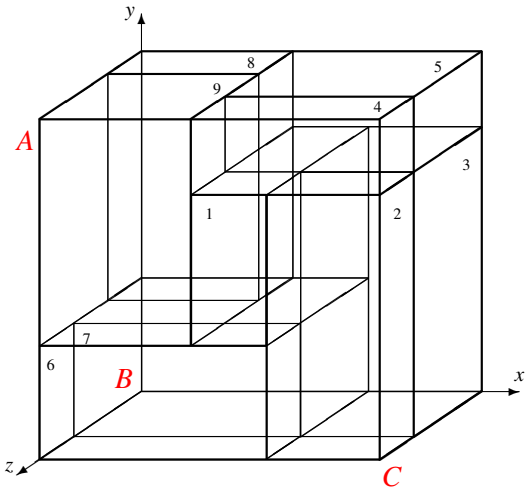
As our upper bound we have chosen to use the Dembo and Hammer [10] upper bound u_{DH} which can be calculated as follows: Sort the items in (1DKP) according to nonincreasing efficiency $p_j/(w_j h_j)$ and fill the knapsack in a greedy way until the first item s (split items) does not fit into the knapsack. Then the Dembo and Hammer upper bound is given by $u_{DH} = \sum_{i=1}^{s-1} p_i + (WH - \sum_{i=1}^{s-1} w_i h_i) p_s/(w_s h_s)$. If we fix a variable $x_j = 1$ the upper bound becomes $u_j^1 = u_{DH} + p_j - (w_j h_j) p_s/(w_s h_s)$ which can be calculated in constant time.

During the simulated annealing, we run this test whenever we encounter an improving solution z^* and remove every item j for which $u_j^1 \leq z^*$ from the problem.

5 Three dimensions

For the three-dimensional problem we will consider a new representation which like the sequence pair for two dimensions will contain the relative box placements for three dimensions. We call the representation sequence triple since it consists of three sequences. Not all three-dimensional packings are obtainable with this representation but we will prove that a large subset of all normalized packings, known as fully robot packable packings, may be represented.

A *robot packing* is a packing which can be achieved by successively placing boxes starting from the bottom-left-behind corner, and such that each box is in-front of, right of, or over each of the previously placed boxes [26]. Robot packings are motivated by several industrial applications, where boxes have to be packed by robots equipped with a rectangular "hand" parallel to the base of the large box. To avoid collisions between the hand and the boxes, it is demanded that no already packed box blocks for the "hand" movement. In [26] it is shown that the quality of a packing is seldom affected by restricting the solution space to the set of robot packings.



i	w_i	h_i	d_i	x_i	y_i	z_i
1	2	4	9	4	3	0
2	3	7	3	6	0	6
3	3	7	6	6	0	0
4	5	2	3	4	7	6
5	5	2	6	4	7	0
6	6	3	3	0	0	6
7	6	3	6	0	0	0
8	4	6	3	0	3	0
9	4	6	6	0	3	3

Figure 5: A packing and the corresponding sequence triple $(A,B,C) = \{< 9,4,8,5,1,6,2,7,3 >, < 7,8,6,9,1,3,5,2,4 >, < 2,3,6,7,1,4,5,9,8 >\}$.

A packing is a *fully robot packable packing* if all six 90 degree rotations of it are robot packings or, equivalently, the robot criteria is satisfied no matter which corner is selected as start corner instead of the bottom-left-behind corner.

This section is organized as follows: First we describe the Sequence Triple representation in detail in Section 5.1. Then we describe an algorithm to transform a Sequence Triple to a packing in Section 5.2. Finally, in Section 5.3, we describe how the same Simulated Annealing strategy we use for the sequence pair is applied to the sequence triple to form a heuristic for 3DKP.

5.1 Sequence triple

A given fully robot packable packing is represented by three sequences A, B and C where each sequence is a permutation of the n boxes. For any sequence X we define the relation $\mathcal{X}_{ij}$ to mean that i is before j in sequence X. For convenience we use the notation $\neg\mathcal{X}_{ij} \Leftrightarrow \mathcal{X}_{ji}$.

In a similar way as in Section 3 we define the relation $\mathcal{A}_{ij}$ by

$$(x_i + w_i \leq x_j \quad \vee \quad y_i \geq y_j + h_j \quad \vee \quad z_i \geq z_j + d_j) \quad \Leftrightarrow \quad \mathcal{A}_{ij} \tag{10}$$

In other words $\mathcal{A}_{ij}$ iff i is located left, over or in-front of j. Using the formulation (2) we have $\mathcal{A}_{ij} \Leftrightarrow \ell_{ij} + o_{ij} + f_{ij} \geq 1$.

Relation $\mathcal{B}_{ij}$ is defined by

$$(x_i \leq x_j + w_j \quad \vee \quad y_i \leq y_j + h_j \quad \vee \quad z_i \leq z_j + d_j) \quad \Leftrightarrow \quad \mathcal{B}_{ij} \tag{11}$$

This means $\mathcal{B}_{ij}$ iff i is located left, under or behind of j. The relation can be expressed as $\mathcal{B}_{ij} \Leftrightarrow l_{ij} + u_{ij} + b_{ij} \geq 1$.

Finally, relation $\mathcal{C}_{ij}$ is defined by

$$(x_i \geq x_j + w_j \quad \vee \quad y_i + h_i \leq y_j \quad \vee \quad z_i \geq z_j + d_j) \quad \Leftrightarrow \quad \mathcal{C}_{ij} \tag{12}$$

In words, $\mathcal{C}_{ij}$ iff i is located right, under or in-front of j, which can be expressed as $\mathcal{C}_{ij} \Leftrightarrow r_{ij} + u_{ij} + f_{ij} \geq 1$.

Due to the definition of fully robot packable packings, there will always be an item which is located furthest left-over-behind. By removing this item and repeating the operation, we get the ordering of sequence A. In a similar way the orderings of B and C can be determined, as illustrated in Figure 5 (The letters A,B,C on the figure indicate the directions which are used for defining the corresponding sequence of boxes). This shows that every fully robot packable packing can be represented by a sequence triple.

Using the relations

$$\begin{array}{ll} \mathcal{A}_{ij} \Leftrightarrow \ell_{ij} + o_{ij} + f_{ij} \geq 1, & \ell_{ij} + r_{ij} \leq 1, \\ \mathcal{B}_{ij} \Leftrightarrow l_{ij} + y_{ij} + b_{ij} \geq 1, & o_{ij} + u_{ij} \leq 1, \\ \mathcal{C}_{ij} \Leftrightarrow r_{ij} + u_{ij} + f_{ij} \geq 1, & f_{ij} + b_{ij} \leq 1, \end{array} \tag{13}$$

we find that

$$\begin{array}{rcl} \mathcal{A}_{ij} \wedge \neg\mathcal{B}_{ij} \wedge \mathcal{C}_{ij} & \Leftrightarrow & f_{ij} = 1 \\ \mathcal{A}_{ij} \wedge \mathcal{B}_{ij} \wedge \mathcal{C}_{ij} & \Leftrightarrow & \ell_{ij} + r_{ij} \geq 1 \vee o_{ij} + u_{ij} \geq 1 \vee f_{ij} + b_{ij} \geq 1 \\ \neg\mathcal{A}_{ij} \wedge \neg\mathcal{B}_{ij} \wedge \mathcal{C}_{ij} & \Leftrightarrow & r_{ij} = 1 \\ \neg\mathcal{A}_{ij} \wedge \mathcal{B}_{ij} \wedge \mathcal{C}_{ij} & \Leftrightarrow & u_{ij} = 1 \\ \mathcal{A}_{ij} \wedge \neg\mathcal{B}_{ij} \wedge \neg\mathcal{C}_{ij} & \Leftrightarrow & o_{ij} = 1 \\ \mathcal{A}_{ij} \wedge \mathcal{B}_{ij} \wedge \neg\mathcal{C}_{ij} & \Leftrightarrow & \ell_{ij} = 1 \\ \neg\mathcal{A}_{ij} \wedge \neg\mathcal{B}_{ij} \wedge \neg\mathcal{C}_{ij} & \Leftrightarrow & \ell_{ij} + r_{ij} \geq 1 \vee o_{ij} + u_{ij} \geq 1 \vee f_{ij} + b_{ij} \geq 1 \\ \neg\mathcal{A}_{ij} \wedge \mathcal{B}_{i} \wedge \neg\mathcal{C}_{ij} & \Leftrightarrow & b_{ij} = 1 \end{array} \tag{14}$$

Notice that $\mathcal{A}_{ij} \wedge \mathcal{B}_{ij} \wedge \mathcal{C}_{ij}$ and $\neg\mathcal{A}_{ij} \wedge \neg\mathcal{B}_{ij} \wedge \neg\mathcal{C}_{ij}$ cannot occur for any packing. We have, however, chosen to assign these cases a meaning, such that every sequence triple has a corresponding packing. This leads to the following four implications, similar to (7) and (8), which are used to determine the relative box positions:

$$\mathcal{A}_{ij} \wedge \mathcal{B}_{ij} \wedge \neg\mathcal{C}_{ij} \Rightarrow i \text{ is left of } j \tag{15}$$

$$\neg\mathcal{A}_{ij} \wedge \mathcal{B}_{ij} \wedge \mathcal{C}_{ij} \Rightarrow i \text{ is under } j \tag{16}$$

$$\neg\mathcal{A}_{ij} \wedge \mathcal{B}_{ij} \wedge \neg\mathcal{C}_{ij} \Rightarrow i \text{ is behind } j \tag{17}$$

$$\mathcal{A}_{ij} \wedge \mathcal{B}_{ij} \wedge \mathcal{C}_{ij} \Rightarrow i \text{ is behind } j \tag{18}$$

Notice that both (17) and (18) impose that i must be behind j in the packing. The unfortunate consequence of this is that the representation is biased towards orderings in that direction which could have a negative impact on the solution process, but as we wish to let every sequence triple represent a packing, an arbitrary choice had to be done.

5.2 A placement algorithm

To find a placement (i.e. the coordinates of the boxes) corresponding to a sequence triple, we can construct three constraint graphs similar to Figure 2: In the first graph we have an edge from item i to item j if i is located left of j (i.e. $\mathcal{A}_{ij} \wedge \mathcal{B}_{ij} \wedge \neg\mathcal{C}_{ij}$). In the second graph we have an edge from item i to item j if i is located under j (i.e. $\neg\mathcal{A}_{ij} \wedge \mathcal{B}_{ij} \wedge \mathcal{C}_{ij}$). In the last graph we have an edge from item i to item j if i is located behind j (i.e. $\neg\mathcal{A}_{ij} \wedge \mathcal{B}_{ij} \wedge \neg\mathcal{C}_{ij}$ or $\mathcal{A}_{ij} \wedge \mathcal{B}_{ij} \wedge \mathcal{C}_{ij}$). Traversing the nodes in topological order for each graph while assigning coordinates to the items, we find the location of all boxes in time $O(n^2)$.

By observing that $\mathcal{B}_{ij}$ is a necessary criteria for node i to precede node j in each of the three constraint graphs, we may actually omit the topological ordering as it is in each case given by the order of sequence B.

The first box in B is placed at $(x,y,z) = (0,0,0)$ and succeeding boxes are placed one by one according to the order of sequence B. At any time let P consist of all previously placed boxes. Now assume we wish to place box i. To determine the position of i we compare i with every box $j \in P$. Let $P_x \subseteq P$ be the subset of boxes which satisfy (15), (i.e. $\mathcal{A}_{ij} \wedge \mathcal{B}_{ij} \wedge \neg\mathcal{C}_{ij}$) let $P_y \subseteq P$ be the subset which satisfy (16), (i.e. $\neg\mathcal{A}_{ij} \wedge \mathcal{B}_{ij} \wedge \mathcal{C}_{ij}$) and let $P_z \subseteq P$ be the subset which satisfy (17) or (18) (i.e. $\neg\mathcal{A}_{ij} \wedge \mathcal{B}_{ij} \wedge \neg\mathcal{C}_{ij}$ or $\mathcal{A}_{ij} \wedge \mathcal{B}_{ij} \wedge \mathcal{C}_{ij}$). Now assign to i the coordinates (x_i, y_i, z_i) determined by

$$x_i = \max(0, \max_{j \in P_x}(x_j + w_j)) \tag{19}$$

$$y_i = \max(0, \max_{j \in P_y}(y_j + h_j)) \tag{20}$$

$$z_i = \max(0, \max_{j \in P_z}(z_j + d_j)) \tag{21}$$

Once a box has been placed it is inserted into P.

If we maintain a table in which the position of each box i in the three sequences A,B,C is saved, we can test whether $\mathcal{A}_{ij}$, $\mathcal{B}_{ij}$ or $\mathcal{C}_{ij}$ holds in constant time for two given boxes i, j. Since placing a box only requires comparison with every previously placed box, calculating (19) to (21) for a given box i can be done in $O(|P|) = O(n)$ time. Placing all n boxes then requires $O(n^2)$ time.

To speed up the placement procedure slightly we remove a box from P if it is completely "shaded" by a newly inserted box. A box j is shaded by a box i if $x_j + w_j \leq x_i + w_i$, $y_j + h_j \leq y_i + h_i$ and $z_j + d_j < z_i + d_i$, hence it does not affect the placement of future boxes, and by removing it we can avoid subsequent redundant checks.

5.3 Simulated annealing

To solve 3DKP, we use the same Simulated Annealing scheme used for two dimensions but with the three-dimensional sequence representation. The neighborhood is increased to accommodate the extra sequence and consists of the following permutations: 1) exchange two boxes from one of the sequences, 2) exchange two boxes in sequence A and B, 3) exchange two boxes in sequence A and C, 4) exchange two boxes in sequence B and C, 5) exchange two boxes in all sequences.

6 Computational experiments

The heuristic described in the previous sections was implemented in C++ using a modified version of the sequence pair algorithm by Pisinger [30] for two dimensions and an implementation of the placement algorithm for sequence triple described in Section 5.2 for three dimensions. The implementation was tested on a computer with an AMD Athlon 64 3800+ (2.4 GHz) processor with 2 GB of RAM using the GNU-C++ compiler (gcc 4.0). This section is divided into three parts; Section 6.1 considers tighter upper bounds to evaluate the quality of solutions. Section 6.2 deals with the 2DKP and Section 6.3 considers the 3DKP.

6.1 Bounds

To determine the quality of the solutions we compare solution values with upper bound introduced by Fekete et al. [13, 14] which is based on *conservative scales*.

Given an instance I of (3DKP) with container dimensions $W \times H \times D$ a new instance I' with dimensions $1 \times 1 \times 1$ is generated by scaling the dimensions of each item $i \in I$ by $\frac{1}{W}$, $\frac{1}{H}$ and $\frac{1}{D}$, respectively. Define the conservative scale:

$$u^{(k)}(x) = \begin{cases} x & \text{for } (k+1)x \in \mathbb{Z} \\ \lfloor (k+1)x \rfloor \frac{1}{k} & \text{otherwise} \end{cases},$$

and let $u^{(0)}(x) = x$. Now for each $j,k,l = 0,1,\ldots,4$, instances are generated from I' by setting item i's dimensions to $u^{(j)}(w'_i) \times u^{(k)}(h'_i) \times u^{(l)}(d'_i)$. The minimal optimal value of the 1-dimensional relaxation of any of these instances is used as an upper bound of I. For an instance of (2DKP) similar bounds are determined by disregarding the third dimension.

For instances where rotation is allowed the upper bound from conservative scales is not valid, hence we use the weaker upper bound given by the optimal value of (1DKP) defined in (9).

6.2 2D computational experiments

To test the 2DKP heuristic we used both classical instances from the literature and a new set of instances (described in Table 1). The instances were used for parameter tuning of the heuristic. Results are reported for instances both without and with rotation allowed.

6.2.1 Classical instances for 2DKP

We use the benchmarks instances considered by Fekete et al. [14], Caprara and Monaci [8] and Alvarez-Valdes et al. [1, 2]. The instances are listed in Table 2.

The instances `beasley1-12` originate from [6]. The `cgcut` and `gcut` instances are guillotine-cut instances from [9] and [5] respectively. The guillotine-cut instance `wang20` is from [36]. The instances `3` to `CHL5` are also guillotine-cut instances by Hifi [16]. The data for `hadchr3` and `hadchr11` was presented in [15]. The instances `okp1-5` are by Fekete and Schepers [12].

To transform the `gcut` instances, exactly one rectangle was created in the 2DKP instance for each rectangle in the original instance. For the constrained instances b_i duplicates of each rectangle were created, where b_i is the maximum number of times rectangle i may be cut from the material.

In addition to the above instances, Beasley [4] presented a set of 630 instances (`ngcutfs`), which are listed in Table 3. In these, the number of distinct items, M, ranges between 40 and 1000, but items are duplicated Q times, with $Q \in \{1,3,4\}$. Therefore the total number of items, n, ranges from 40 to 4000.

6.2.2 New instances for 2DKP

To test the performance of the heuristic for problems where many rectangles can exist simultaneously in the knapsack, we have created 80 new instances. The rectangle dimensions in each instance belongs to one of five different classes which are listed in Table 1. Dimensions of the items are selected randomly from a uniform distribution between the first and last values of the intervals in the 'Width' and 'Height' columns of the table. The five classes are `tall` (T), `wide` (W), `square` (S), `uniform` (U) and `diverse` (D). The number of rectangles, n, in each instance is selected from the set $\{30,50,100,200\}$. The rectangles may be `clustered` (C) and `random` (R). `Clustered` instances consists of only 20 rectangles which are duplicated appropriately, while in the `random` instances all rectangles are independently generated. Finally the area of the bin is either 25 % or 75 % of the total area of the rectangles and the height of the bin is always twice its width.

Table 1: The five different classes of the new ep2 instances.

Class	Description	Width	Height
T	**Tall.** Rectangles are tall	$[1, \frac{1}{3} \cdot 100]$	$[\frac{2}{3} \cdot 100, 100]$
W	**Wide.** Rectangles are wide	$[\frac{2}{3} \cdot 100, 100]$	$[1, \frac{1}{3} \cdot 100]$
S	**Square.** Rectangles are square	$[1, 100]$	Equal to width
U	**Uniform.** Largest dimension is no more than 150% of the smallest	$[\frac{2}{3} \cdot 100, 100]$	$[\frac{2}{3} \cdot 100, 100]$
D	**Diverse.** Largest dimension can be up-to 100 times the smallest	$[1, 100]$	$[1, 100]$

The naming convention is ep2−n−c−t−p, where $n \in \{30, 50, 100, 200\}$ is the number of rectangles, $c \in (\mathrm{T}, \mathrm{W}, \mathrm{S}, \mathrm{U}, \mathrm{D})$ describes the class, $t \in (\mathrm{C}, \mathrm{R})$ describes if it is clustered or random, $p \in 25, 75$ describes the size of the bin in percentage of the total rectangle area. The profit of the rectangles is always the area of the rectangle times a random number from $\{1, 2, 3\}$. The instances are presented in Table 4 and are available along with the source code to generate them at this web-address: http://www.diku.dk/~pisinger/codes.html.

6.2.3 Parameter setting

Three parameters are crucial for the results of Simulated Annealing: The start time t_0, the time step t_s and the stopping-criteria. A time limit is used as stopping-criteria. Suitable values of t_0 and t_s and the time limit depend on the complexity of the instance.

For each instance we determine two indicator values n_0 and n_1. We set $n_0 = n \cdot \frac{V_{knapsack}}{V_{items}}$, where $V_{knapsack}$ is the area (volume) of the knapsack and V_{items} is sum of the items' area (volume). It indicates the average number of items a knapsack may contain.

We use the value n_0 to determine the running time of our experiments: For an instance with n rectangles and n_0 defined as above let $F(n, n_0) = n_0 \lg n$. If we expect n_0 items to fit into the knapsack and the order of the items matters then there are roughly $\frac{n!}{n_0!} \approx n^{n_0}$ ways to select the items and we may expect that there are roughly n^{n_0} different possible solutions to search. Thus $F(n, n_0) = \lg(n^{n_0})$ should give us a rough indication of the size of the solution space of the instance.

The running time $T(n, n_0)$ (in seconds) of each instance is determined from the value $F(n, n_0)$ as follows

$$T(n, n_0) = \begin{cases} 30 & \text{for} & & F(n, n_0) & < 25 \\ 60 & \text{for} & 25 \leq & F(n, n_0) & < 65 \\ 120 & \text{for} & 65 \leq & F(n, n_0) & < 100 \\ 240 & \text{for} & 100 \leq & F(n, n_0) & < 250 \\ 600 & \text{for} & 250 \leq & F(n, n_0) & \end{cases} \tag{22}$$

The value n_1 is the number of items chosen in an optimal solution of (1DKP) defined in (9). For all considered instances, this problem is solved to optimum very quickly using the exact method by Pisinger [28]. n_1 reflects the number of rectangles to be expected in an optimal solution which is another indicator of the complexity of the instance.

To determine appropriate values of t_0 and t_s we experimented with the 23 instances marked with ''*' in Table 2 and 4. These contain between 16 and 200 rectangles.

We performed the experiments with $t_0 \in \{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}, 10^{4}, 10^{5}\}$ and $t_s \in \{10^{2}, 10^{1}, 10^{-1}, 10^{-3}, 10^{-5}, 10^{-7}, 10^{-9}, 10^{-11}, 10^{-13}\}$. For the 23 instances, each of the 81 combina-

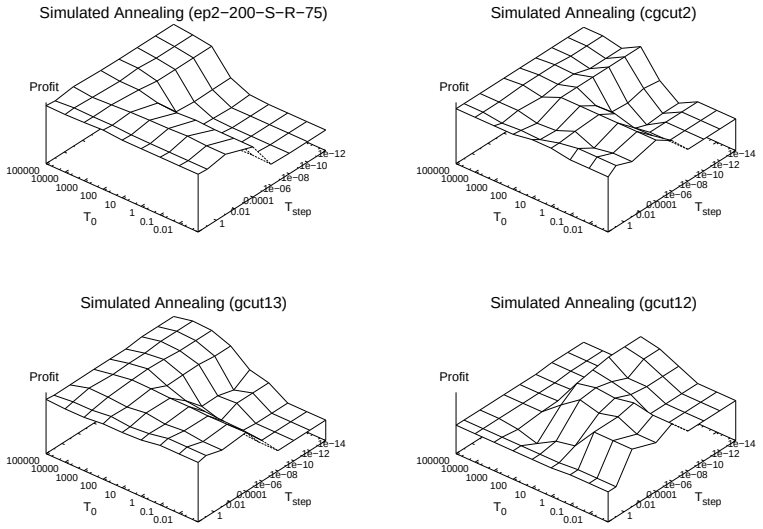


Figure 6: Results of the Simulated Annealing heuristic for different values of t_0 and t_s on four different instances.

tions of t_0 and t_s were tested using the running times from (22). Results from four selected instances are presented in Figure 6.

Based on the results of the parameter tuning for the 23 instances, we were able to establish that good values of t_0 and t_s are:

$$t_0 = n_1^2, \qquad t_s = \frac{n_1^2}{10^7},$$

The values can be interpreted in the following way: The higher t_0 and t_s the less likely is the acceptance of a non-improving permutation. The larger the number of rectangles is in an optimal solution the more improving steps must be undertaken before the heuristic reaches a local minimum.

6.2.4 Results

Using the parameters of the previous section, our heuristic was applied to the instances described in Sections 6.2.1 and 6.2.2. To determine the robustness of the heuristic, we ran it on each instance with 10 different random seeds. For the classical instances, the optimal solution value is reported [14] where known. For the remaining instances, we compare our results with the upper bounds described in section 6.1.

The results are reported in Tables 2–6 which all follow the same format. The average, best and worst results of our heuristic on each instance for the 10 seeds are reported in the columns entitled 'Avg.', 'Best' and 'Worst', respectively. The time before the heuristic discovered the best solution is reported in the column entitled 'Best Time' and the time spent on each seed is reported in the column entitled 'Seed Time' (total time = 10× Seed Time).

In Table 2, we report results for the classical instances. The column 'Optimal' contains values of optimal solutions. For `gcut13`, no optimal value is currently known, but we have reported the results and upper bounds of respectively Caprara and Monaci [8] and Fekete and Schepers [14]. The 'Exact

Methods Time' represent the running time of the algorithms by Caprara and Monaci [8] and Fekete and Schepers [14]. All running times are in seconds. The running times of Alvarez-Valdes et al. [1] are reported in the column entitled 'Alvarez-Valdes' (Instances were solved to optimality). Instances marked with '*' were used for parameter tuning.

The heuristic finds the optimal value in all but four of the instances. For the remaining instances except `hadchr3`, the deviation is less than two percent from optimum. The heuristic is able to improve the best known solution of `gcut13` by 0.8%. The time to find the best solution is generally below one second for the small instances and reaches only a few minutes for the most difficult instances.

For the constrained guillotine-cut instances, we generally get equal or better results than the original authors since we also allow for non-guillotine packing. However, the unconstrained instances (`gcut`) have been transformed and are not comparable with results from the original paper.

Table 3 summarizes results on the 630 `ngcutfs` instances as well as the results of Beasley [4] and Alvarez-Valdes et al. [1, 2]. As in [4] and [1, 2], results are reported as the average *percent* deviation *from* the (1DKP) upper bound in the columns marked 'Avg./1D'. The average time in seconds it took to reach the best solution is given in columns marked 'Time'. The results in 'Beasley' are taken from [4] and the results in 'Alvares-Valdes et al.' are taken from [1].

It is seen that in terms of solution quality, our results are 0.4% better than those of Beasley and only 0.25% worse than Alvarez-Valdes et al. Since their reported results for large instances are found within the first second, we suspect that the approach by Alvarez-Valdes et al. benefits from a greedy constructive algorithm that is applied before their local search, which seems to find optimal solutions for instances with hundreds of items or more. In the present paper, we are mainly interested in comparing the local search frameworks and not the initial greedy heuristics. Moreover, it should be emphasized that although the `ngcutfs` instances contain up-to 4000 items, the average value of n_0 and n_1 are 5.4 and 7.9 and the maximal values are respectively 9.31 and 21. This indicates that the number of rectangles which can fit inside the knapsack is only a few dozen. Therefore it would be interesting to compare results for instances where more than a hundred items fit simultaneously inside the knapsack.

The results on the 80 newly proposed benchmarks are listed in Table 4. Here 'Bound' refers to the upper bound based on conservative scales and 'Best/Bound', 'Avg./Bound', 'Worst/Bound' are the percentage deviations between the heuristic best, average and worst solutions and the conservative scale upper bound calculated as $100 - [\text{solution value}]/[\text{bound value}] \cdot 100$. Only for 11 of the instances is the best result more than 5% from the upper bound and on average the deviation is only 3.0%. The average deviation is 4.6%, 3.1%, 2.4% and 2.1 for $n = 30$, 50, 100, and 200 respectively. For 9 of the instances, the heuristic finds the optimal solution since the deviation from the upper bound is 0. This shows the heuristic's ability to find good solutions for both small and large instances and that the gap between solution value and upper bound decreases as the number of items and size of the knapsack increases.

Figure 7 illustrates the behavior of the heuristic over time for the instances `okp5`, `ep-100-S-R-75` and `ep2-200-U-C-75`. The y-axis is the percentage of the best solution found during the run and the x-axis is the percentage of the full running time for each instance. The graph shows that after rougly half the running time the solution value stays within 1 percent of the best found solution value and that the heuristic quickly converges but allows for minor changes throughout the entire solution process. The best results for three of the instances are shown in Figure 8.

Table 2: Results for the classical benchmark instances.

					Egeblad & Pisinger					Exact Methods Time		Time
Instance	n	n_0	n_1	Optimal	Best	Avg.	Worst	Best Time	Seed Time	Fek-Sch	Cap-Mon	Alvarez-Valdes
beasley1	10	5.3	5	164	164	164	164	≤ 0.02	30	≤ 0.02	-	0
beasley2	17	6.3	8	230	230	230	230	≤ 0.02	60	≤ 0.02	-	0
beasley3	21	7.6	6	247	247	247	247	≤ 0.02	60	≤ 0.02	-	0
beasley4	7	6.5	6	268	268	268	268	≤ 0.02	30	≤ 0.02	-	0
beasley5	14	6	7	358	358	358	358	≤ 0.02	30	≤ 0.02	-	0
beasley6	15	7.8	8	289	289	289	289	≤ 0.02	60	≤ 0.02	-	0
beasley7	8	18.3	8	430	430	430	430	≤ 0.02	30	≤ 0.02	-	0
beasley8	13	8.2	9	834	834	834	834	≤ 0.02	60	≤ 0.02	-	0
beasley9	18	7.4	9	924	924	924	924	0.3	60	≤ 0.02	-	0
beasley10	13	6.8	7	1452	1452	1452	1452	≤ 0.02	60	≤ 0.02	-	0
beasley11	15	9.1	8	1688	1688	1688	1688	≤ 0.02	60	≤ 0.02	-	0
beasley12	22	8.6	11	1865	1865	1865	1865	0.3	60	≤ 0.02	-	0
cgcut1*	16	10.7	8	244	244	244	244	≤ 0.02	60	1.46	0.3	-
cgcut2*	23	14.8	11	2892	2892	2892	2892	1.2	120	531.93	531.93	-
cgcut3*	62	3.9	11	1860	1860	1846	1840	1.6	30	4.58	4.58	-
gcut1*	10	3.82	4	48368	48368	48368	48368	≤ 0.02	30	0.01	≤ 0.02	-
gcut2	20	4.6	5	59798	59798	59704	59563	16.6	30	0.22	0.19	-
gcut3*	30	4.6	6	61275	61275	61275	61275	2.1	30	3.24	2.16	-
gcut4	50	4.3	6	61380	61380	61380	61380	0.9	30	376.52	346.99	-
gcut5	10	4.6	4	195582	195582	195582	195582	≤ 0.02	30	0.5	≤ 0.02	-
gcut6	20	4.1	5	236305	236305	236305	236305	≤ 0.02	30	0.12	0.06	-
gcut7	30	3.7	5	240143	240143	240143	240143	≤ 0.02	30	1.07	0.22	-
gcut8	50	4.5	5	245758	245758	245758	245758	0.1	60	168.5	136.71	-
gcut9	10	4.9	5	939600	939600	939600	939600	≤ 0.02	30	0.08	≤ 0.02	-
gcut10	20	3.7	5	937349	937349	937349	937349	0.6	30	0.14	≤ 0.02	-
gcut11	30	4.6	6	969709	969709	968582.3	958442	≤ 0.02	30	16.3	14.76	-
gcut12*	50	4	4	979521	979521	978727.8	976877	26.2	30	25.39	16.85	-
gcut13*	32	20.1	18	$\geq$8408316 $\geq$8622498 $\leq$9000000	8691947	8637809.1	8615240	239.3	240	1800	1800	-
hadchr3	42	5	4	1178	1086	1086	1086	≤ 0.02	30	≤ 0.02	-	-
hadchr7	7	6.6	5	1865	1865	1865	1865	0.3	60	≤ 0.02	-	-
hadchr8	22	8.6	11	2517	2517	2517	2517	≤ 0.02	30	≤ 0.02	-	-
hadchr11	10	5.6	6	1270	1270	1270	1270	≤ 0.02	30	≤ 0.02	-	-
hadchr12	15	5.3	4	2949	2949	2949	2949	0.4	30	≤ 0.02	-	-
wang20*	15	5.4	3	2726	2716	2712.5	2711	29.2	60	2.72	2.72	0.11
3	62	3.9	11	1860	1860	1846	1840	1.6	30	≤ 0.02	-	-
3s	62	3.9	6	2726	2726	2722.5	2721	8.6	30	≤ 0.02	-	-
a1	62	4.2	11	2020	2020	2004	1960	≤ 0.02	30	≤ 0.02	-	-
a1s	62	4.2	7	2956	2950	2950	2950	18.1	30	≤ 0.02	-	-
a2	53	5.5	11	2615	2615	2594	2545	2.3	30	≤ 0.02	-	-
a2s	53	5.5	7	3535	3535	3517.9	3516	7.6	30	≤ 0.02	-	-
chl2	19	9.1	10	2326	2326	2326	2326	11.9	60	≤ 0.02	-	-
chl2s	19	9.1	9	3336	3336	3336	3336	55.5	60	≤ 0.02	-	-
chl3	35	89.8	35	5283	5283	5283	5283	≤ 0.02	240	≤ 0.02	-	-
chl3s	35	89.8	35	7402	7402	7402	7402	≤ 0.02	240	≤ 0.02	-	-
chl4	27	92.7	27	8998	8858	8763.5	8658	≤ 0.02	240	≤ 0.02	-	-
chl4s	27	92.7	27	13932	13932	13932	13932	≤ 0.02	240	≤ 0.02	-	-
chl5	18	7.4	5	589	589	587	584	1.7	60	≤ 0.02	-	-
okp1*	50	14.3	9	27718	27718	27542.7	27486	6.6	120	35.84	11.6	0.05
okp2	30	9.6	11	22502	22214	22098.6	21947	22.9	60	1559	1535.95	2.14
okp3*	30	8.3	11	24019	24019	23859.8	23531	11	60	10.63	1.91	3.4
okp4	61	10.1	8	32893	32893	32893	32893	4.9	60	4.05	2.13	0.66
okp5*	97	12.6	15	27923	27923	26759	25468	5.4	120	488.27	488.27	≤ 0.02

Table 3: Average results for the 630 `ngcutfs` instances.

Instance	Avg. 1D Deviation			Time		
	Beasley	Egeblad & Pisinger	Alvarez-Valdes et al.	Beasley	Egeblad & Pisinger	Alvarez-Valdes et al.
Type 1	1.64	1.19	0.95	558.1	37.38	19.61
Type 2	1.70	1.29	1.06	668.4	45.03	23.84
Type 3	1.66	1.19	0.94	830.0	62.05	32.56

Table 4: Results for the ep2 instances.

Instance	n	n_0	n_1	Bound	Best	Best Time	Seed Time	Best/Bound	Avg./Bound	Worst/Bound
ep2-30-D-C-25	30	7.5	5	6339	6160	60	6.2	2.82	2.82	2.82
ep2-30-D-C-75*	30	22.2	25	12760	12588	240	12.7	1.35	1.46	1.97
ep2-30-D-R-25	30	7.4	6	6877	6129	60	35.1	10.88	10.88	10.88
ep2-30-D-R-75	30	22.4	24	14395	14259	240	148.8	0.97	1.18	1.53
ep2-30-S-C-25	30	7.4	11	82059	81944	60	0.5	0.14	0.14	0.14
ep2-30-S-C-75	30	22.4	22	198013	195670	240	12.9	1.20	1.37	1.47
ep2-30-S-R-25	30	7.5	9	97151	85220	60	54.0	12.28	14.19	14.40
ep2-30-S-R-75	30	22.5	24	228676	225747	240	174.8	1.28	1.34	1.69
ep2-30-T-C-25	30	7.5	9	30462	22608	60	0.1	25.78	25.78	25.78
ep2-30-T-C-75	30	22.4	22	73944	73565	240	186.8	0.54	0.55	0.61
ep2-30-T-R-25*	30	7.4	8	30570	26034	60	0.2	14.84	14.84	14.84
ep2-30-T-R-75	30	22.3	23	78323	77627	240	206.4	0.89	1.20	1.41
ep2-30-U-C-25	30	7.5	8	143750	133101	60	≤ 0.02	7.74	7.74	7.74
ep2-30-U-C-75	30	22.5	22	354871	343528	240	23.4	3.22	3.22	3.22
ep2-30-U-R-25	30	7.5	8	143127	137739	60	≤ 0.02	3.76	3.76	3.76
ep2-30-U-R-75	30	22.4	22	366621	352982	240	0.5	3.74	3.74	3.74
ep2-30-W-C-25	30	10.6	10	35727	35727	60	11.8	0.00	0.00	0.00
ep2-30-W-C-75	30	22.4	22	46176	46176	240	≤ 0.02	0.00	0.00	0.00
ep2-30-W-R-25	30	10.5	13	34332	34332	60	0.2	0.00	0.00	0.00
ep2-30-W-R-75*	30	22.2	23	45777	45777	240	0.5	0.00	0.00	0.00
ep2-50-D-C-25	50	12.4	11	11094	10369	120	96.8	6.56	6.85	7.21
ep2-50-D-C-75	50	37.1	44	21433	21076	240	127.2	1.70	1.96	2.40
ep2-50-D-R-25	50	12.5	19	12495	11475	120	119.2	8.16	8.82	12.01
ep2-50-D-R-75	50	37.1	38	31657	31426	240	235.6	0.78	1.51	2.11
ep2-50-S-C-25	50	12.5	14	161376	154653	120	24.5	4.21	4.21	4.21
ep2-50-S-C-75	50	37.4	38	391915	387690	240	40.9	1.08	1.75	3.41
ep2-50-S-R-25	50	12.4	12	142758	138263	120	107.4	3.19	3.22	3.27
ep2-50-S-R-75	50	37.3	38	306187	303774	240	234.3	0.82	1.03	1.30
ep2-50-T-C-25*	50	12.4	19	46065	46019	120	12.3	0.10	0.10	0.10
ep2-50-T-C-75	50	37.2	39	118094	114081	240	56.9	3.42	4.25	4.82
ep2-50-T-R-25	50	12.4	12	51175	51175	120	7.7	0.00	0.00	0.00
ep2-50-T-R-75	50	37.2	36	144602	141056	240	138.8	2.45	2.81	4.02
ep2-50-U-C-25	50	12.5	13	242937	231774	120	4.8	4.60	4.79	4.87
ep2-50-U-C-75	50	37.4	37	632455	619080	240	207.4	2.17	2.47	3.64
ep2-50-U-R-25	50	12.4	12	263251	227979	120	8.5	13.40	13.75	13.90
ep2-50-U-R-75*	50	37.4	38	576134	564394	240	235.8	2.07	2.29	3.54
ep2-50-W-C-25*	50	13.7	15	50130	50130	120	18.1	0.00	0.00	0.00
ep2-50-W-C-75	50	37.3	37	94279	87320	240	21.0	7.38	7.38	7.38
ep2-50-W-R-25	50	12.9	13	55920	55920	120	0.5	0.00	0.00	0.00
ep2-50-W-R-75	50	37.4	35	115156	114656	240	97.0	0.49	0.95	1.22
ep2-100-D-C-25	100	24.5	19	23250	22730	240	225.0	2.25	2.80	3.47
ep2-100-D-C-75	100	74.0	77	51241	49732	600	536.7	2.99	3.85	4.99
ep2-100-D-R-25	100	24.7	23	22326	22133	240	237.9	0.92	1.31	2.14
ep2-100-D-R-75	100	74.6	73	51231	50874	600	520.3	0.73	1.11	2.08
ep2-100-S-C-25	100	24.9	28	323640	314198	240	39.2	2.98	3.57	4.15
ep2-100-S-C-75	100	74.9	81	756554	747129	600	580.4	1.29	1.63	2.49
ep2-100-S-R-25	100	24.9	27	254616	250242	240	210.3	1.77	2.69	3.70
ep2-100-S-R-75*	100	74.9	80	523573	519787	600	595.6	0.76	1.26	1.68
ep2-100-T-C-25*	100	24.9	27	92331	92331	240	28.2	0.00	0.00	0.00
ep2-100-T-C-75	100	74.7	69	265970	256480	600	316.4	3.65	4.06	4.96
ep2-100-T-R-25	100	24.8	25	103359	102837	240	34.3	0.56	0.78	1.05
ep2-100-T-R-75	100	74.8	77	262492	257927	600	573.4	1.78	2.39	3.49
ep2-100-U-C-25	100	25.0	25	547224	505794	240	42.2	7.63	8.70	9.77
ep2-100-U-C-75	100	75.0	75	1433510	1400642	600	14.0	2.29	2.29	2.29
ep2-100-U-R-25	100	25.0	26	518661	493901	240	157.0	4.77	5.28	5.51
ep2-100-U-R-75	100	74.9	75	1216431	1203544	600	418.4	1.10	1.26	1.74
ep2-100-W-C-25	100	24.9	29	70437	70164	240	4.9	0.39	0.71	1.06
ep2-100-W-C-75	100	74.9	88	167577	159929	600	450.9	4.56	4.88	6.38
ep2-100-W-R-25*	100	25.0	27	70224	70224	240	165.3	0.00	0.04	0.12
ep2-100-W-R-75	100	74.7	77	247494	230809	600	393.8	6.80	7.40	8.33
ep2-200-D-C-25*	200	49.2	63	46728	45987	600	530.4	1.73	2.55	3.99
ep2-200-D-C-75	200	149.7	160	127834	124146	600	536.5	2.99	3.87	4.73
ep2-200-D-R-25	200	49.6	57	43605	42138	600	591.4	3.46	4.83	5.80
ep2-200-D-R-75	200	149.3	154	99002	97694	600	450.6	1.39	2.19	3.47
ep2-200-S-C-25	200	49.9	54	649446	636050	600	39.9	2.15	2.78	3.43
ep2-200-S-C-75	200	149.8	155	1315780	1297053	600	570.2	1.49	2.29	3.08
ep2-200-S-R-25	200	49.8	48	519498	506437	600	533.1	2.59	5.30	7.48
ep2-200-S-R-75*	200	149.7	143	1225926	1201303	600	581.5	2.06	3.10	3.75
ep2-200-T-C-25*	200	50.0	41	188684	184528	600	319.9	2.26	3.48	3.84
ep2-200-T-C-75	200	149.5	152	441796	431290	600	558.1	2.44	3.35	4.57
ep2-200-T-R-25	200	49.7	50	190638	188967	600	587.1	0.98	1.23	1.53
ep2-200-T-R-75	200	149.5	145	476289	468381	600	597.6	1.72	2.67	3.64
ep2-200-U-C-25	200	50.0	51	1084836	1056636	600	512.6	2.84	2.96	2.97
ep2-200-U-C-75	200	149.8	151	2313551	2265176	600	549.6	2.15	2.58	2.78
ep2-200-U-R-25	200	49.9	49	1039584	1015452	600	587.0	2.39	3.70	5.05
ep2-200-U-R-75	200	149.7	149	2447655	2400803	600	176.6	1.97	2.14	2.65
ep2-200-W-C-25	200	49.6	76	161002	157508	600	577.0	2.21	2.97	3.83
ep2-200-W-C-75	200	149.4	160	390001	375767	600	566.1	3.78	4.26	5.10
ep2-200-W-R-25	200	49.9	50	196128	196086	600	170.7	0.02	0.14	0.26
ep2-200-W-R-75	200	149.6	146	511386	503222	600	586.7	1.69	2.24	2.79

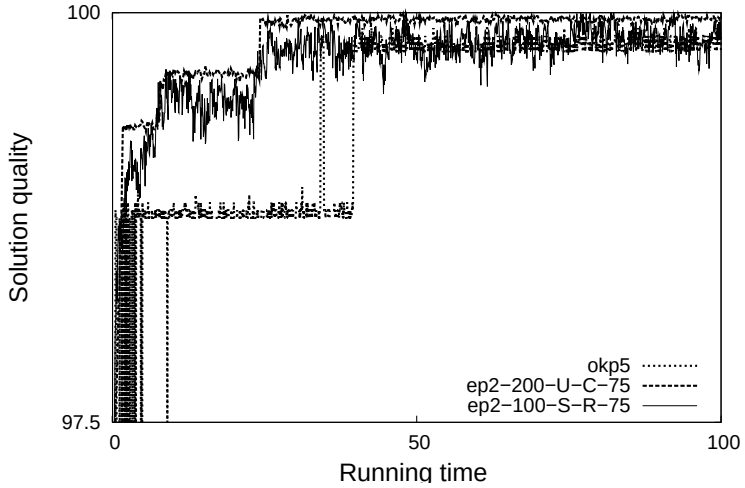


Figure 7: Illustration of the heuristic behavior over time.

6.2.5 Results with Rotations

We repeated all tests allowing rotation, doubling the running time to accommodate for the larger solution space. A maximum time limit of 600 seconds was still assigned to all instances. Parameter-tuning revealed that the setting of t_0 and t_s reported in Section 6.2.3 also give good results when rotation is allowed.

The results on the two sets of instances are reported in Table 5 and Table 6. The tables follow the same format as the tables without rotation, however the column 'No rotation' is the value of the optimal solution without rotation for the classical instances and the best solution without rotation for the `ep2` instances where the optimal solutions are not known. For `gcut13` no optimal values without rotation is know and we have reported the best solution from Table 2. To compare the quality of the solution we used the (1DKP) upper bound, which is weaker than the bound from conservative scales. The column 'No rotation/1D' contains percentage deviations between the result without rotation and the (1DKP) upper bound and 'Best/1D', 'Avg./1D', 'Worst/1D' are deviation between best, average and worst results over the 10 runs and the (1DKP) upper bound.

With the exception of two instances the results with rotation for the classical benchmark instances are better or equal to the results without rotation. This was expected since rotations make it possible to arrange the rectangles in more ways, and hence denser packings can be obtained. Indeed, the heuristic solution is generally larger than 95% of the (1DKP) upper bound, and often reaches 98% when rotation is allowed.

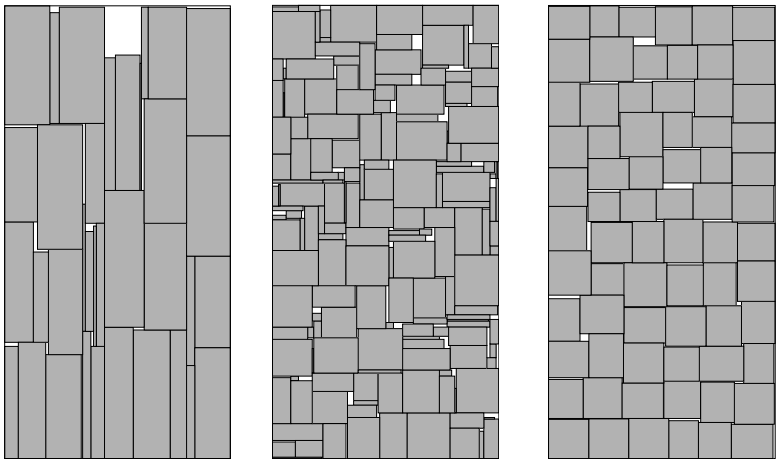
For the 28 of the `ep2` instances, we get results which are more than 2% closer to the upper bound than without rotation. We get slightly worse results when allowing rotation in 15 of the 80 cases but only for 3 instances is the result more than 2% further from the upper bound. This demonstrates the heuristics ability to handle rotation well, even though the solution space is increased by a factor of 2^n which makes it much harder to find near-optimal solutions. The average deviation between the best result and the (1DKP) upper bound for the `ep2` instances is 2.3% which means the heuristic performs well even with rotation for large instances. It also shows that utilization increases by a few percent if rotation is allowed.

Table 5: Results with rotation for the benchmark instances.

Instance	1D	No rotation	Best	Best Time	Seed Time	No rotation/1D	Best/1D	Avg./1D	Worst/1D
beasley1	201	164	193	≤ 0.02	60	18.41	3.98	3.98	3.98
beasley2	253	230	250	0.3	120	9.09	1.19	1.19	1.19
beasley3	266	247	259	0.4	120	7.14	2.63	2.63	2.63
beasley4	275	268	268	≤ 0.02	60	2.55	2.55	2.55	2.55
beasley5	373	358	370	≤ 0.02	60	4.02	0.80	0.80	0.80
beasley6	317	289	300	23.9	120	8.83	5.36	5.99	5.99
beasley7	430	430	430	≤ 0.02	60	0.00	0.00	0.00	0.00
beasley8	938	834	886	0.2	120	11.09	5.54	5.54	5.54
beasley9	962	924	924	4.4	120	3.95	3.95	4.57	4.57
beasley10	1517	1452	1452	≤ 0.02	120	4.28	4.28	4.28	4.28
beasley11	1864	1688	1786	≤ 0.02	120	9.44	4.18	4.18	4.18
beasley12	2012	1865	1932	1.8	120	7.31	3.98	4.47	4.47
cgcut1	260	244	260	0.7	120	6.15	0.00	0.00	0.00
cgcut2	2919	2892	2909	115.3	240	0.92	0.34	0.34	0.34
cgcut3	2020	1860	1940	6.3	60	7.92	3.96	3.96	3.96
gcut1	62488	48368	58136	≤ 0.02	60	22.60	6.96	6.96	6.96
gcut2	62500	59798	60656	23.9	60	4.32	2.95	3.22	3.31
gcut3	62500	61275	61275	49.6	60	1.96	1.96	2.26	2.92
gcut4	62500	61380	61710	30.1	60	1.79	1.26	1.28	1.47
gcut5	249854	195582	233969	1.1	60	21.72	6.36	6.36	6.36
gcut6	249992	236305	239467	0.1	60	5.47	4.21	4.21	4.21
gcut7	249998	240143	245306	34.5	60	3.94	1.88	2.64	2.83
gcut8	250000	245758	247462	21	120	1.70	1.02	1.12	1.26
gcut9	997256	939600	953293	≤ 0.02	60	5.78	4.41	4.41	4.41
gcut10	999918	937349	938036	0.1	60	6.26	6.19	6.19	6.19
gcut11	1000000	969709	979580	20	60	3.03	2.04	2.46	2.96
gcut12	1000000	979521	987674	8.9	60	2.05	1.23	1.23	1.23
gcut13	9000000	≥ 8736757	8897979	344.5	480	2.92	1.13	1.64	2.32
hadchr-3	1347	1178	1272	≤ 0.02	60	12.55	5.57	5.57	5.57
hadchr-7	2012	1865	1932	9.7	120	7.31	3.98	4.03	4.47
hadchr-8	3079	2517	2722	≤ 0.02	60	18.25	11.59	11.59	11.59
hadchr-11	1547	1270	1431	1	60	17.91	7.50	7.50	7.50
hadchr-12	3604	2949	3252	9.8	60	18.17	9.77	9.77	9.77
wang20	2800	2771	2762	118	120	1.04	1.36	1.50	1.50
3	2020	1860	1940	0.9	60	7.92	3.96	3.96	3.96
3s	2800	2726	2758	12.6	60	2.64	1.50	1.54	1.57
a1	2140	2020	2120	40.2	60	5.61	0.93	1.78	1.87
a1s	3000	2956	2985	1.7	60	1.47	0.50	0.50	0.50
a2	2705	2615	2690	18.4	120	3.33	0.55	0.55	0.55
a2s	3600	3335	3579	99.1	120	7.36	0.58	0.58	0.58
chl2	2502	2326	2429	8.9	120	7.03	2.92	3.20	4.32
chl2s	3410	3336	3390	7.8	120	2.17	0.59	0.59	0.59
chl3	5283	5283	5283	≤ 0.02	480	0.00	0.00	0.00	0.00
chl3s	7402	7402	7402	≤ 0.02	480	0.00	0.00	0.00	0.00
chl4	8998	8998	8912	≤ 0.02	480	0.00	0.96	1.77	2.98
chl4s	13932	13932	13932	≤ 0.02	480	0.00	0.00	0.00	0.00
chl5	600	589	600	7.6	120	1.83	0.00	0.00	0.00
okp1	29133	27718	28423	41.7	240	4.86	2.44	4.24	5.47
okp2	24800	22502	24263	1.6	120	9.27	2.17	5.19	8.05
okp3	26714	24019	25216	7.2	120	10.09	5.61	7.28	10.99
okp4	33631	32893	32893	19.7	120	2.19	2.19	2.19	2.19
okp5	29045	27923	27971	121	240	3.86	3.70	6.06	11.43

Table 6: Results with rotation for the ep2 instances.

Instance	Best	Best time	Seed Time	No rotation/1D	Best/1D	Avg./1D	Worst/1D
ep2-30-D-C-25	6340	57.4	120	4.23	1.43	1.52	1.55
ep2-30-D-C-75	12672	347.5	480	1.35	0.69	1.20	1.50
ep2-30-D-R-25	6875	113.6	120	16.54	6.39	6.58	6.66
ep2-30-D-R-75	14330	115.3	480	0.94	0.45	0.62	0.93
ep2-30-S-C-25	81944	1.8	120	6.04	6.04	6.04	6.04
ep2-30-S-C-75	195670	30.4	480	1.18	1.18	1.56	2.20
ep2-30-S-R-25	83160	1.4	120	12.28	14.40	14.40	14.40
ep2-30-S-R-75	225747	127.2	480	1.28	1.28	1.40	2.02
ep2-30-T-C-25	26436	0.5	120	29.29	17.32	17.87	18.52
ep2-30-T-C-75	73565	144.4	480	0.51	0.51	1.04	1.50
ep2-30-T-R-25	29535	3.3	120	16.28	5.02	5.02	5.02
ep2-30-T-R-75	77387	77.6	480	0.89	1.20	1.80	2.49
ep2-30-U-C-25	141588	0.2	120	7.41	1.50	1.50	1.50
ep2-30-U-C-75	351682	204.7	480	3.20	0.90	0.90	0.90
ep2-30-U-R-25	140287	22.6	120	7.51	5.80	5.80	5.80
ep2-30-U-R-75	360202	450.9	480	3.72	1.75	1.75	1.75
ep2-30-W-C-25	37851	75.6	120	11.44	6.17	6.33	6.34
ep2-30-W-C-75	53668	174.6	480	14.33	0.43	0.83	1.37
ep2-30-W-R-25	37258	32.5	120	9.43	1.71	2.29	2.85
ep2-30-W-R-75	52895	254.1	480	14.61	1.33	1.80	2.70
ep2-50-D-C-25	11010	90.0	240	6.54	0.76	1.27	1.66
ep2-50-D-C-75	21230	278.8	480	1.67	0.95	1.56	2.05
ep2-50-D-R-25	12456	209.6	240	8.81	1.02	2.78	5.65
ep2-50-D-R-75	31531	299.1	480	0.73	0.40	1.22	1.84
ep2-50-S-C-25	154653	9.3	240	4.17	4.17	4.20	4.43
ep2-50-S-C-75	387526	285.3	480	1.08	1.12	2.18	3.41
ep2-50-S-R-25	138335	235.4	240	3.15	3.10	3.18	3.53
ep2-50-S-R-75	303605	313.8	480	0.79	0.84	1.23	1.78
ep2-50-T-C-25	46327	91.8	240	2.26	1.61	1.88	3.42
ep2-50-T-C-75	117226	303.3	480	3.40	0.74	1.01	1.23
ep2-50-T-R-25	53383	172.2	240	5.99	1.93	2.10	2.28
ep2-50-T-R-75	142124	388.1	480	2.45	1.71	2.23	3.56
ep2-50-U-C-25	246645	0.1	240	14.86	9.39	9.39	9.39
ep2-50-U-C-75	626218	365.4	480	2.11	0.99	1.46	2.11
ep2-50-U-R-25	241032	26.6	240	14.39	9.49	9.49	9.49
ep2-50-U-R-75	570751	398.3	480	2.04	0.93	1.71	1.80
ep2-50-W-C-25	51360	22.2	240	4.21	1.86	1.86	1.86
ep2-50-W-C-75	111307	354.7	480	22.17	0.79	1.21	2.05
ep2-50-W-R-25	57561	102.5	240	3.92	1.10	1.25	1.26
ep2-50-W-R-75	113868	452.1	480	0.43	1.12	1.55	2.18
ep2-100-D-C-25	23220	415.3	480	2.24	0.13	0.26	0.71
ep2-100-D-C-75	50515	378.0	600	2.94	1.42	2.67	4.17
ep2-100-D-R-25	22281	51.7	480	0.86	0.20	0.42	0.64
ep2-100-D-R-75	50772	410.8	600	0.70	0.90	1.78	2.81
ep2-100-S-C-25	316238	341.7	480	2.92	2.29	3.37	4.36
ep2-100-S-C-75	745608	354.1	600	1.25	1.45	1.91	2.45
ep2-100-S-R-25	251303	343.7	480	1.72	1.30	2.30	3.79
ep2-100-S-R-75	515379	509.9	600	0.72	1.57	2.02	3.96
ep2-100-T-C-25	99429	98.6	480	8.94	1.94	3.31	6.41
ep2-100-T-C-75	260968	574.9	600	3.57	1.88	2.94	3.77
ep2-100-T-R-25	102957	303.1	480	0.51	0.39	1.06	2.93
ep2-100-T-R-75	259019	497.6	600	1.74	1.32	2.30	3.44
ep2-100-U-C-25	531585	192.6	480	7.57	2.86	5.11	6.17
ep2-100-U-C-75	1400642	147.6	600	2.29	2.29	2.37	3.06
ep2-100-U-R-25	504801	48.2	480	4.77	2.67	4.27	4.95
ep2-100-U-R-75	1203634	576.3	600	1.06	1.05	1.63	1.69
ep2-100-W-C-25	94098	205.3	480	26.63	1.61	2.57	4.24
ep2-100-W-C-75	165709	599.5	600	6.77	3.40	3.69	5.00
ep2-100-W-R-25	103431	444.4	480	32.83	1.06	2.83	4.78
ep2-100-W-R-75	242357	356.2	600	6.74	2.08	2.87	3.49
ep2-200-D-C-25	46056	473.0	600	1.59	1.44	2.32	3.09
ep2-200-D-C-75	123724	561.3	600	2.88	3.22	3.56	4.07
ep2-200-D-R-25	42844	572.0	600	3.36	1.75	3.62	5.14
ep2-200-D-R-75	96344	593.3	600	1.32	2.68	3.64	5.88
ep2-200-S-C-25	637579	333.6	600	2.06	1.83	2.86	3.45
ep2-200-S-C-75	1292256	596.0	600	1.42	1.79	2.52	4.30
ep2-200-S-R-25	505687	589.1	600	2.51	2.66	4.78	7.11
ep2-200-S-R-75	1198449	576.0	600	2.01	2.24	4.33	6.11
ep2-200-T-C-25	186248	77.0	600	2.20	1.29	2.32	3.81
ep2-200-T-C-75	432747	563.5	600	2.38	2.05	2.66	3.54
ep2-200-T-R-25	189141	577.5	600	0.88	0.79	1.43	2.61
ep2-200-T-R-75	465169	583.1	600	1.66	2.33	3.67	4.86
ep2-200-U-C-25	1073880	238.0	600	2.60	1.01	2.04	2.73
ep2-200-U-C-75	2265346	570.2	600	2.09	2.08	2.34	2.71
ep2-200-U-R-25	1035978	422.7	600	2.32	0.35	1.95	4.37
ep2-200-U-R-75	2406562	591.2	600	1.91	1.68	2.06	2.25
ep2-200-W-C-25	158468	506.2	600	2.17	1.57	2.36	3.75
ep2-200-W-C-75	382662	587.2	600	3.65	1.88	3.85	6.30
ep2-200-W-R-25	210864	575.9	600	8.51	1.62	2.10	2.50
ep2-200-W-R-75	501738	599.7	600	1.60	1.89	3.51	4.52



ep2-50-T-R-75 ep2-200-D-R-75 ep2-100-U-R-75

Figure 8: Best results for three two-dimensional instances without rotation.

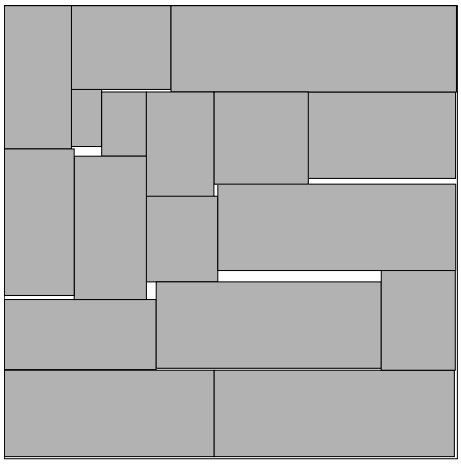


Figure 9: A solution to `gcut13` with profit 8736757 which was reached after 144 10 minute runs.

6.2.6 The instance `gcut13`

Since `gcut13` is the only instance of the classical benchmark instances from the literature where the optimal solution is unknown, we decided to investigate this instance further without considering rotations. We used 144 seeds with 10 minutes running time on each seed; thus the total running time was 24 hours for this instance. The parameters for the Simulated Annealing were based on the results we gathered during parameter-tuning for `gcut13`, and were set to $t_0 = 0.1$ and $t_s = 10$. The best result was reached after 367 seconds for one of the seeds and was 8736757 which is 0.5% closer to the upper bound than the 8691947 we were able to reach with 10 seeds and 4 minutes running time. This demonstrates that the heuristic is able to return marginally improved results given more running time. The resulting placement can be seen in Figure 9.

Table 7: The 5 different classes of new `ep3` instances.

Class	Description	Width	Height	Depth
F	**Flat.** Boxes are flat	$[50,100]$	$[50,100]$	$[25,60]$
W	**Long.** Boxes are long	$[1,\frac{2}{3}\cdot 100]$	$[1,\frac{2}{3}\cdot 100]$	$[50,100]$
S	**Cubes.** Boxes are cubes	$[1,100]$	Equal to width	Equal to width
U	**Uniform.** Largest dimension is no more than 200% of the smallest	$[50,100]$	$[50,100]$	$[50,100]$
D	**Diverse.** Largest dimension can be up-to 50 times the smallest	$[1,50]$	$[1,50]$	$[1,50]$

6.3 3D computational experiments

A set of experiments were also conducted using the three-dimensional variant of our heuristic which follows the same scheme as the experiments conducted for the two-dimensional variant. New instances for 3DKP are introduced in Section 6.3.1, the parameter-tuning is outlined in Section 6.3.2 and results are presented in Section 6.3.3.

6.3.1 New instances

As we were unable to locate any benchmark instances for the three-dimensional knapsack problem from the literature, we have generated 60 random instances. It should be noted that Fekete et al. [14] do report results for a number of 3DKP problem instances, but the instances are not described in detail. The new instances contain 20, 40 or 60 boxes. The dimensions of the boxes were chosen from 5 different classes described in Table 7. The width, height and depth of the boxes in each class are selected randomly from the intervals in the 'Width', 'Height' and 'Depth' columns of the table. As for the two-dimensional case, boxes are clustered and random, and the container has a volume equal to 50% or 90% of the total volume of the boxes. The naming convention is `ep3`$-n-c-t-p$, where $n \in \{20,40,60\}$ is the number of boxes, $c \in (\mathrm{F},\mathrm{L},\mathrm{C},\mathrm{U},\mathrm{D})$ describes the class, $t \in (\mathrm{C},\mathrm{R})$ describes if it is clustered or random, $p \in \{50,90\}$ describes the size of the bin in percentage of the total box volume. The profit of a box is set to the volume times a random number from $\{1,2,3\}$. The instances are presented in Table 8 and are available along with the source code to generate them at this web-address: `http://www.diku.dk/~pisinger/codes.html`.

6.3.2 Parameters

As for the two-dimensional instances we determine values n_0 and n_1 and for each instance we set the running time based on the function $F(n,n_0) = n_0 \lg n$, so that for $F(n,n_0) \leq 110$ the running time is set to 120 seconds, for $110 < F(n,n_0) \leq 200$ the running time is set to 300 seconds, and for $F(n,n_0) > 200$ the running time is set to 600 seconds.

Nine three-dimensional instances were selected for parameter tuning tests. Values for t_0 was selected from $\{10^0, 10^1, 10^2, 10^3, 10^4, 10^5, 10^7, 10^8\}$ and t_s from $\{10^{-10}, 10^{-8}, 10^{-6}, 10^{-4}, 10^{-1}, 10^0, 10^2, 10^4, 10^6\}$. Based on the 81 parameter combinations we found results similar to the two-dimensional case, and based on these we determined good values to be

$$t_0 = \frac{n_1^2}{5}, \text{and } t_s = n_1^2.$$

6.3.3 Results

The results from our 3D tests are presented in Table 8 using the same format as the tables of the two-dimensional results.

For the instances with 20 items, the gap between the best found solution and the conservative scales upper bound is as large as 38.7% for one instance but only 13.6% on average and in two cases the heuristic is able to reach the upper bound thus finding optimal solutions. For the instances with 40 items, 16 of the best solutions are within 15% of the upper bound and as much as 7 are within 10%. The average gap between best solutions and upper bound is 12.53%. For the instances with 60 items, the heuristic reaches best solutions which have a gap which is less than 20% to the upper bound for all but one of the instances. For 8 of the instances, the gap is less than 10% and the average gap is as low as 11.4%. The best results for eight of the instances are shown in Figure 10.

Our method finds high quality solutions quickly with an average gap to the upper bound of only 12.8%. In most of the instances, the best solution is found long before the heuristic's time limit, so solutions may be significantly closer to the optimal value than the bounds indicate. Results with large gaps could be due to the geometry of the boxes which can make it hard to utilize the three-dimensional knapsack as well as in the one-dimensional problems. Another explanation could be that the conservative scales bound does not function well on those instances or that the heuristic simply performs poorly in some cases.

To the best of our knowledge no other authors report gaps to upper bound for the three-dimensional knapsack packing problem. Fekete et al. [14] report only the number of problems solved to optimality. Their items are also larger than ours and we suspect that far fewer may be loaded in the knapsack than in our instances. This reduces the solution space and increases the strength of the bounds, making the instances easier to solve than the instances considered herein.

Instances for the Container Loading Problem contain hundreds of items, and state-of-the-art heuristics (e.g. [24, 27, 29]) reach volume utilization of slightly more than 90% on average. Initial experiments with our heuristic for container loading instances resulted in solutions with utilization of around 84% within 1 minute.

Container loading heuristics however only optimize volume utilization, and hence cannot handle a general profit objective. Moreover, they exploit that most container loading instances contain many similar items. The strength of our heuristic is that it is not based on assumptions on the item sizes or profit. Based on our experiments we believe our heuristic is most suitable for medium sized instances where less than 80 items can fit in the knapsack simultaneously. We also suspect that the proposed heuristic for 3DKP cannot compete with heuristics tailored specifically for the Container Loading Problem.

7 Conclusion

In this paper, we have presented simulated annealing based approaches for the two- and three-dimensional knapsack problem. For the two-dimensional knapsack problem, we utilize an abstract representation for rectangle packings called sequence pair whereas for the three-dimensional problem we utilize a novel abstract representation for box packings called sequence triple. We have proved that the sequence triple is able to represent any fully robot packable packing.

The heuristic for two dimensions is generally able to reproduce the results of exact algorithms with similar running times. The heuristic also gives the best known results for the only unsolved classical-instance; `gcut13`. To demonstrate the high quality of the results of the heuristic for larger instances we have created a new set of instances with up-to 200 rectangles and also here the heuristic

Table 8: Results for the new `ep3` instances.

Instance	n	n_0	n_1	1D	Bound	Best	Best time	Seed time	Best/Bound	Avg./Bound	Worst/Bound
ep3-20-C-C-50.3kp	20	9.9	13	125170	65308	65308	≤ 0.02	120	0.00	0.00	0.00
ep3-20-C-C-90.3kp	20	17.8	19	155222	97422	80124	≤ 0.02	120	17.76	17.76	17.76
ep3-20-C-R-50.3kp	20	9.9	15	108421	63849	62364	≤ 0.02	120	2.33	2.33	2.33
ep3-20-C-R-90.3kp	20	17.8	19	118927	80956	66844	≤ 0.02	120	17.43	17.43	17.43
ep3-20-D-C-50.3kp	20	9.8	13	21792	13192	13192	≤ 0.02	120	0.00	0.00	0.00
ep3-20-D-C-90.3kp	20	17.6	17	38842	30728	27848	0.1	120	9.37	9.37	9.37
ep3-20-D-R-50.3kp	20	10.0	12	19298	17845	15170	23.6	120	14.99	16.26	18.08
ep3-20-D-R-90.3kp	20	17.8	18	25676	24836	20822	3.1	120	16.16	16.16	16.16
ep3-20-F-C-50.3kp	20	9.9	13	114718	89771	71816	≤ 0.02	120	20.00	20.00	20.00
ep3-20-F-C-90.3kp	20	17.8	17	246440	184773	167281	0.6	120	9.47	12.46	13.75
ep3-20-F-R-50.3kp	20	9.9	11	128928	85712	80962	0.1	120	5.54	5.54	5.54
ep3-20-F-R-90.3kp	20	18.0	18	191008	172182	155155	0.4	120	9.89	10.35	12.03
ep3-20-L-C-50.3kp	20	10.0	8	33780	28716	20835	23.0	120	27.44	27.65	27.67
ep3-20-L-C-90.3kp	20	17.8	18	63772	51988	48404	≤ 0.02	120	6.89	6.89	6.89
ep3-20-L-R-50.3kp	20	9.9	10	31415	30037	24809	0.6	120	17.41	17.55	18.15
ep3-20-L-R-90.3kp	20	18.0	17	37570	37832	32892	29.8	120	13.06	14.98	16.80
ep3-20-U-C-50.3kp	20	9.9	10	154428	148412	91036	26.6	120	38.66	41.08	42.11
ep3-20-U-C-90.3kp	20	17.9	17	162251	152837	132291	0.2	120	13.44	16.15	17.04
ep3-20-U-R-50.3kp	20	9.9	10	140762	123675	97358	0.2	120	21.28	22.66	28.45
ep3-20-U-R-90.3kp	20	17.9	18	211545	209795	188691	26.3	120	10.06	10.06	10.06
ep3-40-C-C-50.3kp	40	19.7	29	224754	170318	141418	0.1	120	16.97	16.97	16.97
ep3-40-C-C-90.3kp	40	35.7	33	358993	252425	243447	0.6	300	3.56	3.56	3.56
ep3-40-C-R-50.3kp	40	19.8	22	202725	145067	126069	0.6	120	13.10	13.10	13.10
ep3-40-C-R-90.3kp	40	35.9	38	245888	226890	218971	11.0	300	3.49	3.49	3.49
ep3-40-D-C-50.3kp	40	19.9	22	34128	32520	20464	0.1	120	37.07	40.42	46.33
ep3-40-D-C-90.3kp	40	35.6	30	70302	63600	56124	113.9	300	11.75	14.08	14.34
ep3-40-D-R-50.3kp	40	19.2	24	35001	35803	28019	36.2	120	21.74	24.17	26.14
ep3-40-D-R-90.3kp	40	35.7	36	52769	53698	49476	244.0	300	7.86	9.11	9.81
ep3-40-F-C-50.3kp	40	19.9	19	225834	180302	161310	23.3	120	10.53	10.53	10.53
ep3-40-F-C-90.3kp	40	35.8	33	543292	473556	416467	3.4	300	12.06	13.36	14.06
ep3-40-F-R-50.3kp	40	19.7	21	312163	277727	238990	93.7	120	13.95	17.36	22.64
ep3-40-F-R-90.3kp	40	35.7	37	396059	400886	367045	282.3	300	8.44	8.78	9.60
ep3-40-L-C-50.3kp	40	19.6	21	80264	61222	56742	76.9	120	7.32	8.63	8.78
ep3-40-L-C-90.3kp	40	35.5	34	117699	115632	106602	223.6	300	7.81	8.71	9.90
ep3-40-L-R-50.3kp	40	19.9	21	61275	62931	50389	80.9	120	19.93	21.93	26.22
ep3-40-L-R-90.3kp	40	35.6	35	85517	87114	77514	235.2	300	11.02	11.97	13.10
ep3-40-U-C-50.3kp	40	20.0	20	216956	185028	164976	0.4	120	10.84	11.87	14.27
ep3-40-U-C-90.3kp	40	35.9	36	461920	442120	388456	78.5	300	12.14	12.14	12.14
ep3-40-U-R-50.3kp	40	19.8	20	271163	276228	242819	12.0	120	12.09	12.86	14.07
ep3-40-U-R-90.3kp	40	35.9	37	401482	408182	371662	286.9	300	8.95	9.58	10.52
ep3-60-C-C-50.3kp	60	29.7	37	363614	373341	252584	3.1	300	32.34	34.40	34.98
ep3-60-C-C-90.3kp	60	53.7	57	620658	534168	499572	92.8	600	6.48	6.48	6.48
ep3-60-C-R-50.3kp	60	29.8	44	303842	229567	224875	273.2	300	2.04	4.57	16.73
ep3-60-C-R-90.3kp	60	53.7	58	427077	415339	384302	49.4	600	7.47	7.80	7.97
ep3-60-D-C-50.3kp	60	29.2	38	90084	75732	65502	63.4	300	13.51	13.57	13.63
ep3-60-D-C-90.3kp	60	53.4	55	129504	115968	105540	450.7	600	8.99	9.01	9.02
ep3-60-D-R-50.3kp	60	29.4	41	56227	58185	49011	227.3	300	15.77	18.60	20.39
ep3-60-D-R-90.3kp	60	53.4	56	73109	74464	67202	337.1	600	9.75	10.21	10.70
ep3-60-F-C-50.3kp	60	29.7	40	522640	517890	467730	3.9	300	9.69	9.69	9.69
ep3-60-F-C-90.3kp	60	53.3	50	642192	616504	540732	145.3	600	12.29	12.92	14.60
ep3-60-F-R-50.3kp	60	29.6	30	437008	442187	378079	138.6	300	14.50	15.66	18.52
ep3-60-F-R-90.3kp	60	53.6	55	668571	685430	625362	413.5	600	8.76	9.65	10.02
ep3-60-L-C-50.3kp	60	29.6	34	131466	127740	119676	229.3	300	6.31	9.64	11.53
ep3-60-L-C-90.3kp	60	53.5	52	124652	127340	113107	226.8	600	11.18	12.09	13.08
ep3-60-L-R-50.3kp	60	29.3	29	74166	76856	62190	297.8	300	19.08	21.03	23.27
ep3-60-L-R-90.3kp	60	53.3	54	117361	120313	107270	373.4	600	10.84	11.67	12.06
ep3-60-U-C-50.3kp	60	29.6	34	430130	441418	361340	42.6	300	18.14	18.83	20.13
ep3-60-U-C-90.3kp	60	54.0	54	324792	343738	293852	181.7	600	14.51	15.57	16.70
ep3-60-U-R-50.3kp	60	29.7	31	452615	465935	396794	281.2	300	14.84	16.07	17.17
ep3-60-U-R-90.3kp	60	53.6	55	604852	618814	550375	531.6	600	11.06	11.67	12.17

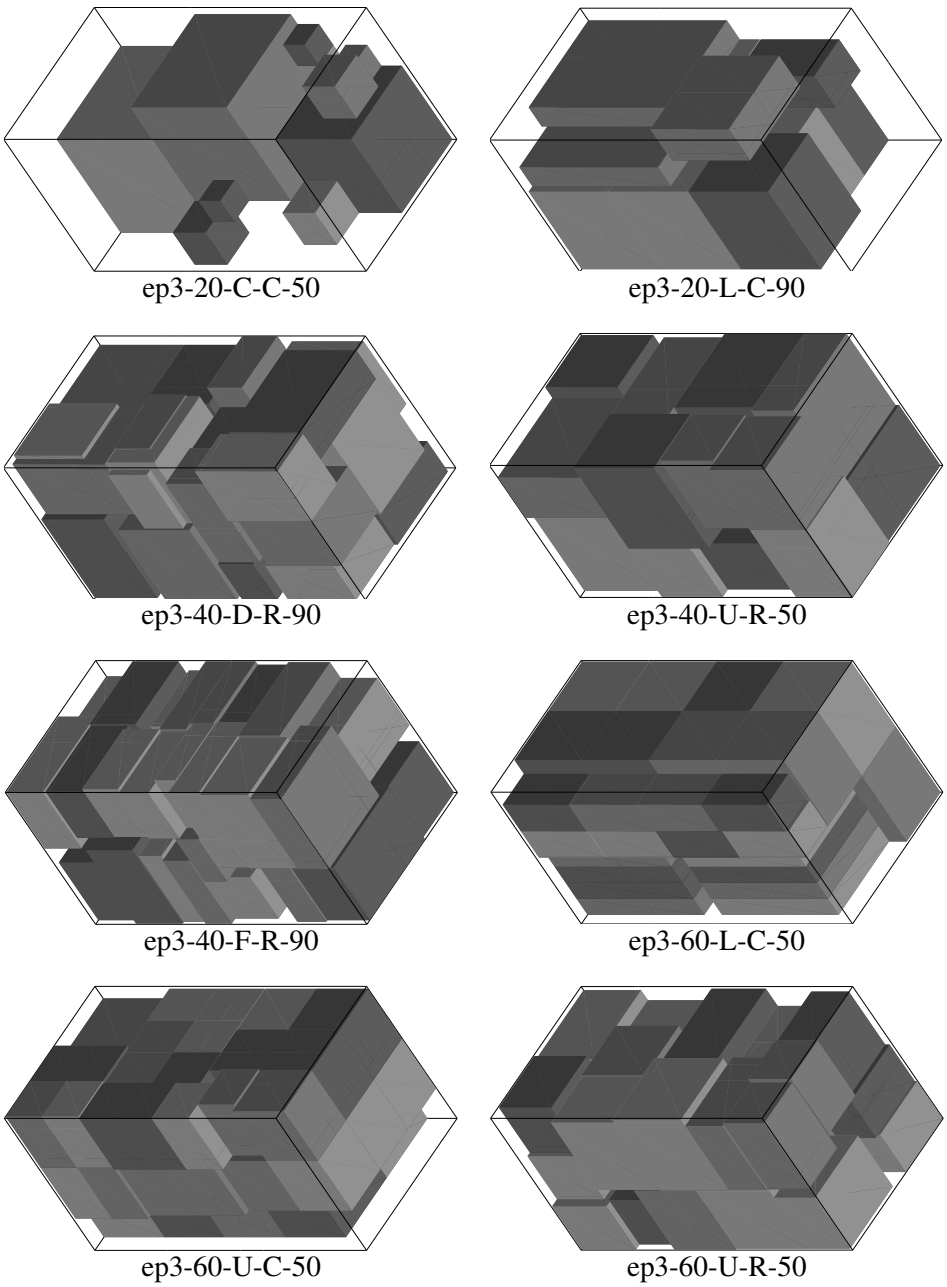


Figure 10: Best results for 8 three-dimensional instances.

performs extremely well by generating results with an average gap to our upper bound of less than 3.0%.

The heuristic for the three dimensional case demonstrates the potential for the sequence triple representation. Exact methods are capable of solving small problems to optimality and tailored heuristics based on greedy principles work well for container loading problems. The proposed local search based heuristic performs very well for medium sized problems with an average gap to the upper bound of only 13%.

The heuristics are generally able to return very good results for both two- and three-dimensional problems within few minutes, and often within seconds for the classical two-dimensional benchmark instances.

Acknowledgements

The authors wish to thank the anonymous reviewers for their many helpful and supporting comments.

References

[1] Alvarez-Valdes, F. Parre no, and J.M. Tamarit. A tabu search algorithm for two-dimensional non-guillotine cutting problems. Technical Report TR07-2004, Universitat de Valencia, 2004.

[2] R. Alvarez-Valdes, F. Parre no, and J.M. Tamarit. A grasp algorithm for constrained two-dimensional non-guillotine cutting problems. *Journal of Operational Research Society*, 56:414–425, 2005.

[3] R. Baldacci and Marco A. Boschetti. A cutting-plane approach for the two-dimensional orthogonal non-guillotine cutting problem. *European Journal of Operational Research*, 2006. Available online.

[4] J. E. Beasley. A population heuristic for constrained two-dimensional non-guillotine cutting. *European Journal of Operational Research*, 156:601–607, 2004.

[5] J.E. Beasley. Algorithms for two-dimensional unconstrained guillotine cutting. *Journal of the Operational Research Society*, 36:297–306, 1985.

[6] J.E. Beasley. An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research*, 33:49–64, 1985.

[7] M.A. Boschetti, E. Hadjiconstantinou, and A. Mingozzi. New upper bounds for the two-dimensional orthogonal cutting stock problem. *IMA Journal of Management Mathematics*, 13:95–119, 2002.

[8] A. Caprara and M. Monaci. On the 2-dimensional knapsack problem. *Operations Research Letters*, 1(32):5–14, 2004.

[9] N. Christophides and C. Whitlock. An algorithm for two dimensional cutting stock problems. *Operations Research*, 25:30–44, 1977.

[10] R.S. Dembo and P.L. Hammer. A reduction algorithm for knapsack problems. *Methods of Operations Research*, 36:49–60, 1980.

[11] S. P. Fekete and J. Schepers. A new exact algorithm for general orthogonal d-dimensional knapsack problems. In *Algorithms ESA '97, Springer Lecture Notes in Computer Science*, volume 1284, pages 144–156, 1997.

[12] S. P. Fekete and J. Schepers. On more-dimensional packing III: Exact algorithms. *submitted to Discrete Applied Mathematics*, 1997.

[13] S. P. Fekete and J. Schepers. A general framework for bounds for higher-dimensional orthogonal packing problems. *Mathematical Methods of Operations Research*, 60:81—94, 2004.

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

[15] E. Hadjiconstantinou and N. Christophides. An exact algorithm for general, orthogonal, two-dimensional knapsack problems. *European Journal of Operational Research*, 83:39–56, 1995.

[16] M. Hifi. Two-dimesional (un)constrained cutting stock problems. *http://www.laria.u-picardie.fr/hifi/OR-Benchmark/2Dcutting/*, 2006.

[17] H.Murata, K.Fujiyoshi, S.Nakatake, and Y.Kajitani. Vlsi module packing based on rectangle-packing by the sequence pair. *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, 15:1518–1524, 1996.

[18] S. Jakobs. On genetic algorithms for the packing of polygons. *European Journal of Operational Research*, 88:165–181, 1996.

[19] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, Berlin, Germany, 2004.

[20] K. K. Lai and J. W. M. Chan. Developing a simulated annealing algorithm for the cutting stock problem. *Computers and Industrial Engineering*, 32:115–127, 1997.

[21] K. K. Lai and J. W. M. Chan. A evolutionary algorithm for the rectangular cutting stock problem. *International Journal on Industrial Engineering*, 4:130–139, 1997.

[22] T.W. Leung, C. H. Yung, and M. D. Troutt. Applications of genetic search and simulated annealing to the two-dimensional non-guillotine cutting stock problem. *Computers and Industrial Engineering*, 40:201–204, 2001.

[23] T.W. Leung, C. H. Yung, and M. D. Troutt. Application of mixed simulated annealing-genetic algorithm heuristic for the two-dimensional orthogonal packing problem. *European Journal of Operational Research*, 145:530–542, 2003.

[24] D. Mack, A. Bortfeldt, and H. Gehring. A parallel hybrid local search algorithm for the container loading problem. *Internatinal Transaction in Operations Research*, 11:511–533, 2004.

[25] S. Martello, M. Monaci, and D. Vigo. An exact approach to the strip packing problem. *INFORMS Journal on Computing*, 3(15):310–319, 2003.

[26] S. Martello, D. Pisinger, D. Vigo, E. den Boef, and J. Korst. Algorithm 864: General and robot-packable variants of the three-dimensional bin packing problem. *ACM Transactions on Mathematical Software*, 33:1–7, 2007.

[27] A. Moura and J. F. Oliveira. A grasp approach to the container-loading problem. *IEEE Intelligent Systems*, 20(4):50–57, 2005.

[28] D. Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45:758–767, 1997.

[29] D. Pisinger. Heuristics for the container loading problem. *European Journal of Operations Research*, 3(141):382–392, 2002.

[30] D. Pisinger. Denser packings obtained in O(n log log n) time. *INFORMS Journal on Computing*, 19:395–405, 2007.

[31] D. Pisinger and M. Sigurd. The two-dimensional bin packing problem with variable bin sizes and costs. *Discrete Optimization*, 2:154–167, 2005.

[32] D. Pisinger and M. M. Sigurd. Using decomposition techniques and constraint programming for solving the two-dimensional bin packing problem. *INFORMS Journal on Computing*, 19:36–51, 2007.

[33] G. Wascher, H. Haussner, and H Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 2007. Available online.

[34] X.Tang and D.F.Wong. Fast-sp: a fast algorithm for block packing based on sequence pair. In *Asia and South Pacific Design Automation Conference*, 2001.

[35] X.Tang, R.Tian, and D.F.Wong. Fast evaluation of sequence pair in block placement by longest common subsequence computation. In *Proceedings of DATE 2000 (ACM), Paris, France*, pages 106–110, 2000.

[36] P. Y.Wang. Two algorithms for constrained two dimensional cutting stock problems. *Operations Research*, 31:573–586, 1983.

Submitted. 2007

Heuristics for container loading of furniture

Jens Egeblad (jegeblad@diku.dk), Claudio Garavelli (c.garavelli@poliba.it),
Stefano Lisi (s.lisi@poliba.it), David Pisinger (pisinger@diku.dk)

Abstract

We consider a real-life container loading problem which occurs at a typical furniture producer. The problem is to determine an optimal subset from a larger set of furniture which can be loaded into a container of given dimensions. Each item has an associated profit and a loadable subset of items with maximal total profit is considered optimal. In the studied company, the problem arises during the planning of transportation of products to clients hundreds of times daily. The instances may contain more than one hundred of different items with irregular shapes. Large-sized items are combined in specific structures to ensure proper protection of the items during transportation and to reduce the complexity of the remaining problem. We have developed a method composed of several heuristics which are applied successively to the problem. The average loading utilization is 91.3% for the most general instances with average running times around 100 seconds.
Keywords: Packing, Combinatorial Optimization, Logistics, Transportation, Heuristics .

1 Introduction

The container loading optimization problem is a central problem in the industry, where it appears in various formulations like *bin-packing*, *knapsack packing*, *container loading* and *multi-container loading*. Surveys on packing problems were presented by Dyckhoff et al. [24] or Wäscher et al. [62].

In this paper we consider container loading of pieces of furniture. The problem occurs at a typical furniture producer and solutions to hundreds of such problems every day may be required. Solutions must be generated within minutes on commodity hardware. Items may have non-rectangular (irregular) shapes and each item has an associated profit value to describe how desirable it is to load.

The problem we address can be formulated as a *three-dimensional knapsack packing problem with irregular shapes*; Given a consignment of items and container dimensions W, H and D, the objective is to determine the maximal profit subset of the consignment, which can fit inside the container. It is easy to see that, since the one-dimensional knapsack problem is NP-hard (see e.g [41]), the three dimensional variant is also NP-Hard. In the typology by Wäscher et al. [62] the problem belongs to the category *strongly heterogeneous three-dimensional Single Knapsack Problem (SKP) with irregular shapes*.

Items (products produced) are divided into three different categories:

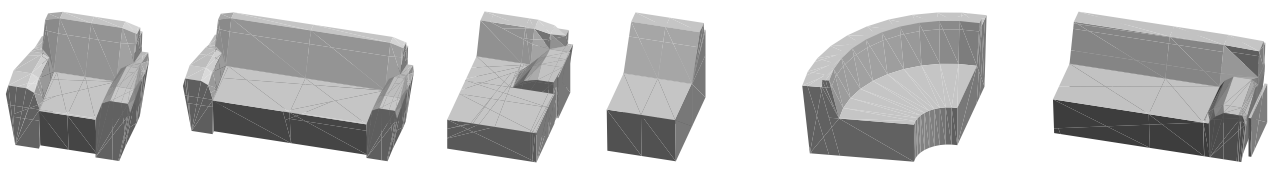


Figure 1: *Example of irregular items to be loaded.*

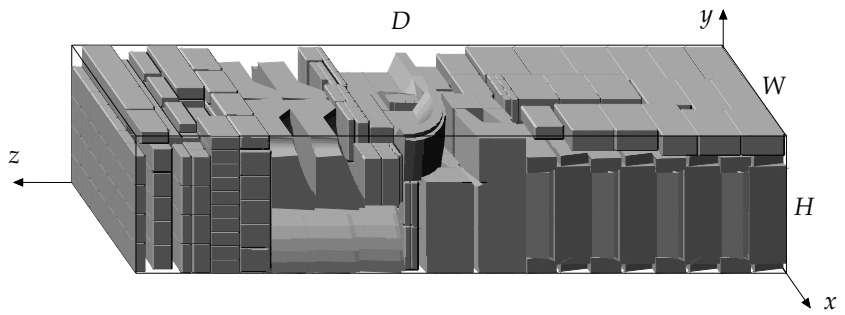


Figure 2: *Example solution with more than 300 items and our coordinate system convention.*

- **Large:** Irregularly shaped items such as armchairs, 2- and 3-seat sofas, chaise lounges and corners which are used to combine two segments of a sofa to a corner-sofa. To avoid damage during transportation, items must be placed in a stable location and only a specific set of rotations are allowed. The validity of a position and rotation may also depend on the surrounding items. See Figure 1.

- **Medium-sized:** Box-shaped robust items, such as ottomans. All six possible axis-aligned rotations are allowed.

- **Small:** Small accessory box-shaped items or items loaded in cardboard boxes, e.g. vases, lamps and glass-plates for tables. These items have different levels of fragility and may be allowed only a subset of the six possible axis aligned rotations.

An example solution with more than 300 items from all three categories is illustrated on Figure 2. For a problem instance we let L, M, S and $I = L \cup M \cup S$ be the sets of large, medium, small items and all items respectively. Instances are generally weakly heterogeneous and we let $\mathcal{L}$, $\mathcal{M}$, $\mathcal{S}$ and $\mathcal{I}$ be the different types of large, medium, small and all items respectively.

Although all items $i \in I$ have an associated profit p_i, large items are always more desirable than medium-sized items which in turn are more desirable than small-items.

In this paper we present a new strategy for handling problems with three-dimensional irregular shapes based on combination and simplification of items. This is achieved by a geometric algorithm and heuristics that generate building blocks called templates. To place large items we also introduce a new simple paradigm called quad-walls, which uses templates for efficient placement.

Our solution method is divided into several steps; during a preprocessing stage combinations of large items are determined. Then a tree-search heuristic finds an initial solution of L. A local search heuristic is used to refine the initial solution and ensure stability. Next, medium-sized items are placed using a greedy-approach. Finally, small items are placed at the end of the container, using a wall-building heuristic, and in remaining free space, using a greedy heuristic.

This paper is organized as follows: First, we present an overview of previous relevant work in Section 2. Then, in Section 3 we present an overview of our heuristic method and devote a section to each step of the optimization process in Section 4 to 10. Finally, in Section 11 we present the experimental results followed by a conclusion in Section 12.

2 Related Work

While the three-dimensional knapsack problem with irregular shapes that we study in this paper has not been studied in the literature, several variants of the problem have been investigated since the seminal work of Gilmore and Gomory [32] in the 1960's.

For two- and three-dimensional packing the most common problem definitions are: *Container loading, pallet packing, strip-packing, bin-packing* and *knapsack-packing*. In general some or all container dimensions are given and one must find a placement of items such that the items do not overlap and some objective is minimized or maximized. The problems are also commonly NP-hard. In the following we discuss prior work in each related problem category.

2.1 Knapsack-packing

In the one-dimensional knapsack problem one is given a maximal weight W and a set of items each with an associated weight w_i and profit p_i,and a subset of the items $I' \subseteq I$ must be selected such that $\sum_{i \in I'} w_i \leq W$ and $z = \sum_{i \in I'} p_i$ is maximized.

In the two- and three-dimensional knapsack problems we are given container dimensions, $W \times H(\times D)$, and a set of rectangular items. Each item has an associated profit value and one has to select a maximal profit subset of the items that can be placed in the knapsack without overlap.

While the one-dimensional knapsack problem has been thoroughly investigated (see e.g. [41]), the multi-dimensional variants have received less attention. Although we are unaware of work within the field of irregular shapes some work has been done with rectangles and boxes.

Several Integer Programming formulations for two-dimensional knapsack problem exists (see e.g. [4, 11, 37]), but they generally suffer from large numbers of integer variables, and numerous symmetric solutions.

A general approach for packing problems with rectangular items was proposed by Fekete and Schepers [26, 27, 28]. Among the list of problems they are able to solve are multidimensional knapsack problems. Their approach is a branch-and-bound algorithm which assigns items to the knapsack without specifying the position of the rectangles. For each assignment of items an advanced graph representation is used to decide if a feasible assignment of coordinates to the items is possible. A branch-and-bound algorithm for the two-dimensional knapsack problem was also developed by Caprara and Monaci [17]. As in the work by Fekete and Schepers, items are assigned to the knapsack without specifying the position of each item and an enumeration scheme from Martello, Monaci, Vigo [46] is used to ensure feasibility.

In the area of heuristics Egeblad and Pisinger [C] use the sequence-pair (see [49, 54, 63]) to represent two-dimensional feasible placements and a novel representation for three dimensions. Representations are modified with Simulated Annealing. Also Alvarez-Valdes et al. [1, 2] applied both GRASP and Tabu-search to the two-dimensional knapsack problem with very impressive results.

2.2 Container Loading

A specific version of the three-dimensional knapsack problem is the *container loading problem* (CLP). Here the "profit" value of each item is equal to its volume. Thus the objective is to maximize the utilization of the container volume.

Solution methods for the container loading problem often utilize a form of wall-building technique originally introduced by George and Robinson [31]. In wall-building the container is filled in the depth

with walls of items. Each wall has dimensions $W \times H \times d_i$ where d_i is the depth of the ith wall. For each wall one usually selects a *layer defining box* (LDB) and let d_i be the depth of that box.

Once the depth of the wall has been determined it is filled either by considering the simpler three-dimensional packing problem ([29]), or a two-dimensional packing problem. While George and Robinson [31] solves the two-dimensional packing problem by placing items in shelves, Pisinger [53] divides the wall into horizontal and vertical strips and each strip is packed by solving a one-dimensional knapsack problem.

Generally the *wall-building* approaches rely heavily on selection of the LDB and efficient strategies for packing each wall. Bischoff and Marriott [7] compare different ranking functions for the LDB, without determining a clear winner. This illustrates the need to elaborate on the greedy strategy, and much better results are achieved when LDB-selection is integrated with metaheuristics. Experiments have been conducted with genetic algorithms ([29]), tabu-search ([10]), and tree-search ([53]).

While the advantage of wall-building is that the problem is reduced to simpler sub-problems, wall-building strategies commonly suffer from the fact that space is lost when items do not fully utilize the depth of a wall.

Several authors have suggested alternative ways to represent free space in the container. Morabito and Arenales [47] suggest a *slicing tree* representation, where each tree corresponds to a guillotine cutting of the container and the leafs represent single boxes. Gilmore and Gomory [32] arrange boxes in *towers* which are placed on the container floor, thus reducing the problem to two dimensional packing. This strategy is also used with the genetic algorithm by Gehring and Bortfeldt [30]. Scheithauer [56] use a three-dimensional *contour* representation along with a form of dynamic programming. Ngoi et al. [50] use a matrix for each cross section in the height of the container to represent free space. Bischoff [8] later simplified this apporach by representing the available height for every location with just one matrix. Eley [25] use a list of available space, which is updated each time a new block of boxes is placed. An interesting approach was also suggested by Terno et al. [59], where layers are build using a non-slicing structure for two-dimensional packing called M4.

While methods for multidimensional knapsack problems generally work well with less than 50-100 items, heuristics for container loading problems are typical geared to problems with 100-200 items. At the time of writing, some of the highest utilization values have been achieved by Mack et al. [45] using a parallelized hybrid of tabu-search and simulated annealing running on 64 processors which utilizes the advanced wall-building procedure of [29]. In general they achieve higher than 90% utilization. Some of the best results with a non-parallel method were achieved by Moura and Oliveira [48] with overall utilization slightly below 90%.

2.3 Irregular Shapes

Most methods that consider packing of irregular shapes are made for the nesting problem, which is the problem of arranging a set of irregular shapes on a two-dimensional strip with fixed height and minimal width. The majority of successful heuristics for the nesting problem are iterative but can be roughly divided in two segments; *legal* and *relaxed* methods. Legal methods iteratively try to improve feasible solutions while relaxed methods also incorporate infeasible solution in which items overlap during the solution process.

In general shapes are represented by polygons. Art [3] was among the first to consider legal placement. He used an *envelope*-principle, which, for a partial solution, defines the legal positions of the next polygon. Legal placement methods also often utilize another concept; the so called *no-fit polygon* (NFP). Given two polygons P and Q the NFP can be defined as $\text{NFP}(P,Q) = \{p - q \mid p \in P, q \in Q\}$, which is the set of translations of Q such that P and Q overlap. Note that the NFP

is closely related to the Minkowski-sum. For two point sets A and B the Minkowski-sum is the set $\{a+b \mid a \in A,\ b \in B\}$, which means that the NFP is the two-dimensional Minkowski-sum where one polygon has been inverted in both axes.

Several methods places items sequentially using the NFP. Oliviera et al. [52] and Dowsland et al. [23] generates a sequence by heuristically estimating how well a piece fits with the next position.

Other authors used a modification scheme where the sequence is iteratively modified and re-evaluated. Blazewicz et al. [9] used tabu-search. Burke and Kendall [13, 14, 15] used the NFP with ant-colony algorithms, simulated annealing and evolutionary algorithms and more recently Burke et al. [12] used tabu-search and hill-climbing.

Also more problem specific heuristics have been investigated. Gomes and Oliveira [35] used a 2-exchange heuristic. Dowsland et al. [22] used a "jostling" mechanism where the placement is alternately generated left-to-right and right-to-left, based on the sequence of the previous placement. Recently, Gomes and Oliviera [36] used a combination of simulated annealing and linear programming to generate impressive results.

A number of researchers have considered *infeasible solutions* during the solution process. Here overlap of shapes is allowed but iteratively reduced. Once the total overlap has been reduced to 0, a feasible solution has been reached. Several authors have experimented both with a simplified raster-model and geometric models for measuring overlap. Lutfiyya et al. [44] use a raster model in conjunction with simulated annealing. Later Oliveira and Ferreira [51] experimented with both raster and geometric models. Dowsland and Bennel al. [5, 6] experimented with intersection depth as a measure of overlap, and combined LP-compaction methods by Li and Milenkovic [43] and the NFP for faster evaluation. A particular ambitious heuristic with relaxed placement is the 4-stage simulated annealing by Heckmann and Lengauer [38]. Lately, Egeblad et al. [A] use the metaheuristic *Guided Local Search* combined with a fast geometric algorithm to determine a minimal-overlap horizontal and vertical translation of a polygon. Currently the best results in the literature are evenly divided among this work and the work by Gomes and Oliveira [36].

2.3.1 Irregular 3D-packing

Three-dimensional packing of irregular shapes has received far less attention. Methods generally represent surfaces of shapes by triangle-mesh structures.

Ikonen et al. [39, 40] were among the first to consider optimization problems with irregular three-dimensional shapes. They proposed to use genetic algorithms with a relaxed placement method based on triangle intersection. Cagan et al. [16] also use a relaxed placement method, but with simulated annealing and spatial octrees (see e.g. [19]) to quickly determine pairwise overlap. Dickinson and Knopf [20, 21] use a legal placement method where items are placed sequentially, and each item is placed with an individual optimization heuristic, but the sequence is only placed once. Also Stoyan et al. [58] use a serial packing method. They have generalized the concept of ϕ-functions (which is a form of description of overlapping area similar to the NFP) to three dimensions. Recently, Egeblad et al. [A] generalized their 2D relaxed placement method to three dimensions with results surpassing those of Ikonen et al. and Dickinson and Knopf both in speed and quality.

2.4 Weight and Stability Considerations

Gehring and Bortfeldt [30] consider a set of constraints which are important with respect to the placement of items. The constraints they consider are: Items may be placed only on top of a stack with sufficient bearing strength to accommodate it. Some items may only be stacked a limited number of

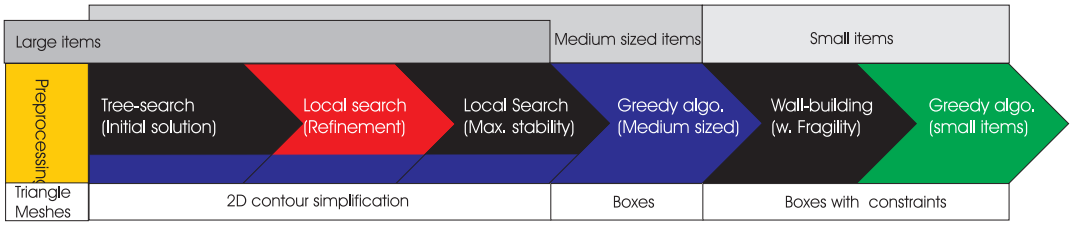


Figure 3: *The different stages of the solution process*

times on top of each other. Some items may only be at the top of a stack. Since strength of an item depends on its orientation, only a limited number of item-depending orientations may be used. Items must be loaded such that they do no drop to the floor during transportation. The center of gravity of the overall placement must coincide with the center of the container for transportation with trucks and airplanes.

The approach by by Gehring and Bortfeldt [30] was based on building stacks which makes it relatively simple to accommodate most of these constraints. To handle the COG constraints they divided the container space into layers (walls) parallel to the back of the container. Once volume-optimization is complete, individual walls are interchanged, mirrored or rotated by 180 degrees to move the COG within the demanded range.

Davies and Bischoff [18] combined the items into deeper layers called blocks and determined a good permutation of the blocks by random search. Eley [25] also apply this principle and reports that only 3-4 blocks are required in order for items to be placed acceptably with respect to the center of gravity.

A weight considereration technique by Ratcliff and Bischoff [55] was later refined by Bischoff [8] to use a matrix representation to describe not only the available space for each region of the floor but also the bearing-strength of each region. As the heuristic is constructive it is easy to ensure that items are positioned without violating load constraints.

3 Solution Process

Our solution process is divided into a number of stages which we outline in this section. The process completes the stages illustrated on Figure 3 one at a time from left to right. In the initial stages the heuristic considers mainly large items. Once a placement for large items is determined medium and small items are considered.

During a preprocessing step the full three-dimensional structure of the items is analyzed to determine how items can be placed relative to each other. To simplify the task of placement, item geometry is reduced for the following stages. The last three stages consider only box-items, but a set of constraints apply to the small items which make them complex to handle.

Each stage of the optimization process is described in the following:

1. **Preprocessing (*L*).** Generation of set of *templates* which describe sub-placements of items. See Section 4.

2. **Quad-wall bulding (*L*).** A tree-search heuristic which fills the container by selecting four templates which fit next to each other in the width and height of the container (quad-wall) determines a placement of large items. See Section 6.

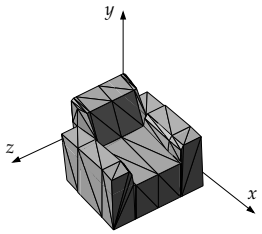


Figure 4: *The triangle mesh-structure of an item shown here in the item-coordinate system.*

3. **Local search (*L*).** Refinement of the quad-wall solution occurs via a local search heuristic for large items as described in Section 7.

4. **Stability (*L*).** A special local search heuristic, described in Section 8, ensures stability of the solution generated in step 3.

5. **Greedy algorithm (*M*).** Once the large items have been placed we proceed to place medium-sized items between and above large items with a greedy algorithm described in Section 9.1. As explained in Section 9.2 this algorithm is also used to evaluate placement of large items. This is indicated on Figure 3 by the area below the three first heuristic steps, which protrudes into the "Greedy algo." step.

6. **Wall-building (*S*).** The available space at the end of the container, which is maximized during the first stages, is filled with small items using a traditional wall-building heuristic to be described in Section 10.2.

7. **Greedy algorithm (*S*).** Remaining small items which are robust enough to be placed on top of large items, are placed using the greedy algorithm of step 5, as described in Section 10.3

3.1 Item Representation

Although three-dimensional shapes may be represented in different ways, we have chosen *a triangle-mesh structure (mesh)*. Here the surface of the items is represented by a list of triangles in space (See Figure 4).

Collecting triangle meshes is a complex problem. Although laser-scanning was considered we settled on parametric models. A set of approximately 50 parameters (measures) per item, taken from the physical model, are converted by a mesh-generation algorithm to a mesh of the item. In the problem studied, only large items are represented by meshes, since other items are packed as boxes.

4 Preprocessing

Prior to optimization, large items undergo two types of analysis; Geometric analysis and template building. The geometric analysis is used to determine *where* item-types must be positioned relative to each other to avoid overlap. Template building is used to create suitable sub-placements of a few large items, which are used to simplify the problem.

4.1 Geometric analysis

The geometric analysis determines non-overlapping compact placements of items relative to each other. Although several combinations could be considered, we only consider placements of two shapes (pairs), such that their surfaces are in contact with each other.

If rotation angles are fixed, the boundary of the three-dimensional Minkowski-sum of A and the inverse of B, describe all surface-contacting translations of B relative to A – Like the NFP in two dimensions (see Section 2.3). For non-convex polyhedra with respectively m and n features the combinatorial complexity of the Minkowski-sum can be as high as $O(m^3n^3)$ (see e.g. [57]) and thus this is an expensive operation.

To simplify the problem, we take advantage of the fact that the shapes we consider describe mainly sofas. When loaded in the bottom of the container, a sofa is first rotated 90 degrees around the x-axis (see Figure 4) and placed with its armrest on the container floor. Therefore the following conditions apply:

- Both items must stand on the ground, so only a two-dimensional set of relative translations may be allowed.
- Fragile parts of sofas must point towards each other, for protection during transportation.
- The items are generally semi-convex sofa-shapes which limits the number of possible relative positions. In general, for every relative x-translation we assume that there is one and only one acceptable y-translation.

Let meshes A and B (e.g. sofas or chairs) be defined as in the coordinate system on Figure 4. Although meshes are rotated around the x-axis when the pairs are placed inside the container we omit this step here, so one should imagine in the remainder that the xy-plane is the container floor. We let B^π be B rotated 180° around the z-axis, so that the fragile parts (the seats) of A and B^π can point towards each other.

The legal translations of B^π are those where B^π touches A and B^π has its seat pointing towards A. We begin by translating A and B^π in the z-direction such that the minimal z-coordinate of their bounding-boxes is 0. Then for any x-translation of B^π we wish to find the minimal y-translation such that B^π does not overlap with A. Figure 5 shows this y-translation as a function of x-translation. This function represents all the different possible pairs, we are allowed to consider, and Figure 6 illustrates five of these pairs.

To calculate the function we commence as follows: First let A' be the triangles from A with upwards pointing normal ($y > 0$) and B' be the triangles from B^π with downwards pointing normal ($y < 0$). All other triangles represent parts of the surfaces which point away from the opposite mesh.

For every triangle $t \in A'$ and every corner point, $p = (p_x, p_y, p_z)$, of a triangle from B' we determine the intersection of t and the plane $z = p_z$. This is either the empty set, a single point, or a line-segment which can be described by a linear function $\overline{f}_{p,t}(x)$ on a closed interval $\overline{J}_{p,t} = [\overline{J}^1_{p,t}, \overline{J}^2_{p,t}]$, that gives the line's y-coordinate for every x-coordinate. Assume the intersection is a line-segment and define $f_{p,t}(x) = \overline{f}_{p,t}(x - p_x)$ and $J_{p,t} = [J^1_{p,t} - p_x, J^2_{p,t} - p_x]$, then for every $x \in J_{p,t}$, $p + (x, f_{p,t}(x), 0) \in t$. I.e. $(x, f_{p,t}(x), 0)$ is the required translation of p, such that p touches t. If the intersection is not a line-segment define $J_{p,t} = \emptyset$.

We also determine $f_{p,t}(x)$ in x for every corner point, p, of any triangle in A' and every triangle t in B', i.e. the opposite set of linear functions.

Now we loop over all pairs of edges, (e_a, e_b), with positive x extent, from triangles in A' and triangles in B' respectively. For every edge pair e_a and e_b we determine a function f_{e_a,e_b} and half-open

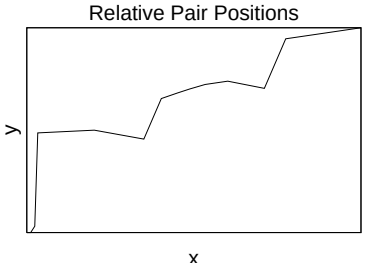


Figure 5: *Example output of the pairing algorithm described in Section 4.1. Two copies of the mesh from Figure 4 are paired together and the output is a piecewise linear function describing relative y-position for every relative x-position.*

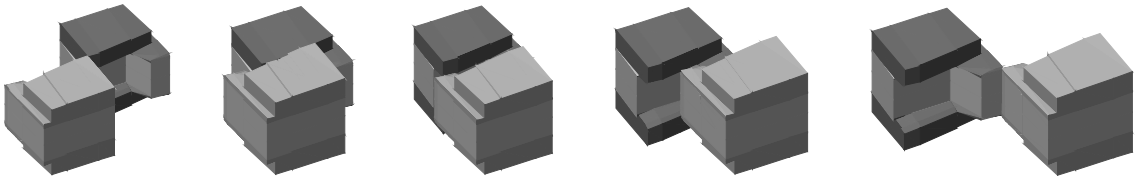


Figure 6: *Five different pairs which arise from the piecewise linear pair-function in Figure 5.*

interval J_{e_a,e_b} such that e_b translated $x \in J_{e_a,e_b}$ along the x-axis and $f_{e_a,e_b}(x)$ along the y-axis intersects with e_a.

For each linear function set $f_{p,t}(x) = -\infty$, $x \notin [J^1_{p,t}, J^2_{p,t})$ and $f_{e_a,e_b}(x) = -\infty$, $x \notin J_{e_a,e_b}$. Let F be the set of all such linear functions and define the piecewise linear function $f_{\text{all}}(x) = \max_{f \in F} f(x)$, as the maximum y-coordinate of any $f \in F$ and for every x. The function $f_{\text{all}}(x)$ can be generated in time $O(|F|^2)$ since every $f \in F$ is a line segment. In total the number of line segments generated is $O(|A'||B'|)$ and the total running time is $O(|A'|^2|B'|^2)$.

Let $B(x,y)$ be B^π translated $(x,y,0)$ units. Now for $x \in (-\infty,\infty)$ the set of translations $(x, f_{\text{all}}(x), 0)$ are translations of B^π such that one or more edges or point of a triangle in $B(x, f_{\text{all}}(x))$ touches A but no triangle of $B(x, f_{\text{all}}(x))$ intersects A.

4.2 Template building

The second part of the preprocessing stage determines a large set of feasible and stable sub-placements of items which we call templates. A template t represents a placement of items of different types and simplify the inner-portions of our heuristics by reducing the number of geometric computations and making it easier to ensure overall stability.

For a template t, let $(t^1,\ldots,t^n)$ be its item-types with $t^i \in \mathcal{L}$ for $i = 1,\ldots,n$ and let $|t| = n$ be the number of its item-types. For each t^i, $i = 1,\ldots,|t|$, the template also contains the relative position and rotation of t^i. Let T be the set of templates. For a template $t \in T$ we let t_p be the total profit of its item types. Let $w(t)$, $h(t)$, $d(t)$ be the dimensions of t's minimal axis-aligned box, $[0,w(t)] \times [0,h(t)] \times [0,d(t)]$, which contains its items (*bounding-box*).

If the template is used at a position $p \in \mathbb{R}^3$ items matching each of the item-types t^i are placed relative to p and orientated as in the template. We refer to the positioned sub-placement of items as a *bundle* and define the bounding-box a bundle b as $[b_{x1}, b_{x2}] \times [b_{y1}, b_{y2}] \times [b_{z1}, b_{z2}]$.

Since the same set of item-types may be combined in many different ways, we group similar templates. For a tuple $(t^1,\ldots,t^n)$ of item-types we let $R(t^1,\ldots,t^n) \subseteq T$ be the set of templates containing

the item-types $t^1, \ldots, t^n$. For $t \in T$ we let $R(t)$ be the set of templates containing the same item-types from t and we say that $s, t \in T$ are related if and only if $s \in R(t)$.

During this stage many templates are generated in each of the following categories: Singletons, stacks, pairs based on the geometric analysis of Section 4.1, pairs of stacks, user-defined templates and fused templates.

For every item-type $i \in \mathcal{L}$ a singleton template containing only i is generated for each allowed rotation. An example singleton-template is depicted in (Figure 7 (a)).

Stacks, as shown in Figure 7 (b), are generated by a recursive algorithm that adds item-types on top of each-other until the container height is reached. All such possible templates are constructed. The bounding-box of the meshes is used to avoid overlap when stacking. To limit the number of templates an item is not allowed on top of a shorter item. Because items are large, stacks commonly contain less than four items.

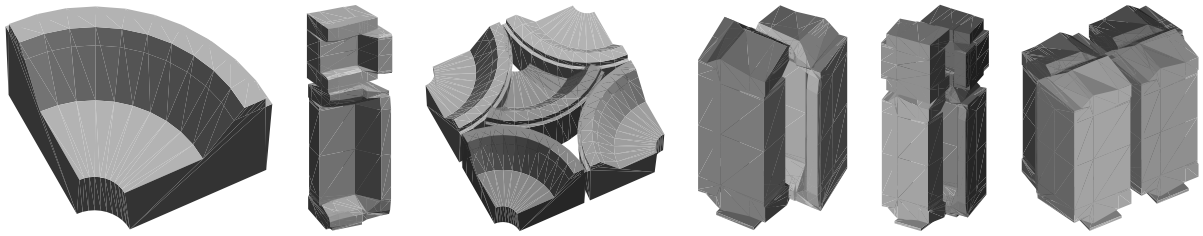
The geometric analysis (see Section 4.1) returns all possible legal pairs between any two sofa item-types as illustrated in Figure 7 (d). To limit the solution space, only a low number of these pairs are used as templates. The pairs chosen are: P1: Pair with minimal bounding-box volume. P2: Pair which fills the container in the width next to P1 (maximal sliding). P3 and P4: rotated versions of P1 and P2 (around y-axis). P5: Pair which occupies half the container width. Although more pairs theoretically allow for better solutions, experiments showed that in practice they tend to have the opposite effect – Presumably because the size of the solution space is increased.

Pairs of stacks (See figure 7 (e)) are built by combining the techniques for pair and stack construction. To limit the number of combinations the stacks are build by adding items one-by-one in order of non-increasing height to the smallest stack. Once the composition of the two stacks have been determined possible combinations are determined using the geometric analysis of Section 4.1 similar to pairs.

Because items in our context have a soft surface, geometric analysis is not as accurate as physical measurements and so there is a great advantage in using measured data. In our implementation we have around 40 different kinds of user-defined templates for which the user can specify the constituent types, width, height and depth of the template. See Figure 7 (c) for an example of a user-defined template.

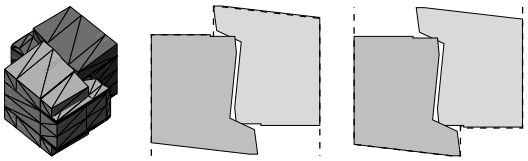
Finally, for reasons which will be clarified in Section 5, the templates of the previous categories are also fused heuristically together along the x-axis to form new templates (See Figure 7 (f)). Specifically any pair of constructed templates are fused if there is room for a third template in the container width. This process is recursive over all templates and may generate new larger templates which are fusions of many narrow templates. To ensure that not too many combinations are generated, this preprocessing step begins by repeating similar templates and continues to combine distinct templates where the resulting utilization will be high. This process stops once f fused templates have been created. We discovered that $f = 2000$ is an adequate number.

In general a template is generated for each allowed orientation of the templates described in the prior sections. This implies a discrete rotational model — not all rotations are possible. This limitation works well with the real-life situation, where only a limited number of rotations may be allowed for some templates or items for quality assurance reasons. For every template at least a y-axis rotated variant will also be generated.



(a) (b) (c) (d) (e) (f)

Figure 7: *Examples of templates.*



(a) (b) (c)

Figure 8: *Contours (b and c) of a template (a).*

5 Sequence Representation

Due to the size of large items there is generally only room for at most two pair-templates in the width and height of the container. Therefore we limit the possible ways a template can be placed in the in xy-plane to four alignments numbered $1-4$ matching the four corners *lower-left*, *lower-right*, *upper-left* and *upper-right*. A solution of n large items may be represented as a sequence of template and alignment pairs of the form $<(t_1,a_1),\ldots,(t_n,a_n)>$ for $t \in T$ and $a \in \{1,\ldots,4\}$. To convert a sequence into a placement templates are placed one-by-one in the order of the sequence.

Assume we apply template t to place bundle b then we proceed as follows. First we determine x and y-extents of a such that (b_{x1},b_{y1}), (b_{x2},b_{y1}), (b_{x1},b_{y2}) and (b_{x2},b_{y2}) coincide with $(0,0)$, $(0,W)$, $(H,0)$ and (H,W) for respectively the lower-left, lower-right, upper-left and upper-right alignments. To determine the z-extent the set of current bundles, B, is considered. Let $B' \subseteq B$ be the set of bundles for which $[b_{x1},b_{x2}] \times [b_{y1},b_{y2}]$ overlaps in the xy-plane. Then b is placed such that $b_{z1} = \max_{b' \in B'} b_{z2}$.

Realizing an entire sequence of n templates takes $O(n^2)$ since this process takes $O(|B|)$ time for each template, however if bundles are sorted and searched in order of highest z, finding the z-coordinate generally requires only evaluation of a few top placed bundles, since the search may stop once all templates have lower z-coordinate than the current maximum.

As an alternative to alignments, templates could be placed according to some greedy principle. However, this strategy easily ruins the entire solution if for instance two templates are exchanged whereas alignments help maintain locally optimal placements.

It is easy to determine a lower z-coordinate than the one arising from the use of bounding-boxes. Since the x- and y-coordinate of each item within the template are fixed once the template is aligned one can search for a collision along the z-axis, using the meshes of each template. This operation is computationally expensive and in the sequel we describe how it can be quickened by reducing to two dimensions and caching values.

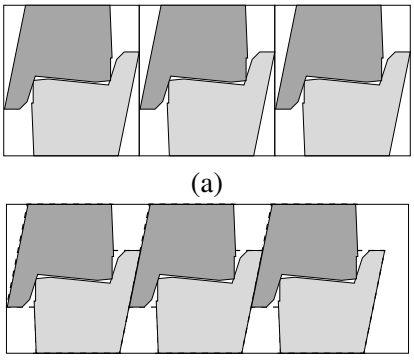


Figure 9: *A substantial amount of space can be recovered by using the contours of templates to determine required distance between bundles (b) rather than the bounding-box (a).*

5.1 Profile Considerations

Rather than considering the full three-dimensional structure of templates we reduce the collision detection problem to a two-dimensional problem. This strategy works by considering either the xz-plane or yz-plane, but the techniques are similar and we will only describe the approach for the xz-plane here.

For each template t we generate a contour of its extreme points in the xz-plane (see Figure 8). Let t_T be the set of triangles of the positioned and oriented meshes of items in t. Let $t_u(x) = \max\{z \mid \exists y,\ s \in t_T\ :\ (x,y,z) \in s\}$ be the value of the maximal z-coordinate of any point on a triangle in t_T that overlaps with x (Figure 8 (b)), and let $t_l(x) = \min\{z \mid \exists y,\ s \in t_T\ :\ (x,y,z) \in s\}$ (Figure 8 (c)). t_l and t_u are piecewise linear functions which can be constructed from the edges of each triangle.

For a bundle b of a template t, define the translated contours $b_u(x) = t_u(x - b_{x1}) + b_{z1}$ and $b_l(x) = t_l(x - b_{x1})$. Now assume, we have aligned a bundle b' (only x- and y-coordinates have been determined). Then if we place b' $b'_{z1} = \max_x b_u(x) - b'_l(x)$ the templates of the two bundles will not overlap.

Since the items we consider generally consists of surfaces which are not completely parallel to the coordinate system planes, a considerable amount of space in the container can be saved by this simple strategy, as seen in Figure 9. On the other hand the nature of the items are such that little, if anything, would be gained by considering a full three-dimensional collision detection.

5.2 Caching

The piecewise linear functions t_u and t_l are calculated for all templates during preprocessing. With many thousands of templates precalculation of required distance between all template-pairs for any relative position is not a viable approach. However, since templates can only be aligned to the left and right side of the container a template can only be placed with two different x-coordinates. Therefore there are only four possible ways the templates can be positioned relative to each other in the x-direction.

Additionally, when the required distance between two templates is calculated as in Section 5.1, it is stored in a binary search tree for later use. Additionally, if accurate physical measures for two templates are known, these can be inserted into the data structure during preprocessing.

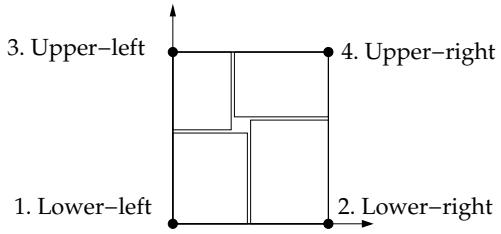


Figure 10: *A quad-wall consists of up-to 4 templates that may be placed in the four alignment corners.*

6 Quad-Wall Building

The first step of the heuristic consists of a tree-search algorithm for large items. The tree-search algorithm fills the container in the depth by determining good 'walls' consisting of four templates (*quad-wall*). Each quad-wall is ranked according to how well it fills the width and height of the container. Quad-walls are appended in the depth until the container is full at which point the tree-search heuristic backtracks and uses walls with less rank.

It is important to note that, unlike traditional wall-building schemes, quad-walls have no boundaries in the z-direction and, when placed, their constituents are pushed as far back in the container as possible as described in Section 5. Another important element is that the constituents of a quad-wall are repeated a number of times in the depth, but not necessarily an equal number of times. Determination and realization of walls are described in the following sections.

6.1 Quad-Wall Selection

A quad-wall is a combination of templates that can fit next to each other in the width and the height of the container. In practice the most commonly used type of template is a pair. Using the same argument as in the start of Section 5 there is commonly only room for up-to four different templates in the width and height of the container (see Figure 10).

Since the number of templates in some instances can reach $30,000$ the number of combinations of four templates could be as high as $30,000^4$, and it is intractable to evaluate all possible walls within reasonable time. Rather, we use an additional tree-search heuristic to find high-quality walls. A fundamental part of evaluating walls is to rank individual templates. For a template $t \in T$ we define $rank(t) = \frac{p(t)}{h(t)d(t)}$, which indicates how profitable t is per height and depth unit.

Quad-walls are constructed by assigning a template to each of the corners in the order lower-left (1), lower-right (2), upper-left (3) and upper-right (4). A partial wall, $< t_1, \ldots, t_i >$ for $i = 1, \ldots, 4$, is an assignment of a template $t_j \in T \cup \{nil\}$ to each corner j, $j = 1, \ldots, i$. A corner j may be empty if $t_j = nil$. The templates that may be assigned to corner i depend on the assignment of templates to the first $i-1$ corners. Let $T_i(< t_1, \ldots, t_{i-1} >) \subseteq T$ be the set of templates allowed at corner i if templates $t_1, \ldots, t_{i-1}$ have been assigned to corners 1 to $i-1$.

The search begins by ranking all partial walls containing lower-left templates t_1:

$$rank_1(< t_1 >) = rank(t_1) + \max_{t_2 \in T_2(< t_1 >)} rank(t_2) + \max_{t_3 \in T_3(< t_1 >)} rank(t_3). \quad (1)$$

The set $T_2(< t_1 >)$ depends only on the width of t_1. Therefore all templates allowed in the lower-right corner can be stored in a balanced search-tree with width as key, and the best ranked t_2, given the width of t_1, can be found in time $O(\log |T|)$. A similar argument holds for t_3. This way each template

can be evaluated in $O(\log|T|)$ time. Let $\overline{T}_1$ be the at most m_1 best ranked walls with a lower-left template where m_1 is a parameter for the heuristic.

The heuristic then proceeds recursively to rank walls with templates assigned to the remaining corners. Define $rank_i(< t_1,\ldots,t_i >)$ of a partial wall consisting of i templates as:

$$rank_i(< t_1,\ldots,t_i >) = \begin{cases} rank(t_i) + \max_{t\in T_4(<t_1,\ldots,t_i>)} rank(t) & \text{for } i = 2,3 \\ rank(t_i) & \text{for } i = 4 \end{cases}, \tag{2}$$

and let $\overline{T}_i(< t_1,\ldots,t_i >)$ be the set of at most m_i best ranked partial walls with respect to $rank_i(< t_1,\ldots,t_{i-1},t_i >)$ for $t_i \in T_i(< t_1,\ldots,t_{i-1} >)$, where m_i is a parameter for the heuristic. Then let $\overline{T}_i$ be the at most $\Pi_{j=1}^{i} m_j$ partial walls consisting of templates at corners $1,\ldots,i$ and let it be defined recursively as

$$\overline{T}_i = \cup_{<t_1,\ldots,t_{i-1}>\in\overline{T}_{i-1}} \overline{T}_i(< t_1,\ldots,t_i >). \tag{3}$$

This means that at each corner we generate the at most $\Pi_{j=1}^{i} m_i$ best ranked partial walls, $\overline{T}_i$, by appending the best ranked partial walls from the previous corner, $\overline{T}_{i-1}$, with templates at corner i. The parameters m_i determines the number of branches at each level of the tree search and are used to describe the width of the search-tree. The set $\overline{T}_4$ are walls with template assignments to all four corners and $|\overline{T}_4| \le m_{tot} = \Pi_{j=1}^{4} m_j$ which we will consider in the following. Note that not all corners need to be assigned a template. This can be dealt with by including a template with no profit and item-types in T.

6.2 Domination

If two walls $w \in \overline{T}_4$ and $v \in \overline{T}_4$ consists of the same item-types, v may *dominate* w by utilizing the space strictly better, and w may be discarded.

To test for domination, all pairs of quad-walls $w,v \in \overline{T}_4$ with equal elements are compared by considering four points based on the bounding-boxes of templates from each wall. Assume a wall w consists of templates $< w_1,w_2,w_3,w_4 >$, and each template w_i would be placed as bundle b^i, if it were to be used at this time, then we define

$$q_1(w) = (b^1_{x2}, b^1_{y2}, b^1_{z2}), \quad q_2(w) = (b^2_{x1}, b^2_{y2}, b^2_{z2}), \quad q_3(w) = (b^3_{x2}, b^3_{y1}, b^3_{z2}), \quad q_4(w) = (b^4_{x1}, b^4_{y1}, b^4_{z2}).$$

The points $q_i(w)$, $i = 1,\ldots,4$ are illustrated on Figure 11. Now, to determine if wall w is dominated by v, we require that $q_i(v)_z \le q_i(w)_z$ for all $i = 1,\ldots,4$ and:

$$\begin{array}{lllllllllll} q_1(v)_x & \le & q_1(w)_x, & q_1(v)_y & \le & q_1(w)_y, & q_2(v)_x & \ge & q_2(w)_x, & q_2(v)_y & \le & q_2(w)_y, \\ q_3(v)_x & \le & q_3(w)_x, & q_3(v)_y & \ge & q_3(w)_y, & q_4(v)_x & \ge & q_4(w)_x, & q_4(v)_y & \ge & q_4(w)_y. \end{array}$$

If w is dominated we simply remove it from $\overline{T}_4$. Note that cases where walls have no assigned template in one (or several) corner(s) i (i.e. $w_i = nil$), can be dealt with by assigned appropriate values to q_i.

6.3 Quad-Wall-appending

The m walls from $\overline{T}_4$ with highest $rank(w) = \sum_{i=1}^{4} \frac{p(t_i)}{d(t_i)}$ for $w =< t_1,t_2,t_3,t_4 >\in \overline{T}_4$ are selected and the best of these is appended to the current solution.

In traditional wall-building a wall is constructed once and independently of the previously placed elements. Here, however, we apply the templates $< t_1,\ldots,t_4 >$ of a quad-wall by a set of rules. Let d_i

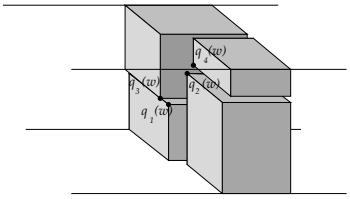
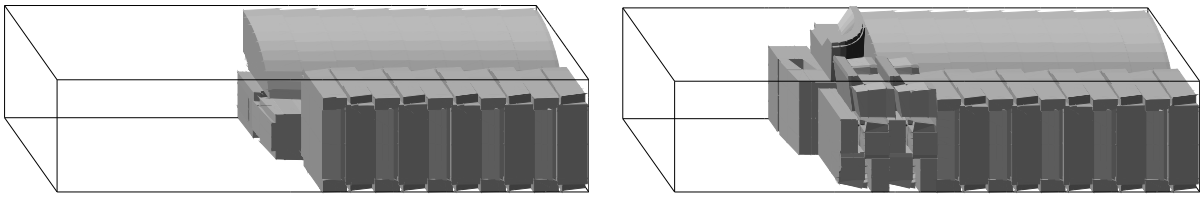


Figure 11: *The four points of each wall used for domination check.*



(a) (b)

Figure 12: *Example placement using quad-walls. (a) Placement after use of first quad-wall (3 templates). (b) Placement after use of second quad-wall (4 templates).*

be the maximum z-coordinate of any bounding-box of current bundles at corner i of the container for $i = 1,\ldots,4$. Let d_i^+ be the resulting maximum z coordinate if template t_i is applied at corner i. Then we apply templates t_i to corner i from the wall one-by-one using the following rule.

1. If $d_4^+ \leq d_2$ and $d_4^+ \leq d_3$ we apply t_4,
2. otherwise if $d_3^+ \leq d_1$ we apply t_3 ,
3. otherwise if $d_2^+ \leq d_1$ we apply t_2 ,
4. otherwise we apply t_1.

We repeat this until there are insufficient items or space to apply one of the four templates.

Note that by these rules a wall is not simply repeated, rather templates are distributed depending on depth. E.g. a very deep template will not be applied as often as a thin template. It is also important to note that as the templates are placed, they are pushed as far back in the container as possible, so while a quad-wall is a collection of items in the width and height of the container it does not fill "slices" of the container. Example of placements after the first and second wall have been used are shown of Figure 12.

6.4 Backtracking

The quad-wall placement is embedded in a tree-search heuristic. Each stage where a wall is appended to the container is a node in a search tree and each candidate wall corresponds to a path to a child-node in the search tree. Once the the container is filled and there is insufficient space for further templates at the end of the container, the heuristic back-tracks in the search tree and continues with the second-best ranked wall. When all walls of a node have been investigated the heuristic back-tracks to the node's parent and continues with the next wall of the parent.

At most 150,000 forward and backward steps in the search-tree are allowed due to time-constraints. To ensure that the heuristic does not search only in the end of the container the values of m_i are reduced at the lower levels of the tree and generally we set $m_i = \frac{1}{2}m_{i-1}$, to balance the inner tree-search.

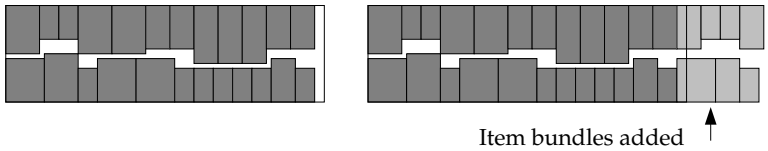


Figure 13: *The best solution from the quad-wall heuristic (a) is appended with remaining items that may protrude out of the container (b) before local search is applied.*

At every q steps (e.g. $q = 10,000$) we back-track to the root node of the tree. This strategy works adequately to ensure that searching is done both within the beginning and the end of the container.

7 Local Search

The best solution found during tree-search is used as start-solution for a local-search heuristic. During local search, solutions are represented as the type of sequences described in Section 5 and one side of the container is 'open' to allow bundles to extend beyond the container boundary. Only the profit of those bundles which are completely inside the container contribute to the objective function. Items which are not part of the initial solution are added to the end of the loading as additional bundles, and will in general protrude the container as shown in Figure 13. The idea is that the local search will be able to move some of these items inside the container boundaries and thereby increase the solution value. This strategy is similar to the one used by [C] for two-dimensional and three-dimensional knapsack problems with boxes.

Let the current solution be defined as the sequence $\sigma =< (t_1,a_1),\ldots,(t_n,a_n) >$ and let $\sigma(i) = (t_i,a_i)$ be the ith element of the sequence, then the local search neighborhoods consists of the following:

Exchange: For every i,j, with $1 \leq i,j \leq n$, $i \neq j$, every alignment $a_i,a_j = 1,\ldots,4$ and every related template $t'_i \in R(t_i)$, $t'_j \in R(t_j)$ we try the sequence σ' where $\sigma'(i) = (t_j,a_j)$, $\sigma'(j) = (t_i,a_i)$ and $\sigma'(k) = \sigma(k)$ otherwise. This corresponds to exchanges of two templates in the sequence combined with replacing them with their relatives and testing all alignments.

Subset Side-Exchange: For every i,j with $1 \leq i < j \leq n$ we try the sequences σ' where $\sigma'(k) = (t_k,o(a_k))$ for $i \leq k < j$ and $\sigma'(k) = \sigma(k)$ otherwise. We define $o(1) = 2$, $o(2) = 1$, $o(3) = 4$ and $o(4) = 3$. This corresponds to a swap of alignment between left and right of templates from all possible consecutive sub-sequences.

Insert: For every i,j with $1 \leq i,j \leq n$, $i \neq j$ and every alignment $a_i = 1,\ldots,4$ we try the sequences $\sigma' = \sigma$, but with t_i removed and reinserted before t_j.

Combine: For templates t and s let $R(t \cup s)$ be the set of templates with all elements from t and s. Then for every i,j with $1 \leq i,j \leq n$, $i \neq j$, every alignment $a_i = 1,\ldots,4$ and every template $t_k \in R(t_i \cup t_j)$ we try the sequences $\sigma' = \sigma$, but with t_i and t_j removed and $\sigma'(i) = (t_k,a_i)$. This corresponds to removing all t_i and t_j and reinserting a template with all the items from both templates at the position of i at all four alignments.

Split: For every i,j with $1 \leq i,j \leq n$, $i \neq j$, every alignment $a_k = 1,\ldots,4$, every $k = 1,\ldots,|t_i|$, every template $t'_i \in R(t_i^1,\ldots,t^{k-1},t^{k+1},\ldots,t_i^{|t_i|})$ and $t_k \in R(t^k)$ we try the sequences $\sigma' = \sigma$, but with $\sigma'(i) = (t'_i,a_i)$ and (t_k,a_k) inserted before the jth element. This corresponds to splitting t_i into two templates, one with $|t_i|-1$ elements and one with one element, and attempting to place the singleton element everywhere in the sequence with every alignment.

Cross-over: For every i, j with $1 \leq i, j \leq n$, $i \neq j$, every $k = 1, \ldots, |t_i|$, every $l = 1, \ldots, |t_j|$, every related template $t_i' \in R(t_i^1, \ldots, t_i^{k-1}, t_i^{k+1}, \ldots, t_i^{|t_i|}, t_j^l)$, every related template $t_j' \in R(t_j^1, \ldots, t_j^{l-1}, t_j^{l+1}, \ldots, t_j^{|t_j|}, t_j^k)$, every alignment $a_i = 1, \ldots, 4$ and every alignment $a_j = 1, \ldots, 4$ we try the sequence $\sigma' = \sigma$, but with $\sigma(i) = (t_i', a_i)$ and $\sigma(j) = (t_j', a_j)$. This corresponds to all exchanges of one of the item-types from t_i with an one item-type from t_j and trying all alignments.

Every time a sequence is tried its templates are applied to their respective alignment to evaluate the objective value of the solution. The local search behaves as a steepest descent algorithm — i.e. all neighboring solutions are evaluated before choosing the change which results in the largest improvement.

Although the neighborhoods are quite large, we are generally able to place more than a 100,000 sequences per second, so in practice many changes may be examined within reasonable times. Experiments showed that not all neighborhoods need to be examined in every iteration. Thus the neighborhoods are examined in the order in which they are described above. If an improving move is found in one neighborhood subsequent neighborhoods are not searched.

Once the heuristic terminates all items which are not within the container boundaries are removed to ensure that the solution is feasible.

7.1 Objective Functions

The primary objective function is to maximize the total profit of bundles loaded within the container boundaries. However, with this objective function, changes are only accepted if they result in increased profit which may be hard to achieve in one single move.

To allow changes that may improve the objective value if more steps are allowed, a set of secondary objective functions is used. At any given time one secondary objective function is active. Changes where the total profit remains the same but increases the active secondary objective are now also accepted. Additionally changes that give the largest improvement in the primary objective value are always preferred, but ties are settled by considering the secondary objective value. Local minimum only occurs when no improvement with respect to neither the primary nor the active secondary objective value can be found.

Our secondary objective functions are as follows:

1. **Minimize total depth.** Minimize z_2 of any bundle.

2-4. **Minimize total depth of item in corner *i*.** Minimize the maximal value of z_2 for any bundle placed at the *i*th corner. This gives rise to four secondary objective functions.

5. **Minimize sum corner-depths.** The sum of the maximal z_2 for each of the four corners is minimized.

6. **Minimize total depth of loaded items.** Similar to 1 but only bundles inside the container are considered.

At any given moment only one secondary objective function is active. Initially 1 is active. When the heuristic reaches a local minimum with respect to the currently active secondary objective function, it switches to the next secondary objective function from list. After 6 it switches to 1 again.

After a specified number of non-profit-improving iterations the heuristic terminates. In practice one can evaluate all secondary objective values for all permutations, so when a local minimum is reached and the secondary objective function is changed, all permutations with respect to the new

secondary objective function have already been evaluated, and the best move with the new secondary objective can be carried out instantly.

7.2 Metaheuristics

The large local search neighborhood is geared towards minor alterations that only shifts items slightly around and can be thought of as a "clean-up" or "tighten" of the quad-wall solution.

Meta-heuristics such as Simulated Annealing [42], Tabu-Search (e.g. [33, 34]) and Guided Local Search [60, 61] were investigated. Unfortunately, none of these proved able to find better solutions than the simple local search scheme within acceptable computational time. An explanation could be that, while the quad-wall heuristic of the previous section determines a good local structure by generating quad-walls, it is harder to locate structured solutions with the mentioned meta-heuristics.

8 Stability-Search

For transportation it is important that items are loaded in such a way, that they are not damaged by dropping to the floor of the container. In this section we consider only large items.

To handle this requirement a second local search heuristic is initiated once the local search of Section 7 is complete. This heuristic starts from the best solution found during the previous step, and it is completely equivalent to the local search in Section 7 except that the objective function has been replaced. We will refer to the heuristic of step as *stability-search*.

In stability search the objective function is to minimize the total profit of unstable templates. When no improving change can be found, the search terminates and templates which violate their stability requirement are removed from the solution and the resulting solution is stable.

Alternatively, one could limit the tree-search and local search heuristics to consider only stable solutions. Preliminary testing, however, showed that the chosen approach is favorable for two reasons. Checking stability is an expensive operation, but solutions are commonly almost stable after the local search step, and resolving the few problems that arise is faster. The second reason is that overall solution quality decreases when only stable solutions could be searched, presumably because the heuristics benefits from passing through unstable solutions.

8.1 Center of Mass

A fundamental part of our stability check is based on centroids. The centroid is the center of mass of an object if it has uniform density. The centroid, R, and signed volume, V, of a tetrahedron with $o = (0,0,0)^t$ and $a,b,c \in \mathbb{R}^3$ as corner points may be calculated as:

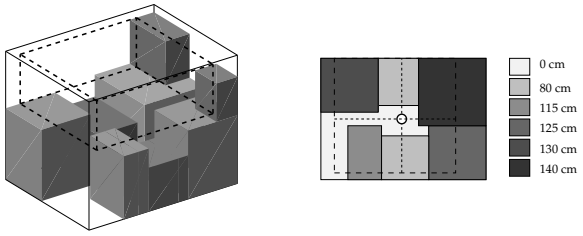
$$R = \frac{a+b+c}{4} \quad , \quad V = \frac{a \cdot (b \times c)}{6}. \tag{4}$$

To calculate the centroid R of a set of n tetrahedra one can use the cumulative expression:

$$R = \frac{\sum_{i=1}^{n} V_i R_i}{\sum_{i=1}^{n} V_i}, \tag{5}$$

where R_i is the centroid and V_i is the volume of each tetrahedron i.

The centroid of a mesh with n triangles each with corner points a_i, b_i, $c_i \in \mathbb{R}^3$, can be determined by decomposing it into n tetrahedra consisting of points o, a_i, b_i, c_i and using the addition formula (5). In principle any point can be chosen as o, since the volume and centroid calculation of each tetrahedron is signed and negative tetrahedra cancels surplus contribution of positive tetrahedra.



(a) (b)

Figure 14: *Stability evaluation. (a) Item to be evaluated is shown as dashed box. (b) Height levels below the item (xz-projection). The circle roughly in the middle is the xz-projection of the centroid of the item. The difference between the maximum height-values of each of the four zones of the centroid must be less than some preset value. Here it is* 140 cm − 115 cm = 25 cm.

8.2 Stability Evaluation

While existing methods in the literature demand that each item is placed on an even surface, this constraint can often be circumvented in practice by use of e.g. polystyrene plates.

Hence we have chosen the following approach. To evaluate if an item is positioned in a stable fashion, we divide the *xz*-projection of its bounding box into four areas around the *xz*-projection of the centroid (which must be within its bounding box). We then determine the maximal height of items within each of the four regions which are below the item considered (see Figure 14). Now we require that the height difference between the maximal height of the four regions must be less than some value h (e.g. 15 cm).

This requirement ensures that the item is properly supported around its centroid and that there is no more than h difference between the height of the supporting items below it. By this strategy items can also be supported by two or more different items below so even "bridges" are acceptable.

9 Medium Sized Items

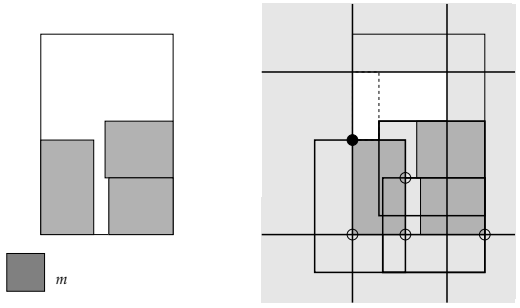
The medium-sized items is the second group we consider. Medium-sized items have identical profit-value and one large item is considered more valuable than any number of medium-sized items. The medium-sized items are boxes which may be rotated 90 degrees around any of the coordinate axis resulting in up-to six different orientations.

Medium sized items are placed using a polynomial time greedy heuristic which will be explained in Section 9.1. A greedy heuristic is adequate for this part of the problem, because items are relatively small and homogeneous. Secondly, an efficient placement method allows us to integrate the greedy heuristic in the heuristics for large items, as will be explained in Section 9.2.

9.1 Greedy Algorithm

The algorithm places types of items, one type at a time, until there is insufficient space for more items. Types are considered in order of decreasing size; Since largest items are often the hardest to place, and small items can appropriately fill the remaining empty holes.

Each type is considered in three steps: 1) Find start position, 2) Determine volume to be filled. 3) Calculate efficient fill. If the volume is insufficient to accommodate all items of the current type, the same type is considered in the next iteration. Otherwise, if all items of this type are placed, or no volume is sufficient for the current type, the algorithm proceeds to the next type.



(a) (b)

Figure 15: *Two-dimensional illustration of NFBs.*

9.1.1 Start Position

The algorithm begins by finding a start-position. Let m be an item of the next item type and let:

$$m^r = [\underline{x}^r_m, \overline{x}^r_m] \times [\underline{y}^r_m, \overline{y}^r_m] \times [\underline{z}^r_m, \overline{z}^r_m], \tag{6}$$

be the bounding-box of m with respect to rotation $r \in \{1,\ldots,6\}$. Let K be the set of bounding-boxes of previously placed items or bundles and define $i \in K$ as follows: $[\underline{x}_i, \overline{x}_i] \times [\underline{y}_i, \overline{y}_i] \times [\underline{z}_i, \overline{z}_i]$. Now, for two boxes i and j the no-fit-box:

$$\mathrm{NFB}(i,j) \quad = \quad [\underline{x}_i - \overline{x}_j, \overline{x}_i - \underline{x}_j] \times [\underline{y}_i - \overline{y}_j, \overline{y}_i - \underline{z}_j] \times [\underline{z}_i - \overline{z}_j, \overline{z}_i - \underline{z}_j],$$

is the set of translations of box j for which j will overlap with box i (like the NFP from Section 2.3).

When we consider an item of type m, $\mathrm{NFB}(i,m^r)$ are created for all previously placed items i and rotations r. Constraints of the form "m not above i" – e.g. if m is too heavy – are easy to handle by expanding $\mathrm{NFB}(i,m^r)$ to the height of the container.

For every triple of i,j,k, if $\mathrm{NFB}(i,m^r)$, $\mathrm{NFB}(j,m^r)$ and $\mathrm{NFB}(k,m^r)$ intersects, we determine the intersection point $(\mathrm{NFB}(i,m^r)_{\overline{x}}, \mathrm{NFB}(j,m^r)_{\overline{y}}, \mathrm{NFB}(k,m^r)_{\overline{z}})^T$ (right side $\mathrm{NFB}(i,m^r)$, top of $\mathrm{NFB}(j,m^r)$ and front of $\mathrm{NFB}(k,m^r)$). If the intersection point is not contained within some $\mathrm{NFB}(l,m^r)$, $l \in K$, it is feasible position of m^r with respect to bounding-boxes of previously placed items. The intersection-point, q, with lexicographically least z, y and x coordinates is chosen as start point for placing boxes of type m. To ensure that items are placed within the container dimensions artificial boxes representing the container sides are introduced.

Figure 15 illustrates this procedure reduced to two dimensions. The current placement is shown in (a) along with the box-type m that we wish to place. The light-shaded rectangles with thick lines in (b) demonstrates the NFBs of the placement and translations of m's lower-left corner to any point within the white area are feasible positions. Filled circles indicate feasible intersection points determined by the algorithm while hollow circles represent infeasible intersection points.

Determination of the start-position takes $O(n^5)$ time, where n is the number of items in the problem instance. However, in practice the algorithm is fast and may be speeded up further by a sweep-line principle, which moves from low to high z-coordinates with breakpoints at $\underline{z}_i$ and $\overline{z}_i$ for $i \in K$. During this traversal, a list of *"active"* boxes, which are the boxes that overlap with the current z-coordinate, are maintained, and one can end the search as soon as the first non-overlapping position has been found.

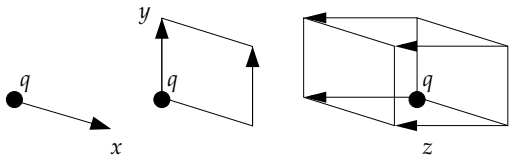


Figure 16: *Volume determination from the start-point q.*

9.1.2 Volume Determination

For the start position q, we determine a suitable box-volume V which will be filled by items of the current type, m. The start position, q, is the lexicographically lowest intersection-point with type m, for some rotation r and q is a feasible position of m^r.

Determination of the extents of V is illustrated on Figure 16. First V's lower-left-back coordinate is set to q; $(\underline{x}_V, \underline{y}_V, \underline{z}_V) = q$. To determine the upper-right-front coordinates $(\overline{x}_V, \overline{y}_V, \overline{z}_V)$ we move right from q parallel to the x-axis until we hit the first box from K. This give us $\overline{x}_V$. We then find the minimal $y \geq \underline{y}_V$ such that the line segment between $\underline{x}_V, y, \underline{z}_V$ and $\overline{x}_V, y, \underline{z}_V$ intersects a box from K and let this be our $\overline{y}_V$. Finally, we determine $\overline{z}_V$ by finding the minimal $z \geq \underline{z}_V$ such that the axis-aligned box with corners q and $(\overline{x}_V, \overline{y}_V, z)$ intersect a box from K.

Determining the volume V can be done in $O(|K|)$ time. An example of the volume V is depicted on Figure 15 as a the dashed rectangle extended from the start-point determined in section 9.1.1.

9.1.3 Volume Filling

To fill V we use a three-level recursive guillotine division of V into smaller volumes. The division considers cuts of V parallel to the x-axis, y-axis and z-axis, such that one direction is chosen for the first level. In each recursion only divisions in directions not used in previous levels are allowed.

We will describe only the x-division since y- and z-divisions are similar. An x-cut divides V into two parts $V'(x') = \{(x,y,z) \in V \mid x \leq x'\}$ and $V''(x') = \{(x,y,z) \in V \mid x \geq x'\}$. Let $W(m^r) = \overline{x}^r_m - \underline{x}^r_m$ and $W(V) = \overline{x}_V - \underline{x}_V$ be the width of volume V and m^r, respectively. Let the height $H(m^r)$, $H(V)$ and depth $D(m^r)$, $D(V)$ be similarly defined.

For a volume V and item m^r, let $C(V, m^r)$ be the number of times m^r can be placed inside V;

$$C(V,m^r) = \left\lfloor \frac{W(V)}{W(m^r)} \right\rfloor \cdot \left\lfloor \frac{H(V)}{H(m^r)} \right\rfloor \cdot \left\lfloor \frac{D(V)}{D(m^r)} \right\rfloor$$

For $r \in \{1,\ldots,6\}$ we consider x-cuts which divides V into volumes $V'(x_i)$ and $V''(x_i)$ for $x_i = \underline{x}_V + i \cdot (w_{m^r})$, $i \in \{0,\ldots,\lfloor \frac{W(V)}{w(m^r)} \rfloor\}$. We then select x' and s such that $C(V'(x'), m^r) + C(V''(x'), m^s)$ are maximal.

We then proceed with the volume $V'(x')$ and $V''(x')$ to consider y-cuts and z-cuts, but such that one side of the new cuts of $V'(x')$ and $V''(x')$ use m^r and m^s respectively. For each of the volumes on the third level only the cut-direction unused at higher levels of the recursion is allowed.

This procedure is repeated with y-cut and a z-cut as initial cut. The cutting sequence and orientation assignment which results in the highest utility of V is used to fill the eight sub-volumes of V.

9.2 Integration

Since a good solution of medium-sized items may depend on the placement of large items, the greedy approach is also integrated with the tree-search and local search heuristics for large items. At any time while running the two heuristics let z'_l be total profit value of the large items and z'_m for the medium-sized items of the currently best known solution. When the heuristics encounter a solution with solution value z_l equal to z'_l for large items, the solution value z_m for medium sized items is calculated. If $z_m > z'_m$ the current best known solution is replaced.

10 Small Items

The last set of items to be placed is the small items. These have a lower precedence than large and medium-sized items. A number of constraints apply to them which will be explained in in Section 10.1. Small items are placed both using a wall-building heuristic, to be described in Section 10.2, and the greedy algorithm of Section 9, as described in Section 10.3.

10.1 Constraints

The constraints that we consider for small items can be divided into three groups; Rotations, robustness and weight.

For each item a specified subset of the six possible 90 degree rotations around coordinate system axis is allowed.

For each item m we let $s(m) \in \{1,\ldots,6\}$ define its robustness and for any two items m_i and m_j, we require $s(m_j) \geq s(m_i)$ for m_i to be be placed on m_j. This ensures that an item is only put on top of a more robust item, so that fragile items are not placed on the bottom of a stack.

Finally, let $g(m)$ be the weight of item m (e.g in Kilograms). Then, for any two items m_i and m_j with $s(m_i) = s(m_j)$, we require that $g(m_i) < g(m_j)$ if m_i is to be placed on top of m_j.

10.2 Wall-building

As previously described the wall-building paradigm of container loading heuristics fills the container in the depth by constructing walls of items. We are using the wall-building approach by Pisinger [53], which improved the heuristic by George and Robinson. Rather than considering just one wall-depth for each wall, Pisinger used a tree-search heuristic which branches on a number of different wall depths. In addition, walls are filled by either horizontal or vertical strips and the heuristic branches on different strip-widths for each strip. Finally each strip is packed optimally by solving a knapsack problem. The heuristic back-tracks once there is no room for additional walls.

The heuristic is used to pack items in a box-volume at the end of the container with dimensions $W \times H \times (D - \max_{l \in L} \overline{z}(l))$ where l is the set of bounding-boxes of templates and medium-sized items from the previous steps. Once the local search and greedy heuristic for large items completes, local search is used an additional time but this time the objective is to minimize $\max_{l \in L} \overline{z}(l)$ so that the input-space for the wall-building heuristic is maximized. This is illustrated on figure 18 where (a) shows a solution with respect to large items which is optimized to the solution of (b) in which the space at the end of the container (bright gray area) is filled by the wall-building heuristic.

The wall-building proceeds similarly to the wall-building in [53], but only vertical strips are allowed to accomodate the constraints described in Section 10.1.

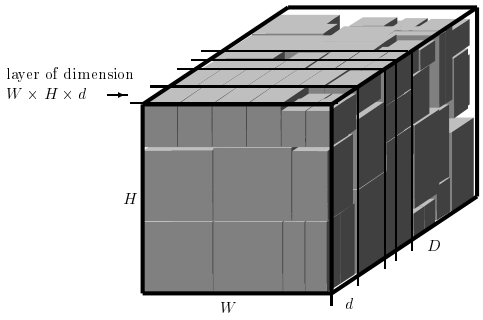


Figure 17: *Figure of wall-building*

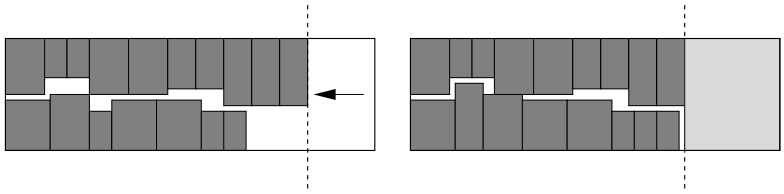


Figure 18: *Maximizing space for small items at the end of the container.*

The filling of vertical strips is again solved using knapsack packing, but once the optimal set of boxes has been determined, the boxes are sorted according to robustness value and weight to ensure that the constraints are not violated.

To handle rotational constraints only allowed rotations are considered during depth determination of layers, width of each layer, and packing in each strip.

10.3 Additional Filling

Although wall-building fills the volume of the container quite well, it is desirable to fill also the volume above the large items. To handle this we simply use the greedy heuristic from Section 9 on each item. When placing an item m robustness constraints are are ensured by expanding the NFBs for items with lower robustness than m to the container height.

11 Experiments

The heuristic has been implemented in C++ (gcc 3.4.2) running under Linux on a AMD 64 3800+ 2.4 GHz. Since the problem has not been studied in prior literature no test-instances were available. Therefore, to demonstrate the capabilities of the heuristic we have constructed a small dataset and report results in this section.

11.1 Dataset

The dataset consists of 40 large, 10 medium sized and 40 small items types.

The large items are shapes of chairs, sofas, chaise lounges and corners. Items are paired together and used in complex templates generated automatically as explained in Section 4. To further mimic the behavior at a furniture producer a set of user-defined templates were determined by a geometric tool. For each item-type less than 6 user-defined templates were created. These templates are comprised of up-to 5 items, but only one type of item. Profit values for large items were selected from the set $\{10, 20, 30\}$, and is correlated to the dimensions of the item. Both the user-defined templates and the profit values mimic the practical use of the heuristic.

Medium-sized box-shaped items were generated with random dimensions taken from the interval 20 to 80. The profit of all medium-sized items was set to 5.

Small box-shaped items were generated with random dimensions from the interval 10 to 80. The set of allowed rotations, robustness values and weight were determined randomly. Weight was set randomly between 1 and 10 kg. The profit of small items was set to 2 for all items.

A total of 61 instances divided among three groups were created: (A) 21 instances containing only large items, (B) 20 instances with large and medium sized items, and (C) 20 instances with large, medium and small items. The most homogeneous instances consists of three large item-types, while the largest and most heterogeneous instances consists of a total of 90 distinct item-types (40 large, 10 medium, 40 small). All instances used container dimensions equal to a 40 ft. high-cube container ($234 \times 239 \times 1185 cm^3$). The high-cube container allows us to better demonstrate the heuristic's ability to place items in multiple layers. Instances were generated so that the total profit was approximately and at least 160, 180 and 200 for (A), (B) and (C) respectively. The characteristics of the instances are reported in Table 1 which is described in the following section. The dataset is available for download at *http://www.diku.dk/˜pisinger/* along with a description of the file-format used.

11.2 Results

Results of the computational experiments for all 61 instances are reported in Table 1. For each instance we report the number of item-types in each of the three group in the column marked $|\mathcal{L}|/|\mathcal{M}|/|\mathcal{S}|$ and the number of items in the columns marked $|I|$, $|L|$, $|M|$, $|S|$. The column labeled "Loaded" under "Items" indicates the total number of items loaded. The columns labeled "Profit" gives the total profit of items (I) and profit of large items (L). The columns labeled "Loaded profit" is the best solution value for all items (I) and for each of the three item groups L, M and S.

Since no real performance measure exists, we have reported the utilization of each of the instances as two different values. The bounding-box utilization in the column labeled "BB" is the percentage of the container volume occupied by the bounding-boxes of the items. Since bounding-boxes can overlap, only non-overlapping volume is accounted for. The mesh-utilization is the percentage of the container occupied by meshes (for large items) and boxes (for medium-sized and small items) and is reported in column "Mesh".

The average results for the three instance groups and all instances are reported in the rows labeled $1-21$, $22-41$, $42-61$ and $1-61$.

On average the bounding-box utilization is 89.2% and the mesh-utilization is 59%. The utilization generally increases as more small items are available. When only large-sized items are considered, the average utilization in percent is 86.5 resp. 56.4 increasing to 89.9 resp. 59.8 when the cargo also contains medium-sized items. However, for the hardest series of instances containing all three types of items and up-to 200 items, volume utilization is 91.3% and 60.9% respectively. This compares well with traditional container loading heuristics where the state-of-the-art is around $90-91\%$ for box-shaped items.

Since the space between pairs of large items cannot be used for quality assurance reasons, the

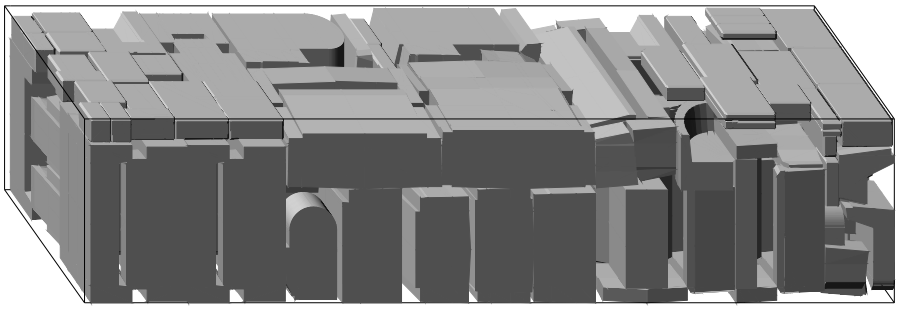


Figure 19: *Solution of instance 61 with 114 items (running time was 134 sec.)*

bounding-box utilization is probably the most correct performance measure, as it accounts for this lost space between the items.

The running time in seconds is reported in column "Time". The average running time is only around 100 seconds, which is highly acceptable for real-life applications. A solution to a typical instance (instance 61) is shown on Figure 19.

12 Conclusion

We have developed a new heuristic for the three-dimensional knapsack container loading problem with irregular shapes. The heuristic consists of several sub-heuristics, which each solves a specific part of the overall problem.

Items are divided into three different groups reflecting their importance, size and complexity. Large items are irregular, represented by three-dimensional triangle-meshes and initially a set of templates which are used by tree-search and local-search heuristics are generated. Medium-sized item are rectangular and placed using a simple greedy heuristic. Small items are rectangular and loaded primarily in the end of the container with a modified wall-building approach. To fill out the remaining parts of the container small items are also placed using a greedy heuristic.

The solution-method is able to find good solutions for problems with hundreds of heterogeneous items within minutes on current common hardware. The bounding-boxes of items occupy more than 89% of the container on average and in instances with items in all three groups the average utilization is 91%. These results compares well with state-of-the-art container loading heuristics that consider only smaller boxes and reach around 90% utilization.

Finally, the algorithm was implemented at a major European furniture producer and improved their utilization by $3-5\%$.

References

[A] Egeblad, J., Nielsen, B. K., and Odgaard, A. Fast neighborhood search for two- and three-dimensional nesting problems. *European Journal of Operational Research*, 183(3):1249–1266, 2007.

[C] Egeblad, J. and Pisinger, D. Heuristic approaches for the two- and three-dimensional knapsack packing problem. *Computers and Operations Research*, 2007. In press (available online).

	Items						Profit		Loaded profit				Utilization		Time
Inst.	$\|\mathcal{L}\|/\|\mathcal{M}\|/\|\mathcal{S}\|$	$\|I\|$	$\|L\|$	$\|M\|$	$\|S\|$	Loaded	I	L	I	L	M	S	BB	Mesh	(sec.)
1	3/0/0	85	85	0	0	54	1600	1600	1270	1270			89.9	59.8	58.6
2		82	82	0	0	62	1600	1600	1360	1360			84.2	60.7	46.8
3		82	82	0	0	74	1600	1600	1520	1520			88	53.2	61.5
4		76	76	0	0	73	1600	1600	1550	1550			81.2	50.6	50.8
5		86	86	0	0	60	1610	1610	1350	1350			85	64.3	67.9
6	6/0/0	78	78	0	0	67	1600	1600	1470	1470			91.6	60.5	76
7		61	61	0	0	53	1610	1610	1490	1490			82.3	53.4	55.7
8		70	70	0	0	55	1610	1610	1400	1400			87.8	62	68.5
9		80	80	0	0	65	1610	1610	1360	1360			89.2	61.4	90.3
10	12/0/0	82	82	0	0	65	1610	1610	1400	1400			86.6	57.6	57.5
11		71	71	0	0	64	1600	1600	1430	1430			87.9	58	75.7
12		75	75	0	0	63	1600	1600	1430	1430			89.1	61.5	89.5
13		62	62	0	0	51	1610	1610	1370	1370			84.6	56.8	46.9
14	24/0/0	77	77	0	0	63	1610	1610	1370	1370			84.6	53.5	120.4
15		83	83	0	0	69	1600	1600	1410	1410			85.1	51.2	168.7
16		78	78	0	0	62	1600	1600	1370	1370			85.6	53.2	82.3
17		84	84	0	0	68	1610	1610	1380	1380			85.3	54.4	104.2
18	40/0/0	84	84	0	0	65	1600	1600	1360	1360			86.8	52.6	71.8
19		80	80	0	0	65	1620	1620	1400	1400			85.9	53.1	81.5
20		79	79	0	0	63	1600	1600	1360	1360			88.9	53.2	145.9
21		82	82	0	0	67	1610	1610	1410	1410			87.7	53.6	129
1-21		78				63.2	1605.24	1605.24	1402.9	1402.9			86.5	56.4	83.3
22	3/1/0	80	63	17	0	73	1815	1730	1675	1590	85		89.8	62.6	44.8
23		106	79	27	0	73	1815	1680	1395	1260	135		89.2	63.2	86.4
24		90	74	16	0	79	1810	1730	1530	1450	80		89.6	61.7	49.2
25		92	63	29	0	79	1805	1660	1575	1430	145		91.2	62.5	63.1
26	6/2/0	117	88	29	0	81	1805	1660	1445	1300	145		86.4	56.1	114.9
27		130	97	33	0	83	1825	1660	1525	1360	165		93.5	67.6	176.6
28		100	78	22	0	88	1800	1690	1570	1460	110		86.7	56.9	77.2
29		99	70	29	0	76	1805	1660	1645	1500	145		90.6	61.6	132.7
30	12/4/0	118	83	35	0	100	1825	1650	1585	1410	175		90.4	59.6	78.2
31		101	71	30	0	89	1820	1670	1560	1410	150		87.1	58.6	52.3
32		107	78	29	0	80	1805	1660	1665	1520	145		91.9	63	140
33		92	70	22	0	72	1820	1710	1520	1410	110		91.2	63.8	67.3
34	24/8/0	108	82	26	0	84	1800	1670	1580	1450	130		89.9	58.7	95.4
35		110	81	29	0	90	1815	1670	1585	1440	145		90.8	59.8	109.2
36		100	79	21	0	84	1805	1700	1525	1420	105		91.7	59.3	129.8
37		109	83	26	0	89	1800	1670	1570	1440	130		91.7	59.8	165.5
38	40/10/0	104	84	20	0	84	1800	1700	1530	1430	100		89.4	56.4	96.4
39		103	85	18	0	81	1800	1710	1430	1340	90		87.4	52	123.8
40		113	83	30	0	88	1800	1650	1540	1390	150		89.7	56.7	168.6
41		101	85	16	0	82	1800	1720	1530	1450	80		89.5	55.6	78
22-41		104	78.8	75.2		82.8	1808.5	1682.5	1549	1423	126		89.9	59.8	102.5
42	3/1/3	114	82	32	104	66	2008	1640	1600	1440	160	0	91.3	58.8	88.2
43		221	108	24	89	100	2018	1720	1316	1170	120	26	88.2	61.5	52
44		194	90	19	85	120	2025	1760	1593	1420	95	78	90	64.7	55.8
45		238	103	32	103	77	2016	1650	1490	1330	160	0	93.8	60.7	109.9
46	6/2/6	167	75	26	66	106	2012	1750	1654	1480	130	44	92.3	64.2	78.6
47		205	92	26	87	99	2014	1710	1716	1550	130	36	89.9	57.5	124.8
48		211	87	31	93	96	2021	1680	1665	1470	155	40	90	62.1	123
49		200	78	36	86	127	2022	1670	1706	1480	180	46	93.4	64.3	105.7
50	12/4/12	181	78	24	79	103	2008	1730	1606	1440	120	46	92.5	64.2	106.5
51		152	72	20	60	101	2020	1800	1566	1410	100	56	89.9	62	57.7
52		165	75	27	63	119	2001	1740	1587	1380	135	72	90.9	64.8	71.1
53		172	75	23	74	113	2013	1750	1587	1390	115	82	91.8	64.4	78.3
54	24/8/24	219	83	33	103	108	2001	1630	1581	1390	165	26	92.7	58.6	73.6
55		196	90	27	79	101	2013	1720	1559	1380	135	44	92.1	60.4	70.6
56		219	92	31	96	130	2017	1670	1487	1260	155	72	88.1	56.9	64.9
57		183	84	22	77	109	2014	1750	1600	1450	110	40	91.8	58.9	134.6
58	40/10/40	191	88	12	91	112	2002	1760	1596	1460	60	76	91.2	56.2	142.7
59		200	88	24	88	120	2006	1710	1590	1410	120	60	92.5	59	177.6
60		194	84	22	88	123	2016	1730	1606	1430	110	66	92	59.5	236.3
61		204	90	17	97	114	2019	1740	1595	1450	85	60	91.7	60.2	134
42-61		191.3	85.7	25.4	85.4	107.2	2013.3	1715.5	1585	1409.5	127	48.5	91.3	60.9	104.3
1-61		123.7				84			1510.5	1411.6	83	15.9	89.2	59	96.5

Table 1: Characteristics and computational results for the 61 test instances.

[1] Alvarez-Valdes, R., Parreno, F., Tamarit, J., 2004. A tabu search algorithm for two-dimensional non-guillotine cutting problems. Tech. Rep. TR07-2004, Universitat de Valencia.

[2] Alvarez-Valdes, R., Parreno, F., Tamarit, J., 2005. A grasp algorithm for constrained two-dimensional non-guillotine cutting problems. Journal of Operational Research Society 56, 414–425.

[3] Art, Jr., R. C., September 1966. An approach to the two dimensional, irregular cutting stock problem. Tech. Rep. 36.Y08, IBM Cambridge Scientific Center.

[4] Beasley, J., 1985. Algorithms for two-dimensional unconstrained guillotine cutting. Journal of the Operational Research Society 36, 297–306.

[5] Bennell, J. A., Dowsland, K. A., 1999. A tabu thresholding implementation for the irregular stock cutting problem. International Journal of Production Research 37, 4259–4275.

[6] Bennell, J. A., Dowsland, K. A., 2001. Hybridising tabu search with optimisation techniques for irregular stock cutting. Management Science 47 (8), 1160–1172.

[7] Bischoff, E., Marriott, M., 1990. A comparative evaluation of heuristics for container loading. European Journal of Operational Research 44, 267–276.

[8] Bischoff, E. E., 2006. Three-dimensional packing of items with limited load bearing strength. European Journal of Operational Research 168, 952–966.

[9] Blazewicz, J., Hawryluk, P., Walkowiak, R., 1993. Using a tabu search approach for solving the two-dimensional irregular cutting problem. Annals of Operations Research 41, 313–325.

[10] Bortfeldt, A., Gehring, H., 1998. Applying tabu search to container loading problems. In: Operations Research Proceedings 1997. Springer, Berlin, pp. 533–538.

[11] Boschetti, M., Hadjiconstantinou, E., Mingozzi, A., 2002. New upper bounds for the two-dimensional orthogonal cutting stock problem. IMA Journal of Management Mathematics 13, 95–119.

[12] Burke, E. K., Hellier, R., Kendall, G., Whitwell, G., 2006. A new bottom-left-fill heuristic algorithm for the two-dimensional irregular packing problem. Operations Research 54 (3), 587–601.

[13] Burke, E. K., Kendall, G., 1999. Applying ant algorithms and the no fit polygon to the nesting problem. In: Proceedings of the 12th Australian Joint Conference on Artificial Intelligence (AI'99). Vol. 1747. Springer Lecture Notes in Artificial Intelligence, pp. 454–464.

[14] Burke, E. K., Kendall, G., 1999. Applying evolutionary algorithms and the no fit polygon to the nesting problem. In: Proceedings of the 1999 International Conference on Artificial Intelligence (IC-AI'99). Vol. 1. CSREA Press, pp. 51–57.

[15] Burke, E. K., Kendall, G., 1999. Applying simulated annealing and the no fit polygon to the nesting problem. In: Proceedings of the World Manufacturing Congres. ICSC Academic Press, pp. 70–76.

[16] Cagan, J., Degentesh, D., Yin, S., 1998. A simulated annealing-based algorithm using hierarchical models for general three-dimensional component layout. Computer Aided Design 30 (10), 781–790.

[17] Caprara, A., Monaci, M., 2004. On the 2-dimensional knapsack problem. Operations Research Letters 1 (32), 5–14.

[18] Davies, A., Bischoff, E., 1999. Weight distribution considerations in container loading. European Journal of Operational Research 114, 509–527.

[19] de Berg, M., Kreveld, M. V., Overmarks, M., Schvarzkopf, O., 2000. Computational Geometry: Algorithms and applications. Springer Verlag, Berlin, Germany.

[20] Dickinson, J. K., Knopf, G. K., 1998. Serial packing of arbitrary 3d objects for optimizing layered manufacturing. In: Intelligent Robots and Computer Vision XVII. Vol. 3522. pp. 130–138.

[21] Dickinson, J. K., Knopf, G. K., 2002. Packing subsets of 3d parts for layered manufacturing. International Journal of Smart Engineering System Design 4 (3), 147–161.

[22] Dowsland, K. A., Dowsland, W. B., Bennell, J. A., 1998. Jostling for position: Local improvement for irregular cutting patterns. Journal of the Operational Research Society 49, 647–658.

[23] Dowsland, K. A., Vaid, S., Dowsland, W. B., 2002. An algorithm for polygon placement using a bottom-left strategy. European Journal of Operational Research 141, 371–381.

[24] Dyckhoff, H., Scheithauer, G., Terno, J., 1997. Cutting and Packing (C&P). In: Dell'Amico, M., Maffioli, F., Martello, S. (Eds.), Annotated Bibliographies in Combinatorial Optimization. John Wiley & Sons, Chichester.

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

[26] Fekete, S. P., Schepers, J., 1997. A new exact algorithm for general orthogonal d-dimensional knapsack problems. In: Algorithms ESA '97, Springer Lecture Notes in Computer Science. Vol. 1284. pp. 144–156.

[27] Fekete, S. P., Schepers, J., 1997. On more-dimensional packing III: Exact algorithms. submitted to Discrete Applied Mathematics.

[28] Fekete, S. P., Schepers, J., van der Veen, J. C., 2006. An exact algorithm for higher-dimensional orthogonal packing. Tech. Rep. cs/0604045, ArXiv Computer Science e-prints.

[29] Gehring, H., Bortfeld, A., 2002. A parallel genetic algorithm for solving the container loading problem. International Transactions in Operational Research 9, 497–511.

[30] Gehring, H., Bortfeldt, A., 1997. A genetic algorithm for solving the container loading problem. International Transactions in Operational Research 4, 401–418.

[31] George, J., Robinson, D., 1980. A heuristic for packing boxes into a container. Computers and Operations Research 7, 147–156.

[32] Gilmore, P., Gomory, R., 1965. Multistage cutting stock problems of two and more dimensions. Operations Research 13, 94–120.

[33] Glover, F., 1989. Tabu search - part 1. ORSA Journal on computing 1 (3), 190–206.

[34] Glover, F., 1990. Tabu search - part 1i. ORSA Journal on computing 2 (1), 4–32.

[35] Gomes, A. M., Oliveira, J. F., 2002. A 2-exchange heuristic for nesting problems. European Journal of Operational Research 141, 359–370.

[36] Gomes, A. M., Oliveira, J. F., 2006. Solving irregular strip packing problems by hybridising simulated annealing and linear programming. European Journal of Operational Research 171 (3), 811–829.

[37] Hadjiconstantinou, E., Christophides, N., 1995. An exact algorithm for general, orthogonal, two-dimensional knapsack problems. European Journal of Operational Research 83, 39–56.

[38] Heckmann, R., Lengauer, T., 1995. A simulated annealing approach to the nesting problem in the textile manufacturing industry. Annals of Operations Research 57 (1), 103–133.

[39] Ikonen, I., Biles, W. E., Kumar, A., Wissel, J. C., Ragade, R. K., 1997. A genetic algorithm for packing three-dimensional non-convex objects having cavities and holes. In: Proceedings of the 7th International Conference on Genetic Algortithms. Morgan Kaufmann Publishers, East Lansing, Michigan, pp. 591–598.

[40] Ikonen, I. T., Biles, W. E., Lewis, J. E., Kumar, A., Ragade, R. K., 1998. Garp: genetic algorithm for part packing in a rapid prototyping machine. In: Gopalakrishnan, B., Murugesan, S. (Eds.), Intelligent Systems in Design and Manufacturing. Vol. 3517 of Proceedings of SPIE. pp. 54–62.

[41] Kellerer, H., Pferschy, U., Pisinger, D., 2004. Knapsack Problems. Springer, Berlin, Germany.

[42] Kirkpatrick, S., Jr., C. D. G., Vecchi, M. P., 1983. Optimization by simulated annealing. Science 220 (4598), 671–680.

[43] Li, Z., Milenkovic, V., 1995. Compaction and separation algorithms for non-convex polygons and their applications. European Journal of Operational Research 84 (3), 539–561.

[44] Lutfiyya, H., McMillin, B., Poshyanonda, P., Dagli, C., 1992. Composite stock cutting through simulated annealing. Journal of Mathematical and Computer Modelling 16 (2), 57–74.

[45] Mack, D., Bortfeldt, A., Gehring, H., 2004. A parallel hybrid local search algorithm for the container loading problem. Internatinal Transaction in Operations Research 11, 511–533.

[46] Martello, S., Monaci, M., Vigo, D., 2003. An exact approach to the strip packing problem. INFORMS Journal on Computing 3 (15), 310–319.

[47] Morabito, R., Arenales, M., 1994. An and/or graph approach to the container loading problem. International Transactions in Operational Research 1, 59–73.

[48] Moura, A., Oliveira, J. F., 2005. A grasp approach to the container-loading problem. IEEE Intelligent Systems 20 (4), 50–57.

[49] Murata, H., Fujiyoshi, K., Nakatake, S., Kajitani, Y., 1996. Vlsi module packing based on rectangle-packing by the sequence pair. IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems 15, 1518–1524.

[50] Ngoi, B. K. A., Tay, M. L., Chua, E. S., 1994. Applying spatial representation techniques to the container packing problem. International Journal of Production Research 32, 111–123.

[51] Oliveira, J. F., Ferreira, J. S., 1993. Algorithms for nesting problems. Applied Simulated Annealing, 255–273.

[52] Oliveira, J. F., Gomes, A. M., Ferreira, J. S., 2000. TOPOS - a new constructive algorithm for nesting problems. OR Spektrum 22, 263–284.

[53] Pisinger, D., 2002. Heuristics for the container loading problem. European Journal of Operations Research 3 (141), 382–392.

[54] Pisinger, D., 2006. Denser packings obtained in O(n log log n) time. INFORMS Journal on Computing to appear.

[55] Ratcliff, M. S. W., Bischoff, E. E., 1998. Allowing for weight considerations in container loading. OR Spektrum 20, 65–71.

[56] Scheithauer, G., 1992. Algorithms for the container loading problem. Operations Research Proceedings 1991, 445–452.

[57] Skiena, S., 1997. Minkowski sum. In: The Algorithm Design Manual. Springer-Verlag, New York, pp. 395–396.

[58] Stoyan, Y. G., Gil, N. I., Scheithauer, G., Pankratov, A., Magdalina, I., 2005. Packing of convex polytopes into a parallelepiped. Optimization 54 (2), 215–235.

[59] Terno, J., Scheithauer, G., Sommerweiss, U., Riehme, J., 2000. An efficient approach for the multi-pallet loading problem. European Journal of Operations Research 2 (132), 371–381.

[60] Voudouris, C., Tsang, E., August 1995. Guided local search. Tech. Rep. CSM-147, Department of Computer Science, University of Essex, Colchester, C04 3SQ, UK.

[61] Voudouris, C., Tsang, E., 1999. Guided local search and its application to the traveling salesman problem. European Journal of Operational Research 113, 469–499.

[62] Wäscher, G., Haussner, H., Schumann, H., 2006. An improved typology of cutting and packing problems. European Journal of Operational ResearchIn Press.

[63] X.Tang, D.F.Wong, 2001. Fast-sp: a fast algorithm for block packing based on sequence pair. In: Asia and South Pacific Design Automation Conference. pp. 521–526.

Submitted. 2008

Placement of two- and three-dimensional irregular shapes for inertia moment and balance

J. Egeblad*

Abstract

We present a heuristic for the problem of placing irregular shapes in two or three dimensions within a container, such that the placement of the shapes is optimized for balance and inertia moment and no two shapes overlap. The heuristic is based on a technique that iteratively removes overlap. The technique was introduced by Faroe et al. [7] for rectangular objects and later generalized by Egeblad et al. [A] to handle irregular shapes. We extend this method and demonstrate its ability to optimize an objective function related to the individual position of each shape. The approach iteratively reduces an augmented objective function which is the sum of balance, inertia moment and overlap and uses the metaheuristic Guided Local Search.
Keywords: packing, nesting, balanced packing, three-dimensional packing

1 Introduction

Problems that involve a placement of two- or three-dimensional shapes within a container or a set of containers are generally referred to as cutting and packing problems (see Wäscher et al. [13] for a survey) or sometimes layout problems (see Cagan et al. [2] for a survey). While researchers have thoroughly investigated problems such as bin-packing, knapsack-packing, strip-packing and component layout problems, methods for ensuring overall stability of the placements have received less attention. In this paper we investigate the problem of packing shapes while ensuring balance and reduce inertia moment of the placement.

Balance must be ensured by minimizing the difference between the global center of gravity of the items and a specified target center of gravity. The moment of inertia, which describes the force required to turn the items around an axis going through the center of gravity, must also be minimized.

The problem occurs in transportation problems where balance is important. Ships must be loaded such that the likeliness of capsizing is minimal. Trucks should not tip and the weight should be distributed evenly on the wheels. Cars must be designed such that the engine and other elements are placed in balance and with minimal inertia of moment. Airplanes and space exploration vehicles must be in balance and the moment of inertia should be minimal to minimize fuel consumption.

In this paper we view the problem of achieving balance solely as a post-processing problem which is applied to solutions arising from typical transportation problems such as container-loading, knapsack-packing or bin-packing. Our objective is to determine optimal positions of items within container boundaries, since we assume that the selection of shapes have occurred a priori.

We consider both the two- and three-dimensional variants of the problem. For two dimensions we consider polygonal shapes and for three dimensions polyhedra. Our technique is described mainly for three dimensions, and we only detail the simpler two-dimensional variant in cases where the two differ substantially.

*jegeblad@diku.dk, Department of Computer Science, University of Copenhagen, DK-2100 Copenhagen Ø, Denmark

Given a container shape C and a set of items P described as polygons (polyhedrons), where each item $i \in P$ has an associated weight g_i, we wish to solve the following problem:

$$\begin{array}{l} \text{minimize } F(\mathcal{P}), \\ s.t. \\ \quad \begin{array}{lrcll} \text{(I)} & \mathcal{P}(i) \cap \mathcal{P}(j) & = & 0 & \forall i, j \in P \\ \text{(II)} & \mathcal{P}(i) \cap C & = & \mathcal{P}(i) & \forall i \in P. \end{array} \end{array} \tag{1}$$

Where $\mathcal{P}$ is a placement of P and $\mathcal{P}(i)$ is i translated and rotated as described by $\mathcal{P}$. The constraints (I) and (II) ensure that no two shapes overlap and all shapes are placed within the container boundaries. Notice that if the container C is a box then $C = [X_0, X_1] \times [Y_0, Y_1] \times [Z_0, Z_1]$.

Assume that the desired center of gravity is located in the coordinate system origin, $(0,0,0)$, then the objective function F is defined as:

$$F(\mathcal{P}) = \alpha \sum_{i=1}^{n} g_i (x_i^2 + y_i^2 + z_i^2) + \beta G_x(\mathcal{P})^2 + \gamma G_y(\mathcal{P})^2 + \delta G_z(\mathcal{P})^2$$

where x_i, y_i and z_i are the positions of the center of gravity of shape $i \in P$ in the placement $\mathcal{P}$, and G_x, G_y and G_z are defined as follows:

$$G_x(\mathcal{P}) = \frac{\sum_{i \in P} g_i x_i}{\sum_{i \in P} g_i} \quad G_y(\mathcal{P}) = \frac{\sum_{i \in P} g_i y_i}{\sum_{i \in P} g_i} \quad G_z(\mathcal{P}) = \frac{\sum_{i \in P} g_i z_i}{\sum_{i \in P} g_i}$$

The first summation term of the objective function describes the moment of inertia while the last three terms are used to describe the square distance between origin (desired center of gravity) and actual center of gravity in each of the three directions. The values α, β, γ and δ are weights which can be used to adjust the importance of individual terms in the resulting solution. We refer to this problem as the *Balanced Weight Placement Problem (BWPP)* and the two- and three-dimensional variants as respectively BWPP-3D and BWPP-3D.

An alternative formulation is to minimize the moment of inertia such that the center of gravity is confined to a target region around the origin given by $[L_x, U_x] \times [L_y, U_y] \times [L_z, U_z]$. This formulation is achieved with β, γ and δ set to 0 and adding the additional constraints:

$$\begin{array}{ll} \text{(III)} & L_x \leq G_x(\mathcal{P}) \leq U_x \\ \text{(IV)} & L_y \leq G_y(\mathcal{P}) \leq U_y \\ \text{(V)} & L_z \leq G_z(\mathcal{P}) \leq U_z \end{array} \tag{2}$$

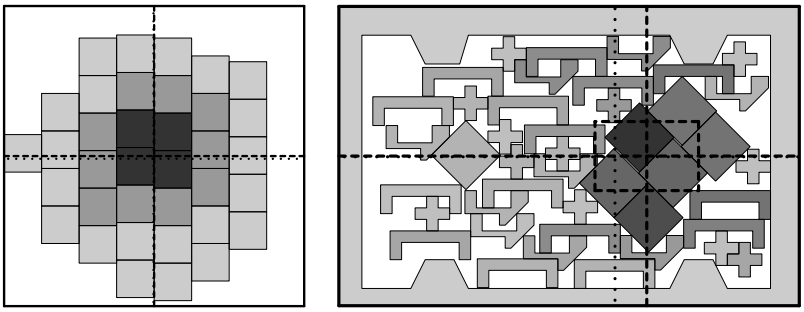
We refer to this formulation as the *Constrained Balanced Weight Placement Problem (CBWPP)* and the two- and three-dimensional variants as CBWPP-2D and CBWPP-3D.

Two example solutions to the two-dimensional problem are depicted in Figure 1. Here each item has been colored according to weight; The darker the color the heavier the item. The desired center of gravity is marked by dashed lines. While the actual center of the solution is marked by dotted lines; The actual center and the desired center are very close in the two examples.

The problem we consider is NP-hard, as stated by the following theorem.

Theorem 4. *The Balanced Weight Placement Problem and Constrained Balanced Weight Placement Problem are NP-Hard.*

Proof. Let DBWPP-2D be the decision variant of BWPP-2D and let it be defined as follows; Given a value k, a container shape C and a set of items P described as polygons (polyhedrons), where each



(a) (b)

Figure 1: *Example solutions. (a) 64 1x1 squares in 10x10 container (BWPP-2D). (b) Irregular shapes within irregular container (CBWPP-2D).*

item $i \in P$ has an associated weight g_i, determine if we can can find a placement $\mathcal{P}$ with $\mathcal{F}(\mathcal{P}) \leq k$ which is feasible under the constraints (I-II).

To prove that BWPP-2D is NP-Hard we show that DBWPP-2D is NP-complete by reduction from the Set Partition Problem (SPP) which is known to be NP-complete (see e.g. Cormen et al. [3]). The SPP is given as follows; Given a set of n items each with size a_i, let $M = \sum_{i=1}^{n} a_i$ and determine if a subset S of the items can be found such that:

$$\sum_{i \in S} a_i = \sum_{i \notin S} a_i = \frac{M}{2}. \tag{3}$$

To show that DBWPP-2D is NP-complete we define the reduction function from an instance of SPP to an instance, I, of DWBPP-2D as follows: Create items with center of gravity in their middle and dimensions $[-\frac{a_i}{2}, \frac{a_i}{2}] \times [-0.5, 0.5]$ all with weight $g_i = a_i$ (mass density 1), let the container be defined as $C = [-\frac{M}{4}, \frac{M}{4}] \times [-1, \infty]$, set $\alpha = 0$, $\beta = 1$, $\gamma = 1$ and $k = 0$.

We first show that a solution of the SPP also constitute a solution to I. Assume that the rectangles from S are numbered $1, \ldots, m$ and the remaining rectangles are numbered $m+1, \ldots, n$, this means that $\sum_{i=1}^{m} a_i = \sum_{i=m+1}^{n} a_i = \frac{M}{2}$. Then assign coordinates to the rectangles as $x_i = \sum_{j=1}^{i-1} a_j + \frac{a_i}{2} - \frac{M}{2}$ and $y_i = -\frac{1}{2}$ for $i = 1, \ldots, m$. and $x_i = \sum_{j=m+1}^{i-1} a_j + \frac{a_i}{2} - \frac{M}{2}$ and $y_i = -\frac{1}{2}$ for $i = m+1, \ldots, n$. Denote this placement $\mathcal{P}$ and observe that $\mathcal{P}$ is a feasible placement of BWBPP-2D due to the dimensions of the rectangles (see figure 2 (a)). Now we can calculate the center of gravity of $\mathcal{P}$:

$$G_x(\mathcal{P}) = \frac{1}{M}\left(\sum_{i=1}^{m} a_i \left(\sum_{j=1}^{i-1} a_j + \frac{a_i}{2} - \frac{M}{4}\right) + \sum_{i=m+1}^{n} a_i \left(\sum_{j=m+1}^{i-1} a_j + \frac{a_i}{2} - \frac{M}{4}\right)\right) \tag{4}$$

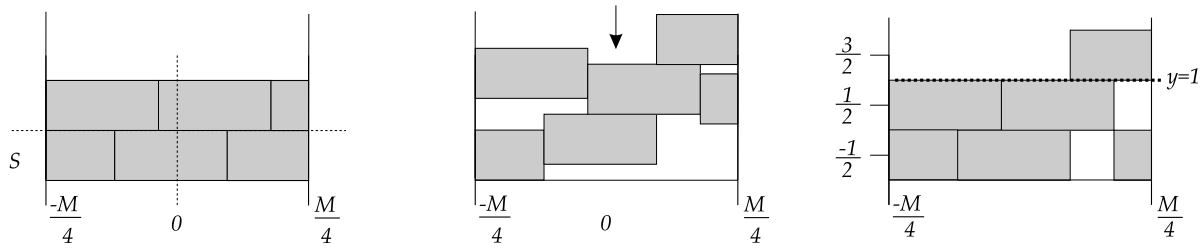
$$= \frac{1}{M}\left(\frac{M^2}{8} - \frac{M^2}{8} + \frac{M^2}{8} - \frac{M^2}{8}\right) = 0. \tag{5}$$

$$G_y(\mathcal{P}) = \frac{1}{M}\left(-\sum_{i=1}^{m} a_i \frac{1}{2} + \sum_{j=m+1}^{i-1} a_i \frac{1}{2}\right) = \frac{1}{M}\left(-\frac{M}{2} + \frac{M}{2}\right) = 0. \tag{6}$$

This shows that a solution to SPP constitute a solution to the instance I.

Conversely, assume we have a solution, $\mathcal{P}$, to I. Then we know that

$$(\sum_{i=1}^{n} a_i x_i)^2 + (\sum_{i=1}^{n} a_i y_i)^2 = 0. \tag{7}$$



(a) (b) (c)

Figure 2: *Illustrations for Theorem 4 (see text). (a) Solution from SPP used to generate a solution to DBWPP-2D. (b) Placement $\mathcal{P}$ from the proof of Theorem 4; All rectangles are slid down. (c) Placement $\mathcal{P}'$ where all rectangles have y-coordinate $y_i = \{-\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \ldots\}$.*

Therefore $\sum_{i=1}^{n} a_i x_i = 0$ and $\sum_{i=1}^{n} a_i y_i = 0$. Now create a new placement, $\mathcal{P}'$ by sliding all rectangles downwards in the y-direction as far as possible without creating overlap (see figure 2 (b)). Each rectangle i now has coordinates $x'_i = x_i$ and $y'_i \leq y_i$. Note that the x-portion of the center of gravity $\mathcal{P}$ is the same as $\mathcal{P}'$ since we do not alter the x-coordinates.

Since all rectangles have height 1 and we cannot move any rectangle to a lower y-coordinate, we know that all $y'_i \in \{-\frac{1}{2}, \frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \frac{7}{2}, \ldots\}$ (see figure 2 (c)). Let S be the group of rectangles with $y'_i = -\frac{1}{2}$ and let S' be the set of all other rectangles.

The total area occupied by the rectangles in S which all have $y'_i = -\frac{1}{2}$ cannot exceed $\frac{M}{2}$ since they all have height one and the width of the container is $\frac{M}{2}$ so $\sum_{i \in S} a_i \leq \frac{M}{2}$. For a rectangle i in $\mathcal{P}$ which extends beyond the horizontal line $y = 1$ ($y_i + \frac{1}{2} > 1$), we have either slid i down such that $y'_i < y_i$ or we have $y'_i \geq \frac{3}{2}$ since $y_i > \frac{1}{2}$ and every rectangle is 1 unit high. Assume at least one such rectangle exists then we have:

$$0 = \sum_{i=1}^{n} a_i y_i = -\sum_{i \in S}^{n} a_i \frac{1}{2} + \sum_{i \in S'} a_i y_i > -\frac{M}{2} + \sum_{i \in S'} a_i \frac{1}{2} = -\frac{M}{2} + \frac{M}{2} = 0, \tag{8}$$

which shows that the assumption that $y_i > \frac{1}{2}$ must be wrong and no rectangle can therefore stretch beyond the horizontal line $y = 1$ in $\mathcal{P}$.

Therefore we can divide the rectangles in $\mathcal{P}$ in two groups $S = \{i \in \{1, \ldots, n\} | y_i = -\frac{1}{2}\}$ and $S' = \{i \in \{1, \ldots, n\} | y_i = -\frac{1}{2}\}$ and we have that

$$\sum_{i \in S} a_i = \sum_{i \in S'} a_i = \frac{M}{2}, \tag{9}$$

since the width of the container is $\frac{M}{2}$. S and S' can now be used for a solution to SPP. The reduction may be done in polynomial time, and the resulting solution to DBWPP-2D represents a valid solution to SPP. A solution to DBWPP-2D may be verified in polynomial time, and therefore DBWPP-2D is NP-complete and BWPP-2D is NP-hard. A similar reasoning shows that CBWPP-2D is NP-hard. □

1.1 Contribution

In this paper we present a heuristic for determining the optimal placement of a set of two- or three-dimensional items with respect to (1). Our solution method is based on work by Egeblad et al. [A] on

the strip-packing problem of irregular shapes. This method is briefly summarized in Section 4, while we detail how to accommodate the objective function and additional constraints in Section 5.

The primary element of the method by Egeblad et al. [A] is an algorithm which is able to find the minimal overlap translation of one shape among a placement of shapes in polynomial time, and we will show how this algorithm is extended to accommodate the objective function $\mathcal{F}(P)$.

The remainder of the paper is organized as follows. In Section 2 we describe relevant work from the literature that considers similar problem formulations. We explain how to determine the center of mass of each individual item in Section 3.

Finally we present computational results for both the two- and three-dimensional variants with respect to both rectangular and irregular shapes in Section 6.

2 Related Work

We briefly summarize the solution methods that consider balanced loadings here.

Amiouny et al. [1] consider the problem of placing items in an airplane or in a truck with two axles. It is explained that although airplanes need not be loaded such that they are in complete balance along the long axis, it is generally favorable to do so, since the pilot will otherwise have to compensate with increased fuel consumption. In some countries there are limits to the maximal allowed weight on each axle of a truck, and they argue that if one wishes to minimize the maximal weight on each axle the center of gravity of the load must be located halfway between the two axles. The problem they consider is a one-dimensional problem and they present two approximation algorithms and a heuristic for solving the problem. Both approximation algorithms have running time $O(n \log n)$ and guarantee that the center of gravity of the load does not deviate from the target point by more than half the size of the largest box. An alternative heuristic for this problem was later proposed by Mathur [10].

Fasano [8] describes a knapsack variant of the problem where one is given a set of items and must select and place a subset of items that maximizes the utilization of the container, given that the center of gravity must fall within a given convex domain. The problem considered is a three-dimensional problem with boxes and "tetris"-like shapes. His solution method is based on Mixed Integer Programming. No results are reported.

Wodziak and Fadel [14] describe a genetic algorithm for two-dimensional placement of rectangles with applications in the area of truck-loading.

Teng et al. [11] describe a method for optimizing the placement of parts in space satellites. The satellite is modeled as a section of a cone which is split in two parts by a plate. Parts must be mounted on either side of the plate such that the moment of inertia is minimized. First the two-dimensional problems of placing parts on the plate is solved, then the appropriate position of the plate is determined. The items they consider consists of both cylinders and boxes. The paper does not detail their placement approach, but they report that they use a Broyden-Fletcher-Goldfarb-Shanno (BFGS) variable metric unconstrained minimization method.

Gehring and Bortfeldt [9] consider center of gravity during container loading. Their heuristic uses the wall-building paradigm commonly used for container loading problems, where the container is divided into smaller parts (walls) in its depth and items in each part are placed by solving a two-dimensional packing problem. Each item must be contained fully within the wall it has been assigned to. Once the packing heuristic completes, the walls of the best solution are interchanged, mirrored or rotated by 180 degrees to move the center of gravity of the overall placement towards the center of the container.

A similar approach was used by Davies and Bischoff Davies and Bischoff [4] for trucks. However,

their walls were deeper and they utilized random search to optimize for the center of gravity. Eley [5] also use this principle. He reports that as little as 3-4 to walls may be sufficient to achieve acceptable solutions.

3 The Center of Mass

We consider only items with their mass distributed uniformly throughout their volume. If an object has uniform mass, its center of mass is referred to as its centroid. For rectangles and boxes the centroid coincides with the geometric center. For more complex items this is not the case.

The standard way to calculate the centroid $c = (x_c, y_c) \in \mathbb{R}^2$ of a polygon with n points, $(x_i, y_i) \in \mathbb{R}^2$, $i \in \{1, \ldots, n\}$, is.

$$\begin{aligned} x_c &= \frac{1}{6A}\sum_{i=1}^{n}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i), \\ y_c &= \frac{1}{6A}\sum_{i=1}^{n}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i), \\ A &= \frac{1}{2}\sum_{i=1}^{n}(x_i y_{i+1} - x_{i+1} y_i), \end{aligned}$$

where $(x_{n+1}, y_{n+1}) = (x_1, y_1)$ and A is the area of the polygon.

For a tetrahedron i with corner points $(0,0,0)^t$ and $a, b, c \in \mathbb{R}^3$ its signed volume, V_i, and centroid, R_i, can be calculated as:

$$V_i = \frac{a \cdot (b \times c)}{6}. \qquad R_i = \frac{a + b + c}{4}.$$

To calculate the centroid of a set of n tetrahedra one can use the cumulative expression:

$$R = \frac{\sum_{i=1}^{n} V_i R_i}{\sum_{i=1}^{n} V_i}, \tag{10}$$

where R_i is the centroid and V_i is the volume of each tetrahedron i.

If a polyhedron is described by a set of n triangles in space, each with corner points a_i, b_i, $c_i \in \mathbb{R}^3$, its centroid is equal to the centroid of the n tetrahedra consisting of the four points $(0,0,0)^t$, a_i, b_i, c_i, $i = 1, \ldots, n$ which can be calculated using (10).

4 Finding a non-overlapping placement

In this section we briefly summarize the work by Faroe et al. [7] and Egeblad et al. [A] which is the foundation of our heuristic.

The solution process of Faroe et al. [7] and Egeblad et al. [A] revolves around solving the decision variant of a packing problem in which a non-overlapping placement of a set of shapes must be found within given container dimensions. The method uses the metaheuristic Guided Local Search (GLS) by Voudouris and Tsang [12] to control the solution process.

Egeblad et al. [A] begin by defining the total overlap in a placement $\mathcal{P}$ as,

$$G(\mathcal{P}) = \sum_{i,j \in P} \text{overlap}(i, j),$$

where overlap(i, j) is the area or volume of overlap of shapes i and j in placement $\mathcal{P}$. In order to find a solution to the placement problem, $G(\mathcal{P})$ is minimized by iteratively reducing the overlap. In each iteration shapes are translated in one of the two or three axis-parallel directions to the position that results in least overlap. This procedure terminates once $G(\mathcal{P}) = 0$ for the current placement $\mathcal{P}$, since this placement has no overlap and is a solution.

The metaheuristic GLS by Voudouris and Tsang [12] is applied to help escape local minima. GLS uses an augmented objective, and Faroe et al. [7] formulate it as follows:

$$\min \qquad H(\mathcal{P}) = \sum_{p,q \in P} \text{overlap}(p,q) + \lambda \sum_{p,q \in P} I(p,q)\rho_{p,q}$$

where λ is a fine-tuning parameter for the heuristic, $I(p,q) = 1$ if shapes p and q overlap in placement $\mathcal{P}$ and 0 if not, and $\rho_{p,q}$ is a penalty term. At each local minimum the value $\rho_{p,q}$ is increased for the pair of shapes p and q for which the value $\frac{\text{overlap}(p,q)}{\rho_{p,q}+1}$ is largest. After $\rho_{p,q}$ has been increased, the solution process commences with the modified objective function. The consequence of increasing the penalty term for p and q, is that the heuristic will prefer a placement where p and q no longer overlap.

For two dimensions, the minimal overlap translation for a polygon p is determined by an algorithm with running time $O(mn \log mn)$, where m is the number of edges from p and n is the number of edges belonging to other polygons. The algorithm works by considering every pair of edges from p and all other shapes. For each pair of an edge from p and and edge from another shape a piecewise quadratic function describes the size of the area between the two edges for all horizontal translations of p. Each piece of the quadratic function represents an interval of translations of p. The piecewise quadratic functions are added together to represent a piecewise quadratic function that describes the total area of overlap. Penalties are accounted for by adding each to the piecewise quadratic in the intervals that corresponds to a positive overlap between the shapes of that penalty. The combined overlap and penalties form one piecewise quadratic function $H(\mathcal{P}(p,t))$ which describes the overlap and penalty value for each t translation of p along the x-axis relative to placement $\mathcal{P}$:

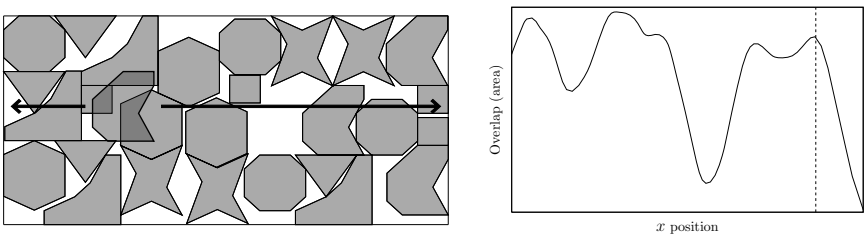
$$H(\mathcal{P}(p,t)) = \sum_{j \in P} \text{overlap}(p(t),q) + \lambda \sum_{j \in P} I(p(t),q)\rho_{pq}. \tag{11}$$

Here $p(t)$ is p translated t units along the x-axis and $\mathcal{P}(p,t)$ is the placement $\mathcal{P}$ with shape $p(t)$ instead of p. The minimal overlap position is determined by traversing the piecewise quadratic function $H(\mathcal{P}(p,t))$ from low to high values of t and analyzing each segment for minima. Only minima corresponding to translations within the container boundaries are considered and t for the global such minimum is selected as best translation. An example of $H(\mathcal{P}(p,t))$ is illustrated on Figure 3.

A similar approach works for three dimensional triangle-mesh polyhedra. Here the volume between each triangle from p and each of the triangles from all other polyhedra can be represented by a piecewise cubic function in the amount t p is translated. Combined these constitute a complete piecewise cubic function which describes the volume of overlap and any penalties for all values of t.

5 Solving the Balance Problem

The procedure described in Section 4 was used by Faroe et al. [6] to optimize the placement of rectangles (modules) in Final Placement of VLSI design. Here the objective was to find a non-overlapping placement of the modules with minimal wire-length. To solve this problem Faroe et al. [6] minimized a sum of wire-length and overlap as objective function. Since overlap and wire-length minimization in VLSI design are counteracting objectives the procedure by Faro et al. slowly increased the weight



(a) (b)

Figure 3: *Illustration of the procedure by Egeblad et al. [A]. (a) An overlapping placement containing one shape. (b) The total overlap is described as a piecewise quadratic function of the horizontal translation of the shape.*

of overlap in the objective function from 0 towards ∞ such that overlap minimization increasingly dominates the objective and a legal placement with little wire-length would eventually be found. The paradigm described does not have any requirements with respect to the initial placement and may therefore be well suited for improving and legalizing an initial, possibly infeasible, placement.

Our approach follows the method by Faroe et al. [6] and considers the augmented gravity objective function:

$$\text{minimize} \qquad E(\mathcal{P}) = H(\mathcal{P}) + \omega F(\mathcal{P}). \tag{12}$$

$E(\mathcal{P})$ is a weighted sum of overlap, balance and inertia in placement $\mathcal{P}$. Starting from some initial placement the objective value is iteratively reduced by translating shapes parallel to one of the two or three coordinate axes to the position that has minimal value E. Like the previous methods we use GLS to ensure that the local search can overcome local minima.

The value of ω changes during the solution process. Initially ω is set to a very large number so only balance and moment of inertia is optimized. We now slowly decrease the value of ω, so that in the beginning of the solution process the balance objective, $F(\mathcal{P})$ is the most important objective, and the heuristic converges towards non-overlapping placements. Let ω_i be the value of ω after i translations, then we set $\omega_{i+1} = \psi\omega_i$. Empirically we found that $\psi = 0.999$ is a good compromise between fast convergence and high solution quality.

Once an feasible non-overlapping placement has been determined, the current value of ω is multiplied by k and the procedure continues with the new objective function. Initially k is set to 2. If the objective value of the feasible solution, w.r.t balance, is equal to the last found feasible solution k is doubled until a new feasible placement is found at which point k is reset to 2. This is done to avoid cyclic behavior.

The development of ω and the balance inertia objective value for the instance `ep2-100-U-R-75` (to be presented in Section 6) is depicted in figure 4. Here the the best found objective value (objective) is shown as a function of time along with the logarithm of the value of ω (omega). The first feasible solution is found after 5 seconds and it is demonstrated and the heuristic continuously find better solutions during the first 45 seconds.

The minimal overlap algorithm by Egeblad et al. is adapted to accommodate the augmented objective function E. The algorithm by Egeblad et al. searches a piecewise quadratic function for global minimum. By adding $F(\mathcal{P})$ to this quadratic function, which account for the overlap, we can ensure that the minimum found by the algorithm is the translation of a polygon (polyhedron) that reduces E the most.

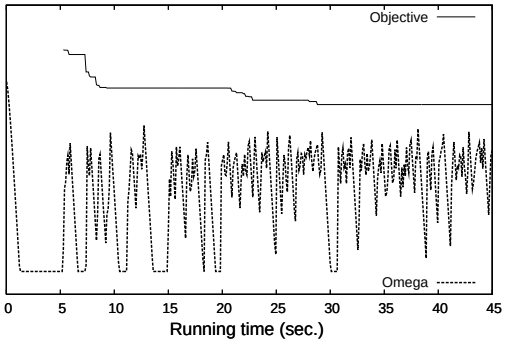


Figure 4: *The solution process of ep2-100-U-R-75.*

The value of the balance and inertia objective function F for the placement $\mathcal{P}(p,t)$ can be determined as:

$$F(\mathcal{P}(p,t)) = F(\mathcal{P}) + \alpha g_p(t - x_p) + \beta \left(G_x + \frac{g_p(t-x_p)}{\sum_{i \in P} g_p} \right)^2, \tag{13}$$

where G_x is the x-coordinate of the center of gravity for $\mathcal{P}$. This shows that $F(\mathcal{P}(p,t))$ can be evaluated in constant time for any t if $F(\mathcal{P})$ is known and that $F(\mathcal{P}(p,t))$ is a quadratic function in t. Let $E((\mathcal{P}(p,t)) = \omega H(\mathcal{P}(p,t)) + F(\mathcal{P}(p,t))$ be the full augmented objective value of the placement $\mathcal{P}(p,t)$ comprising of overlap, penalties and balance and inertia. In two dimensions $E(\mathcal{P}(p,t))$ is the sum of a quadratic and a piecewise quadratic function and may therefore be described as a piecewise quadratic function. In three dimensions $E(\mathcal{P}(p,t))$ is piecewise cubic. The same analysis performed on $H(\mathcal{P}(p,t))$ as described in the end of Section 4 may be performed on $E(\mathcal{P}(p,t))$ to determine the minimal translation with respect to the augmented gravity objective function E. Note that (13) shows that F is quadratic for translations of p that are parallel to the x-axis, however, a similar argument holds for the other two coordinate system axis.

5.1 Target Region Constraints

To handle the CBWPP version of the problem where the center of gravity must be kept within a rectangular region of the container (constraints $(III) - (V)$), all items are placed initially such that their center of gravity falls within this region.

Let $G_x(\mathcal{P})$ be the current x-coordinate of the center of gravity, then for a shape p the x-coordinate of the center of gravity of the placement can be described as a function of the translation t of p along the x-axis:

$$G_x(\mathcal{P}(p,t)) = G_x(\mathcal{P}) + \frac{g_p(t-x_p)}{\sum_{i \in P} g_i}, \tag{14}$$

where x_p is the x-coordinate of p in $\mathcal{P}$. Since $G_x(\mathcal{P}(p,t))$ is linear we may determine values t_0 and t_1 such that $G_x(\mathcal{P}(p,t_0)) = L_x$ and $G_x(\mathcal{P}(p,t_1)) = U_x$. We now only allow translations t of i such that $t_0 \leq t \leq t_1$, i.e. translations which retain the center of gravity within the target region.

5.2 Initial weight

The initial value of ω, ω_0, should be carefully selected to match the problem instance. We assume that the heuristic starts with some initial random placement, $\mathcal{P}_0$. We now set:

$$\omega_0 = \sum_{p \in P} \text{area(p)}/F(\mathcal{P}_0),$$

where area(p) is the area of polygon (polyhedron) p. If we expect the total area to be in the same order of magnitude as the initial overlap, then this choice of ω ensures that the contribution of $\mathcal{H}(\mathcal{P}_0)$ and $\mathcal{F}(\mathcal{P}_0)$ in (12) are of same order of magnitude. Since the overlap is rarely equal to the total area, the contribution of $\mathcal{F}(\mathcal{P}_0)$ is also slightly larger than that of $\mathcal{H}(\mathcal{P}_0)$. This strategy was found empirically to weigh $\mathcal{F}$ high enough during the initial steps to find good solutions.

6 Computational Experiments

The necessary changes described above were added to the implementation of the heuristic described in Egeblad et al. [A]. The heuristic was implemented in C++ (gcc 4.1.3) and tests were conducted on a computer with two quad-core Intel Xeon 5355 2.66 GHz processors and 8 GB RAM. The implementation did not use any form of parallelism and therefore did not take advantage of the multi-cpu multi-core system.

To demonstrate the capabilities of the heuristic we conduct experiments in both two- and three-dimensions and for both BWPP and CBWPP. We also compare results of the heuristic with random legal placements.

6.1 Two dimensions

To test the two-dimensional variant of the heuristic we use a total of 16 instances which are described in Table 1. Both rectangular and irregular shapes were used for the computational experiments. The type of shapes are described in the column 'Shapes' and are either rectangular (Rect.) or irregular (Irre.). The column 'n' describes the number of shapes in each instance.

We have selected 10 instances with irregular shapes which are commonly used for The Two-Dimensional Nesting Problem and are described in Egeblad et al. [A], and 3 instances with rectangular shapes used for Two-Dimensional Orthogonal Knapsack Packing Problems which are described in Egeblad and Pisinger [C].

The dimensions of the container were adjusted appropriately for both sets of instances. For the instances from The Two-Dimensional Nesting Problem the strip-length was set to 105% of the best strip-length reported in Egeblad et al. [A]. For the knapsack instances the rectangles of the best found subset was used and the container dimensions were expanded to 105% in all directions. The modification of container dimensions were made to give the heuristic adequate freedom to optimize for balance and moment of inertia without too much emphasis on searching for non-overlapping legal placements.

In addition three more instances were created. Two instances were created to test the heuristic's ability to handle odd-sized containers (one with rectangular shapes and one with irregular shapes), these are called Ship and Car. Finally an instance containing 64 10x10 squares within a 80x80 container (64-squares) was created to test the heuristic's ability to handle simple problems.

The weight, g_i, of each item i within each instance was set to area$(i) \cdot r_i$ where r_i was chosen uniform randomly within the interval $[\frac{1}{2}, 2]$.

Instance	n	Shapes	Target (x/y)	Target size (%)	Small			Large		
					W	H	Util.	W	H	Util.
64-squares	64	Rect.	$\frac{1}{2}/\frac{1}{2}$	20	80	80	100.0	120	120	44.4
ep2-50-D-R-75	41	Rect.	$\frac{1}{2}/\frac{1}{2}$	20	82.95	166.95	89.8	118.5	238.5	44.0
ep2-100-U-R-75	69	Rect.	$\frac{2}{3}/\frac{1}{2}$	20	536.6	1073	88.5	766.5	1533	43.3
ep2-200-D-R-75	157	Rect.	$\frac{1}{2}/\frac{1}{2}$	20	155.4	310.8	88.0	222	444	43.1
Albano	24	Irre.	$\frac{1}{2}/\frac{1}{2}$	20	10453	5071.5	80.5	14934	7350	38.9
Dagli	30	Irre.	$\frac{2}{3}/\frac{1}{2}$	30	63.0	62.1	77.8	90	90	37.6
Fu	12	Irre.	$\frac{1}{2}/\frac{1}{2}$	40	33.6	39.33	82.0	48.0	57.0	39.6
Mao	20	Irre.	$\frac{2}{3}/\frac{2}{3}$	20	1818	2677.5	77.2	2598	3825	37.8
Marques	24	Irre.	$\frac{1}{2}/\frac{1}{2}$	30	81.9	107.64	81.6	117.0	156.0	39.4
Shapes0	43	Irre.	$\frac{2}{3}/\frac{1}{2}$	30	63.0	41.4	61.2	90	60	29.6
Shapes2	28	Irre.	$\frac{1}{2}/\frac{1}{2}$	20	28.65	15.52	73.6	22.5	40.5	35.6
Shirts	99	Irre.	$\frac{2}{3}/\frac{2}{3}$	20	65.20	41.4	80.0	60	94.5	38.1
Swim	48	Irre.	$\frac{1}{2}/\frac{1}{2}$	20	6494.3	5953.32	65.8	8628	9277.5	31.8
Trousers	64	Irre.	$\frac{2}{3}/\frac{1}{2}$	30	251.50 5	81.765	82.5	364.5	118.5	39.8
Ship	83	Rect.	$\frac{2}{3}/\frac{1}{2}$	20	58.0	600.0	89.3	-	-	-
Car	43	Irre.	$\frac{2}{3}/\frac{1}{2}$	20	52	80	66.6	-	-	-

Table 1: Instances used for two-dimensional experiments.

To test different target center of gravity positions, the instances have their center of gravity target set to either $\frac{1}{2}$ or $\frac{2}{3}$ of both x- and y-dimensions of the container. The target center position is written in column 'Target' of Table 1.

For CBWPP-2D we restrict the center of gravity to a region around the center of gravity target. The dimensions of this rectangular region is described as a percentage of the container dimensions in the column marked 'Target size'.

This set of 16 instances we refer to as the set of small container instances. The set of small container instances with a rectangular container was copied to create a set of 14 large instances. The dimensions of the large instances was set to 150% their original/best found value but the weight of each item was kept intact. This allow us to compare the results of the heuristic for compact and less compact placements. The dimensions of small and large instances are in the columns 'Small' and 'Large' where 'W' is the width, 'H' is the height and 'Util.' is the utilization of the instance under the given container dimensions.

Results of the test instances are listed in Table 2. For each instance we list the results with respect to moment of inertia and center of gravity after 30, 120 and 300 seconds. The resulting distance to the target center of gravity is listed in the columns entitled 'COG' and the resulting moment of inertia in the columns entitled 'Moment'. The distance to the target center of gravity is reported as percentage deviation between the diagonal length of the instance ($\sqrt{W^2+H^2}$) and the euclidian distance from the solution center of gravity to the target center of gravity.

We do not report the total value of $F(\mathcal{P})$, but it may be extrapolated using container dimensions and the target center of gravity deviation.

Each instance is tested using 3 different formulations; First for solving the BWPP-2D, letting the heuristic run 300 seconds with α, β, γ and δ all set to 1. This way the moment of inertia is optimized without sacrificing good solutions to the center of gravity. The results of this test are listed in the first 3 columns of Table 2 with incrementally best result reported after 30, 120 and 300 seconds. Secondly, 300 seconds with the CBWPP-2D formulation are reported in the column entitled '300 s. (CBWPP-

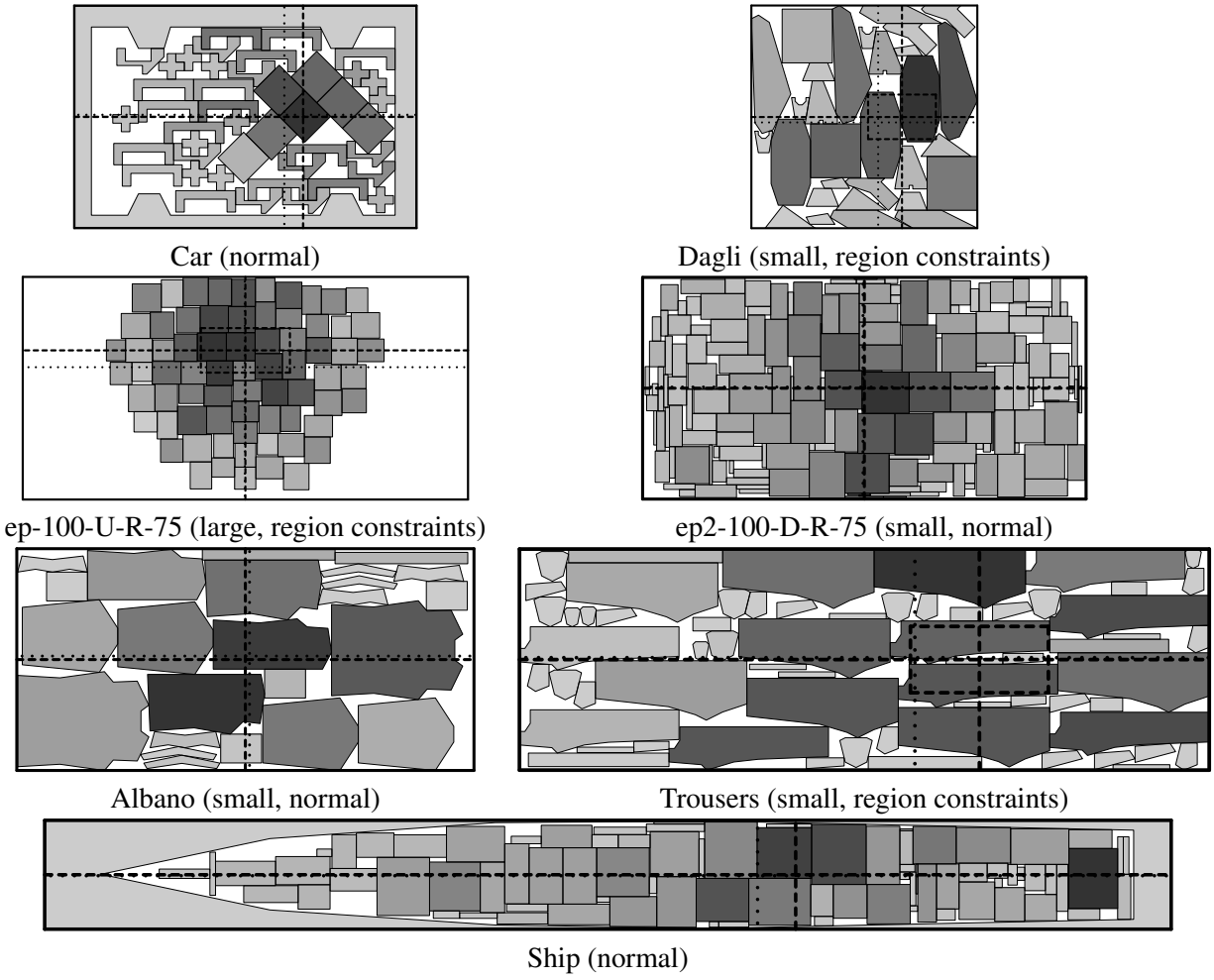


Figure 5: *Best solution for selected instances.*

2D)'. And finally in the column entitled 'Random' we report the solutions achieved by legalizing random initial solutions using the procedure by Egeblad et al. [A]. Each instance was tested 10 times with 10 different random seeds leading to 10 different initial placements for each formulation. We report the average of the best found result within the designated time-limit. For some instances, not all 10 seeds led to legal placements within the time limit and in this case the number of successful runs is reported as '(x)' in front of the associated results, and the average is taken over the successful runs.

Inspection of the results reveals that for the small container instances the improvement of the combined objective function is on average 4.11 % between the first 30 and 120 seconds and 2.30% during the last 180 seconds. Similarly for the large container instances the improvements are 5.00 % between the results after 30 and 120 seconds and 1.16% during the last 180 seconds. For the odd container instances the improvement between the first 30 and 120 seconds is 2.95 % while it is 1.92 % during the last 180 seconds. This shows that most improvements occur during the first 120 seconds and relatively little improvement occurs during the last 180 seconds.

For the small container instances the resulting moment of inertia is surprisingly 0.43 % worse

Instance Name	30 s.		120 s.		300 s.		300 s. (region)		Random	
	COG	Moment	COG	Moment	COG	Moment	COG	Moment	COG	Moment
Small Container Instances										
64-squares	0.53	$1.69 \cdot 10^5$	0.02	$1.67 \cdot 10^5$	0.01	$1.66 \cdot 10^5$	0.79	$1.68 \cdot 10^5$	0.05	$2.44 \cdot 10^5$
Albano	2.42	$1.11 \cdot 10^9$	1.78	$1.06 \cdot 10^9$	1.56	$1.05 \cdot 10^9$	1.91	$1.06 \cdot 10^9$	3.31	$1.50 \cdot 10^9$
Dagli	9.34	$1.08 \cdot 10^5$	7.40	$1.03 \cdot 10^5$	6.93	$1.01 \cdot 10^5$	7.53	$1.01 \cdot 10^5$	11.9	$1.25 \cdot 10^5$
Fu	3.99	$2.46 \cdot 10^4$	3.26	$2.33 \cdot 10^4$	3.09	$2.24 \cdot 10^4$	(9) 3.14	$2.31 \cdot 10^4$	4.43	$2.90 \cdot 10^4$
Marques	1.71	$1.69 \cdot 10^5$	1.53	$1.63 \cdot 10^5$	1.54	$1.61 \cdot 10^5$	1.21	$1.62 \cdot 10^5$	3.96	$2.28 \cdot 10^5$
Shapes0	(9) 11.2	$7.14 \cdot 10^4$	10.3	$6.87 \cdot 10^4$	8.96	$6.67 \cdot 10^4$	10.1	$6.47 \cdot 10^4$	13.7	$8.36 \cdot 10^4$
Shapes2	2.05	$1.19 \cdot 10^4$	2.05	$1.14 \cdot 10^4$	1.89	$1.11 \cdot 10^4$	1.90	$1.11 \cdot 10^4$	2.31	$1.41 \cdot 10^4$
Shirts	11.7	$1.15 \cdot 10^5$	11.7	$1.08 \cdot 10^5$	11.1	$1.04 \cdot 10^5$	12.1	$1.05 \cdot 10^5$	15.0	$1.27 \cdot 10^5$
Swim	(1) 1.25	$7.96 \cdot 10^8$	(8) 1.66	$8.13 \cdot 10^8$	(9)1.12	$8.03 \cdot 10^8$	1.59	$7.93 \cdot 10^8$	1.84	$8.40 \cdot 10^8$
Troousers	(9) 12.6	$1.25 \cdot 10^6$	11.6	$1.19 \cdot 10^6$	10.7	$1.15 \cdot 10^6$	(6) 8.63	$1.10 \cdot 10^6$	14.9	$1.44 \cdot 10^6$
ep2-50-D-R-75	1.05	$3.88 \cdot 10^7$	1.21	$3.85 \cdot 10^7$	0.73	$3.82 \cdot 10^7$	1.13	$3.83 \cdot 10^7$	1.70	$4.59 \cdot 10^7$
ep2-100-U-R-75	5.85	$6.50 \cdot 10^{10}$	6.03	$6.40 \cdot 10^{10}$	5.91	$6.36 \cdot 10^{10}$	4.45	$6.73 \cdot 10^{10}$	8.42	$7.97 \cdot 10^{10}$
ep2-200-D-R-75	0.64	$4.20 \cdot 10^8$	0.52	$4.02 \cdot 10^8$	0.63	$3.96 \cdot 10^8$	0.57	$3.96 \cdot 10^8$	1.37	$5.09 \cdot 10^8$
Large container Instances										
64-squares	0.50	$1.65 \cdot 10^5$	0.40	$1.61 \cdot 10^5$	0.28	$1.59 \cdot 10^5$	0.60	$1.59 \cdot 10^5$	10.7	$5.71 \cdot 10^5$
Albano	0.67	$8.15 \cdot 10^8$	0.60	$8.07 \cdot 10^8$	0.67	$8.02 \cdot 10^8$	0.76	$8.02 \cdot 10^8$	2.48	$1.80 \cdot 10^9$
Dagli	2.04	$9.08 \cdot 10^4$	1.88	$8.88 \cdot 10^4$	1.95	$8.80 \cdot 10^4$	2.99	$8.78 \cdot 10^4$	7.81	$1.98 \cdot 10^5$
Fu	0.81	$1.89 \cdot 10^4$	0.72	$1.87 \cdot 10^4$	0.77	$1.86 \cdot 10^4$	3.15	$1.87 \cdot 10^4$	0.24	$4.25 \cdot 10^4$
Marques	0.58	$1.50 \cdot 10^5$	0.00	$1.49 \cdot 10^5$	0.00	$1.49 \cdot 10^5$	0.72	$1.48 \cdot 10^5$	0.03	$2.99 \cdot 10^5$
Shapes0	0.92	$4.46 \cdot 10^4$	0.78	$4.37 \cdot 10^4$	0.79	$4.35 \cdot 10^4$	1.38	$4.34 \cdot 10^4$	9.83	$1.38 \cdot 10^5$
Shapes2	0.62	$8.51 \cdot 10^3$	0.55	$8.43 \cdot 10^3$	0.57	$8.31 \cdot 10^3$	0.66	$8.29 \cdot 10^3$	1.86	$1.96 \cdot 10^4$
Shirts	3.35	$9.04 \cdot 10^4$	1.48	$6.97 \cdot 10^4$	1.22	$6.66 \cdot 10^4$	1.58	$6.66 \cdot 10^4$	9.20	$2.24 \cdot 10^5$
Swim	0.94	$8.74 \cdot 10^8$	0.68	$7.19 \cdot 10^8$	0.52	$7.02 \cdot 10^8$	0.89	$6.96 \cdot 10^8$	2.26	$1.50 \cdot 10^9$
Troousers	3.94	$7.12 \cdot 10^5$	4.09	$6.53 \cdot 10^5$	3.90	$6.48 \cdot 10^5$	4.53	$6.36 \cdot 10^5$	8.85	$2.05 \cdot 10^6$
ep2-50-D-R-75	0.44	$2.96 \cdot 10^7$	0.52	$2.95 \cdot 10^7$	0.40	$2.94 \cdot 10^7$	0.59	$2.95 \cdot 10^7$	4.04	$6.19 \cdot 10^7$
ep2-100-U-R-75	1.80	$5.02 \cdot 10^{10}$	1.91	$4.95 \cdot 10^{10}$	1.95	$4.94 \cdot 10^{10}$	2.98	$4.98 \cdot 10^{10}$	8.48	$1.38 \cdot 10^{11}$
ep2-200-D-R-75	0.07	$3.28 \cdot 10^8$	0.06	$3.22 \cdot 10^8$	0.07	$3.17 \cdot 10^8$	0.10	$3.17 \cdot 10^8$	0.55	$8.54 \cdot 10^8$
Irregular Container Instances										
Ship	(9) 5.98	$4.36 \cdot 10^8$	(9) 5.91	$4.33 \cdot 10^8$	(9) 5.87	$4.25 \cdot 10^8$	6.25	$4.25 \cdot 10^8$	10.44	$5.87 \cdot 10^8$
Car	7.30	$6.81 \cdot 10^4$	6.74	$6.55 \cdot 10^4$	6.42	$6.47 \cdot 10^4$	6.72	$6.35 \cdot 10^4$	13.86	$1.04 \cdot 10^5$

Table 2: Results of two-dimensional experiments.

on average when the heuristic solves for CBWPP-2D instead of BWPP-2D and the average distance between the target center of gravity and actual center of gravity is also larger. For the large container instances the moment of inertia is 0.2 % better and for the odd container instances it is 0.84 % better when the heuristic optimizes for a specific center of gravity region. This shows that that little is gained by optimizing for a target region for the center of gravity. A likely cause is that there is a strong correlation between solutions with low moment of inertia and with the center of gravity close to the target center of gravity. Another possible cause is that the limited target region limits the type of changes the heuristic can conduct during local search.

The difference between considering the objective function during optimization and random placements can be seen when comparing the results of the heuristic after 300 seconds with results of the random placements. For the small container instances the value of the combined objective function is 31.6 % higher for the random placements than for those produced by the heuristic. The moment of inertia is 28.0 % higher and the quadratic distance between the actual center of gravity and the target center of gravity is 294.9 %. For the large container instances the values are respectively 178.0 %, 162.9 % and 15170 %, and for the odd container instances 68.8 %, 49.6 % and 292.3 %. This shows that random placements are far from optimal with respect to balance and moment of inertia.

The objective value for the large container instances are on average 17.44% better than for the small container instances for unconstrained solutions. Since the same set of items were used, but with larger container dimensions this shows that the heuristic behaves only slightly better even if the container dimensions are larger, which should simplify the problem of finding feasible solutions and give greater freedom for positions of items.

Example solutions are shown in figure 5. the target center of gravity is indicated as the intersection of the dashed lines, while the actual center of gravity is indicated as the intersection of the dotted lines. Target regions are indicated as dashed rectangles for the solutions of the problems with a target region.

6.2 Three dimensions

A similar set of tests were conducted for a three-dimensional variant of the heuristic. Three instances with rectangular items and two instances with irregularly shaped items were used for testing. The rectangular instances are based on solutions to knapsack problems reported in [C] while the irregular instances are based on the best found solutions to the three-dimensional strip-packing problems reported by Egeblad et al. [B]. As for the two-dimensional instances, the input containers for the instances with rectangular items were expanded by 5 % in every direction (small container instances) and by 50 % in every direction (large container instances). The instances are listed in Table 3. For the instances with irregular shapes the two fixed container dimensions were kept intact and height (strip-length) was set to 110% of the average height reported in the Egeblad and Pisinger [C]. For ep3-60-C-R-50 and stoyan3 the target center of gravity is set close to the bottom center of the container; $(\frac{1}{2}W, \frac{1}{2}H, \frac{1}{3}L)$ for small sized containers and $(\frac{1}{2}W, \frac{1}{2}H, \frac{1}{4}L)$ for large sized containers. The dimensions of the instances are reported in the columns entitled W (width), H (height) and L (length).

The results of running the heuristic are reported in Table 4 using the same terminology as was used for the two-dimensional results. Several example solutions are shown in Figure 6.

The average improvement from 30 to 120 seconds of running time is respectively 3.42 % and 0.82 % for the small and large container instances, and respectively 4.03 % and 0.41 % from 120 to 300 seconds. This shows, that although little improvement occurs for the large instances during the last 180 seconds, there is still substantial improvement for the small container instances.

The value of moment of inertia is 2.14% and 0.88 % better for respectively the small and large container instances, when the heuristic optimizes for CBWPP-3D rather than BWPP-3D. This matches

Instance	n	Shapes	Target	Target size (%)	Small				Large			
					W	H	L	Util.	W	H	L	Util.
ep3-60-C-R-50	46	Rect.	Bottom	20	128.1	257.25	128.1	68.9	241.5	483.0	241.5	31.7
ep3-40-L-C-90	24	Rect.	Center	20	169.05	338.1	169.05	69.1	183.0	367.0	183.0	25.8
ep3-60-C-R-90	44	Rect.	Center	20	205.8	411.0	205.8	68.6	294.0	588.0	294.0	28.9
stoyan2	12	Irre.	Center	20	15.0	20.9	14.0	38.4	22.5	32.25	21.0	11.6
stoyan3	25	Irre.	Bottom	25	15.0	31.9	16.0	40.27	22.5	46.95	24.0	13.3

Table 3: Instances used for three-dimensional experiments.

the results from the two-dimensional experiments which showed that little is gained with the CBWPP formulation.

For the small container instances the difference between the random solutions and the heuristic solutions is 28.9 %, 25.1 % and 354 % for respectively the full objective function, the moment and center of gravity components of the objective functions. This shows that the three-dimensional random solutions are suboptimal and matches the results for two dimensions.

As for the two-dimensional instances, the instances with large container dimensions only have an objective value which is 15.24 % better on average than the instances with small container dimensions. This, again, shows that the heuristic only works slightly worse when it is easier to find a feasible placement.

7 Conclusion

We have described a simple approach for solving the two- and three-dimensional placement problem of shapes with respect to balance and inertia moment. The objective is to minimize the deviation between actual center of gravity and a target center of gravity as well as the inertia of moment of the shapes.

The solution method uses a technique previously used by Egeblad et al. [A] that finds feasible placements of shapes, by minimizes the amount of overlap in the placement. The objective function we minimize here a weighted sum of overlap and the balance objective. Initially the method focuses on ensuring proper balance, and slowly increases the weight of overlap in the objective function. Once a placement with no overlap has been found, the method again reduces the weight of the overlap term, to continue the search for other feasible placements that are better with respect to balance.

Our method expands on the work by Egeblad et al. [A] to efficiently search the local search neighborhood consisting of axis-aligned translations. We show that the balance terms of the objective function for all axis-aligned translations of a single shape can be described by a quadratic function. This quadratic function is added to the piece-wise quadratic function which describes overlap and enables us to find the minimal overlap translation for a shape efficiently.

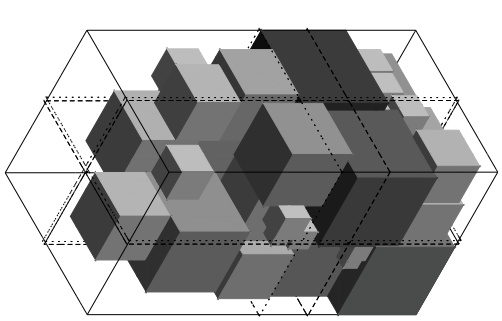
Good results are returned within minutes even for instances with more than 150 rectangles, however our implementation is general and quality results may be obtained even faster for rectangular instances with an implementation specifically designed for rectangular placement. The approach is, to our knowledge, the only method capable of optimizing the center of gravity and moment of inertia for irregular shapes in both two- and three-dimensions.

7.1 Acknowledgements

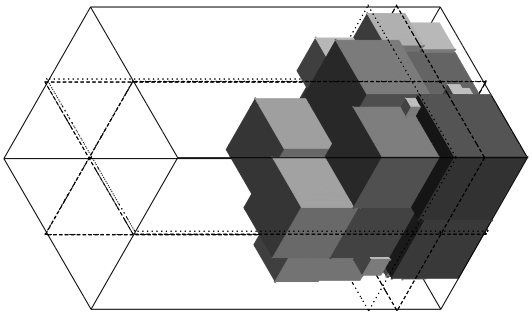
The author wishes to thank Benny K. Nielsen and David Pisinger for inspiring and fruitful discussions.

Instance Name	30 s.		120 s.		300 s.		300 s. (region)		Random	
	COG	Moment	COG	Moment	COG	Moment	COG	Moment	COG	Moment
Small Container										
ep3-60-C-R-50	6.34	$9.16 \cdot 10^{10}$	6.19	$9.15 \cdot 10^{10}$	6.16	$7.70 \cdot 10^{10}$	(4) 6.33	$7.60 \cdot 10^{10}$	13.1	$1.21 \cdot 10^{11}$
ep3-40-L-C-90	0.90	$2.19 \cdot 10^{10}$	0.81	$2.19 \cdot 10^{10}$	0.89	$2.19 \cdot 10^{10}$	1.22	$1.50 \cdot 10^{10}$	3.05	$2.59 \cdot 10^{10}$
ep3-60-C-R-90	2.51	$2.44 \cdot 10^{11}$	3.00	$2.43 \cdot 10^{11}$	2.56	$2.41 \cdot 10^{11}$	2.85	$2.14 \cdot 10^{11}$	5.46	$3.04 \cdot 10^{11}$
stoyan2	(9) 7.28	$1.92 \cdot 10^{4}$	4.37	$2.08 \cdot 10^{4}$	4.76	$2.06 \cdot 10^{4}$	4.03	$1.57 \cdot 10^{4}$	5.42	$2.08 \cdot 10^{4}$
stoyan3	(4) 8.22	$5.79 \cdot 10^{4}$	10.2	$6.64 \cdot 10^{4}$	9.26	$6.38 \cdot 10^{4}$	(9) 6.94	$5.29 \cdot 10^{4}$	6.52	$5.95 \cdot 10^{4}$
Large Container										
ep3-60-C-R-50	0.76	$1.57 \cdot 10^{11}$	0.75	$1.56 \cdot 10^{11}$	0.78	$1.56 \cdot 10^{11}$	1.09	$9.16 \cdot 10^{10}$	7.11	$2.84 \cdot 10^{11}$
ep3-40-L-C-90	0.60	$6.68 \cdot 10^{10}$	0.54	$6.68 \cdot 10^{10}$	0.54	$6.68 \cdot 10^{10}$	0.70	$1.77 \cdot 10^{10}$	5.35	$6.75 \cdot 10^{10}$
ep3-60-C-R-90	4.17	$3.20 \cdot 10^{11}$	4.14	$3.22 \cdot 10^{11}$	4.14	$3.21 \cdot 10^{11}$	4.89	$2.76 \cdot 10^{11}$	16.6	$1.03 \cdot 10^{12}$
stoyan2	≈ 0.95	$3.24 \cdot 10^{4}$	0.94	$2.93 \cdot 10^{4}$	0.89	$2.70 \cdot 10^{4}$	0.89	$8.61 \cdot 10^{3}$	3.50	$4.49 \cdot 10^{4}$
stoyan3	2.41	$3.02 \cdot 10^{4}$	2.38	$3.01 \cdot 10^{4}$	2.38	$3.01 \cdot 10^{4}$	2.68	$2.93 \cdot 10^{4}$	10.1	$1.17 \cdot 10^{5}$

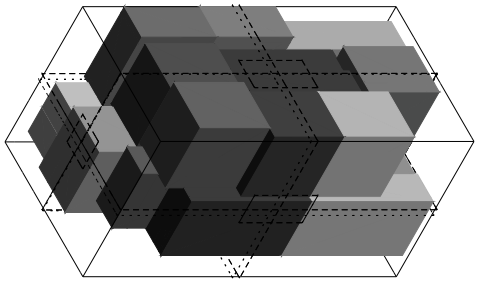
Table 4: Results of three-dimensional experiments.



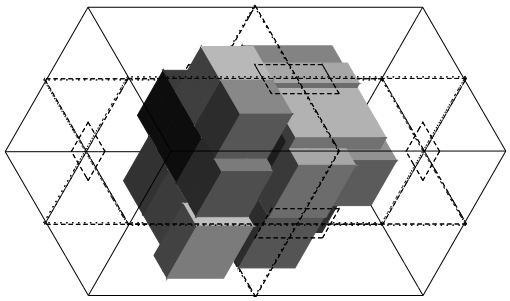
ep3-60-C-R-50 (small, normal)



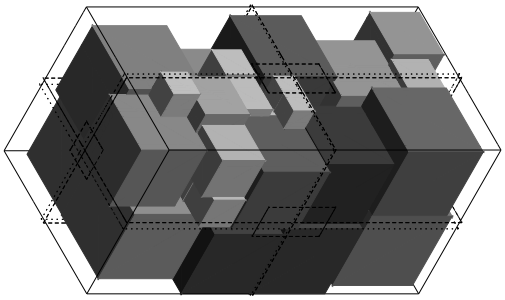
ep3-60-C-R-50 (large, normal)



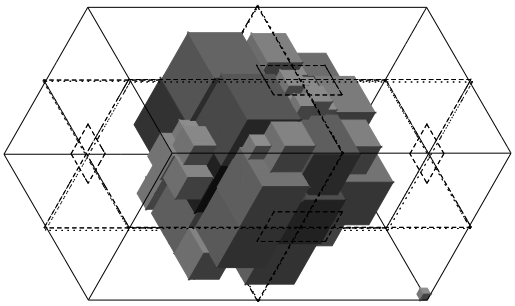
ep3-40-L-C-90 (small, region constraints)



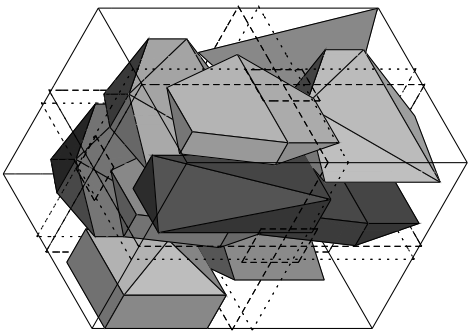
ep3-40-L-C-90 (large, region constraints)



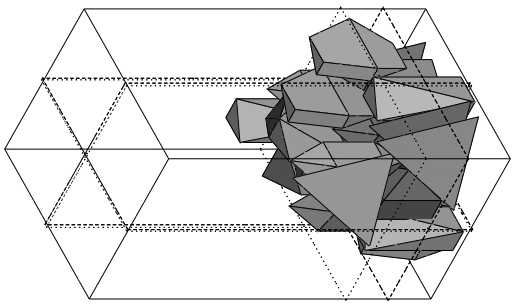
ep3-60-C-R-90 (small, normal)



ep3-60-C-R-90 (large, region constraints)



stoyan2 (small, region constraints)



stoyan3 (large, normal)

Figure 6: *Best solution for selected three-dimensional instances (rotated 90 degrees).*

References

[A] J. Egeblad, B. K. Nielsen, and A. Odgaard. Fast neighborhood search for two- and three-dimensional nesting problems. *European Journal of Operational Research*, 183(3):1249–1266, 2007.

[B] J. Egeblad, B. K. Nielsen, and M. Brazil. Translational packing of arbitrary polytopes. *CGTA. Computational Geometry: Theory and Applications*, 2008. accepted for publication.

[C] J. Egeblad and D. Pisinger. Heuristic approaches for the two- and three-dimensional knapsack packing problem. *Computers and Operations Research*, 2007. In press (available online).

[1] S. V. Amiouny, III J. J. Bartholdi, J. H. Vande-Vate, and J. Zhang. Balanced loading. *Oper. Res.*, 40(2):238–246, 1992. ISSN 0030-364X.

[2] J. Cagan, K. Shimada, and S. Yin. A survey of computational approaches to three-dimensional layout problems. *Computer-Added Design*, 34:597–611, 2002.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

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

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

[6] O. Faroe, D. Pisinger, and M. Zachariasen. Guided local search for final placement in vlsi design. *Journal of Heuristics*, 9(3):269–295, 2003. ISSN 1381-1231.

[7] 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.

[8] G. Fasano. A mip approach for some practical packing problems: Balancing constraints and tetris-like items. *4OR*, 2(2):161–174, 2004.

[9] H. Gehring and A. Bortfeldt. A genetic algorithm for solving the container loading problem. *International Transactions in Operational Research*, 4:401–418, 1997.

[10] K. Mathur. An integer-programming-based heuristic for the balanced loading problem. *Operations Research Letters*, 22(1):19–25, 1998.

[11] H. Teng, S. Sun, D. Liu, and Y. Li. Layout optimization for the objects located within a rotating vessel a three-dimensional packing problem with behavioral constraints. *Comput. Oper. Res.*, 28(6):521–535, 2001. ISSN 0305-0548.

[12] C. Voudouris and E. Tsang. Guided local search. Technical Report CSM-147, Department of Computer Science, University of Essex, Colchester, C04 3SQ, UK, August 1995.

[13] G. Wäscher, H. Haussner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 2006. In Press.

[14] J. Wodziak and G. Fadel. Packing and optimizing the center of gravity location using a genetic algorithm.

Draft. 2008

Three-dimensional Constrained Capsule Placement for Coarse Grained Tertiary RNA Structure Prediction

J. Egeblad* L. Guibas† M. Jonikas‡ A. Laederach§

Abstract

We present a novel technique to solve problems in which a set of capsules are to be placed within an arbitrary container. The placements may be further constrained by inter-capsule distances and angles. We also study optimization variants of the problem where capsules must be placed such that some objective is optimized. The model has applications within coarse grained RNA tertiary structure prediction, which we model as a network of inter-connected capsules which represent alpha-helices that must be placed within some molecular surface. In our model such a surface is represented by a triangle-mesh. The problem is solved heuristically via an iterative local search method which utilizes the metaheuristic Guided Local Search. The local search neighborhood consists of all axis aligned translations of a single capsule and is searched efficiently using a polynomial time algorithm. Results show that the method is capable of finding feasible placements of networks consisting of up to 50 capsules under compact conditions. Experiments with a model of an RNA-molecule consisting of 7 helices and a molecular envelope return helical placements with an average RMSD of 20 Å to the crystal structure.
Keywords: Cylinder packing, RNA prediction, Irregular Packing

1 Introduction

Knowing the three-dimensional structure of RNA molecules is vital for studying and determining their function. While x-ray crystallography may be used to determine the structure experimentally, this is a time and labour consuming process and methods which can accurately predict the structure computationally may help scientist to uncover the mysteries of the molecules.

The secondary structure of an RNA molecule consists of a list of base pairs. The tertiary structure consists of a set of three-dimensional coordinates for each atom. Among the successful methods for secondary structure prediction is the dynamic programming method Mfold [25] and the probabilistic method ContraFold [5]. Prediction of the tertiary structure of RNA may be done based on the base-paired regions of a secondary structure prediction.

In this paper we consider a coarse grained model for RNA structure prediction, in which we assume that the proper secondary structure may be accurately predicted, and that a set of helical regions can be deduced from it. In our model we treat each helical region as a rigid body. We let each

*Computer Science Department, University of Copenhagen, DK-2100 Cph Ø, Denmark. E-mail: jegeblad@diku.dk.

†Computer Science Department, Stanford University, Stanford, CA 94305, USA. E-mail: guibas@cs.stanford.edu

‡Bioengineering department, Stanford University, Stanford, CA 94305, USA. E-mail: jonikas@stanford.edu

§Department of Biomedical Sciences, Wadsworth Center, New Scotland Av. Albany, NY 12208. USA, E-mail: alain@wadsworth.org

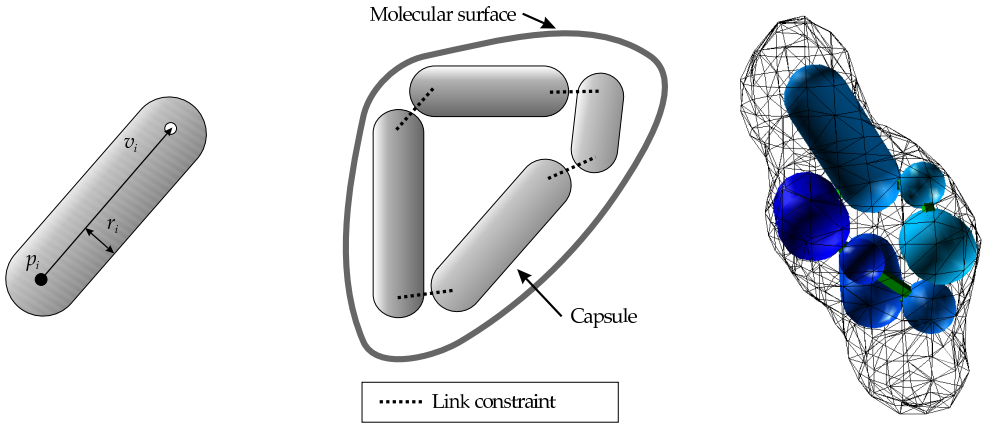


Figure 1: *(a) Example capsule. (b) Coarse model for tertiary structure used in this paper. (c) Example placement within triangle mesh (Link constraints are indicated with green beams).*

α-helix i, from the predicted secondary structure, be represented by a cylinder with spherical ends (*capsule*) whose radius, r_i, and length, l_i corresponds to the extents of the helix it represents. Since atoms cannot overlap, we require that *the helices do not overlap*.

Base-pairs between two helices are modeled as distance constraints (links), so that the endpoints of the capsules corresponding to the helices are required *to be within a distance of each other* that corresponds to estimated physical distance between the two helices.

Additional data may be available. In some cases scientist will know the molecular envelope of the RNA molecule which can be determined by SAXS and we may require that all capsules *must be located completely within the envelope*.

By observing experimental data, it may also be possible to identify parts of the helices that are exposed and should therefore lie close to the molecular surface and finally, angles between helices may be deduced and used to describe the relative orientation of two capsules. A sketch of the model is depicted on Figure 1 (a,b) along with a real placement of capsules on Figure 1 (c).

In this paper we focus on the problem of determining one or several placements of the capsules given the requirements. As we will show in Section 2, this problem is $\mathcal{NP}$-complete.

To solve this problem we define an objective function in which any violation of the requirements contributes positively to the objective value. A objective value of zero implies that we have found a feasible placement of the helices, that is, a placement where all requirements are met.

The method begins with an infeasible random placement and iteratively refines the placement until an objective value of zero is reached. In each iteration of the refinement we translate or rotate a single capsule to a position that reduces the value of the objective function.

Current techniques for RNA tertiary structure prediction may consider molecules with approximately 50 nucleotides. Since our method considers pure geometry it may open the door for rough placements of RNA structures with hundreds of nucleotides, that can later be refined by other techniques which accurately determine the positions of the individual nucleotides.

The remainder of this paper is organized as follows: In Section 2 we give the exact problem formulation. In Section 3 we describe related scientific work. In Section 4 we outline the solution method and in Section 5 we describe an efficient implementation of the local search moves.

2 Problem Formulation

We consider a simplified model for coarse grained RNA prediction consisting of a network of n interconnected helices represented as capsules with radius r_i, and length, l_i for $i = 1, \ldots, n$.

A capsule i in space may be represented as a coordinate $\mathbf{p}_i$, a direction vector $\mathbf{v}_i$ and a radius r_i, where capsule i is the set of points,

$$\{\mathbf{p} \in \mathbb{R}^3 \mid \min_{t \in [0,1]} ||\mathbf{p} - \mathbf{p}_i + t\mathbf{v}_i||\} \tag{1}$$

which are within a distance r_i from the line segment between $\mathbf{p}_i$ and $\mathbf{p}_i + \mathbf{v}_i$ (See figure 1 (a)).

Each capsule is defined in a local coordinate system where the endpoints $\mathbf{p}_i$ and $\mathbf{p}_i + \mathbf{v_i}$ of capsule i's are $(-\frac{l_i}{2}, 0, 0)^T$ and $(\frac{l_i}{2}, 0, 0)^T$. A placement of n capsules consists of a transformation for each capsule from its local coordinate system to a global coordinate system. Each transformation consists a rotation and translation. The translation is given by the vector $\mathbf{c}_i \in \mathbb{R}^3$. The rotation is represented by the matrix $M_{rot}(\theta_i, \phi_i)$, which first rotates the coordinate system $\theta_i \in \mathcal{A}$ radians around the z-axis, then $\phi_i \in \mathcal{A}$ radians around the x-axis. In this text it is assumed that the set of allowed angles $\mathcal{A}$ is a discrete set. Capsule i's endpoints in the global coordinate system for a placement $\mathcal{P} = (\mathbf{c}, \theta, \phi) \in \mathbb{R}^{3n} \times \mathcal{A}^n \times \mathcal{A}^n$ are $\mathbf{p}_i = M_{rot}(\theta_i, \phi_i)(-\frac{l_i}{2}, 0, 0)^T + \mathbf{c}_i$ and $\mathbf{p}_i + \mathbf{v}_i = M_{rot}(\theta_i, \phi_i)(\frac{l_i}{2}, 0, 0)^T + \mathbf{c}_i$.

Links which describe how individual helices are connected are modeled as maximal distance constraints. Links are numbered 1 to m and link i connects capsule $s(i)$ with $e(i)$. For a feasible placement we require that the distance between the endpoints, $\mathbf{p}_{s(i)} + \mathbf{v}_{s(i)}$, and $\mathbf{p}_{e(i)}$, must be less than $b(i)$, i.e:

$$||\mathbf{p}_{s(i)} + \mathbf{v}_{s(i)} - \mathbf{p}_{e(i)}||^2 \leq b(i)^2.$$

We also wish to ensure that all capsules lie within some molecular envelope, E, whose surface is closed and represented by a set of non-intersecting triangles E_s. A point $\mathbf{p}$ is enclosed by E_s if any ray from $\mathbf{p}$ intersects E_s an odd number of times. In other words, for a point $\mathbf{p} \in \mathbb{R}^3$, let $c(\mathbf{p}) \in \mathbb{Z}_+$ be the number of times the ray $r_-(\mathbf{p}) = s\mathbf{a} + \mathbf{p}$, for any vector $\mathbf{a}$ and $s < 0$ intersects any triangle $T \in E_s$, then the set of points enclosed by the envelope are:

$$E = \{\mathbf{p} \in \mathbb{R}^3 \mid c(\mathbf{p}) \equiv 1 \mod 2\},$$

which, since the surface is closed, is independent of the choice of $\mathbf{a}$.

To ensure that a capsule i is completely within E, we require that both its endpoints are in E and that the minimal distance from the line segment s_i to any triangle in T is larger than r_i. Note that such a capsule cannot cross the surface, since that would imply a minimal distance of 0.

We may also consider angles between capsules. For each link i we require that the angle in radians between the incident pair of capsules, $s(i)$ and $e(i)$ falls within some interval $[\alpha_i^-, \alpha_i^+] \subset]-\pi, \pi]$.

The primary objective in this paper is to find a placement $\mathcal{P}$, such that all the constraints listed above are met and we will refer to this problem as the *Interconnected Capsule Placement Decision Problem* (ICPDP).

Theorem 5. *The Interconnected Capsule Placement Decision Problem with rational coordinates and values is $\mathcal{NP}$-complete.*

Proof. Assume, for now, that we can determine if a capsule is located within a triangle envelope and if two capsules overlap in polynomial time (This will be shown in Section 5). Then we may determine

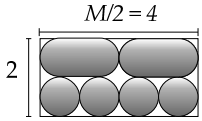


Figure 2: *Placement of the capsules that corresponds to the set of items* $\{1,1,1,1,2,2\}$ *in the proof of Theorem 5. Here capsules are either* $1-1$ *or* $2-1$ *units in length and have radius* $\frac{1}{2}$.

if a placement is feasible in polynomial time based on the coordinates of the capsules, and we can therefore use a placement as a polynomial size certificate.

To prove that the problem is $\mathcal{NP}$-complete we show that if we can solve any instance of ICPDP, we can solve any instance of the Set Partition Problem (SPP) which is $\mathcal{NP}$-complete (see e.g. [8]).

SPP is defined as follows; Given a set of items S, each with a positive integer value $a_i \in \mathbb{N}$ for $i \in S$, determine if we can divide S into two disjoint sets S' and S'' such that $S' \cup S'' = S$ and $\sum_{i \in S'} a_i = \sum_{i \in S''} a_i = \frac{1}{2}\sum_{i \in S} a_i$.

Given an instance I of SPP we may create an instance, I', of ICPDP as follows. For each item $i \in S$ from I create a capsule i in I' with length $l_i = a_i - 1$ and radius $\frac{1}{2}$. Set $M = \sum_{i \in S} a_i$ and create an envelope with the feasible domain $C = [0, \frac{M}{2}] \times [0,2] \times [0,1]$. Note that the only feasible z-coordinate for the endpoints of any capsule is $\frac{1}{2}$. We also fix the feasible set of rotation angles to $\{0\}$, so no rotation is allowed.

Given a solution to I, we may create a solution to I' in the following way. Assume (WLOG) that the items are enumerated such that $S' = \{i \mid 1 \leq i \leq |S'|\}$ and $S'' = \{i \mid |S'|+1 \leq i \leq |S|\}$. For each capsule i we set $\mathbf{v_i} = (a_i, 0, 0)$ and set $\mathbf{c}_i = (\frac{1}{2}a_i + \sum_{j<i} a_j, \frac{1}{2}, \frac{1}{2})$ for $i \in S'$ and $\mathbf{c}_i = (\frac{1}{2}a_i + \sum_{|S'|<j<i} a_j, \frac{3}{2}, \frac{1}{2})$ for $i \in S''$ (see figure 2). Note that no two capsules overlap with this assignment, since the length of capsule i is $a_i - 1$.

Conversely, assume we have a solution $\mathcal{S}$ to I', then divide the capsules into two sets $S' = \{i | c_{i,y} < 1\}$, consisting of the capsules with center y-coordinate less than 1, and $S'' = S \setminus S'$. Due to the dimensions of the capsules, the fact that rotations are not allowed, and the fact that no two capsules overlap in $\mathcal{S}$, we can deduce that the only feasible placement must be similar to the one shown in Figure 2, where the capsules are divided into two rows and therefore:

$$\sum_{i \in S'} (a_i - 1 + 2r_i) = \sum_{i \in S'} a_i \leq M$$

Similarly, $\sum_{i \in S''} a_i \leq M$. This shows that S' and S'' constitute a solution to I, which completes the proof that ICPDP is $\mathcal{NP}$-complete. □

3 Related work

This paper fall between the fields RNA-structure prediction and optimization of packing and layout problems and relevant work from both fields is briefly discussed in the following.

3.1 RNA Structure Prediction

In both the surveys on RNA structure prediction by Shapiro et al. [20] and Capriotti and Marti-Renom [2] a section is devoted to tertiary structure prediction of RNA. Presently, most methods for tertiary structure prediction rely on some form of human assistance. The Erna-3D program by Muller [19]

builds helices based on the secondary structure. The helices are combined to form a complete tertiary structure. The program is based on human manipulation of the generated structured on different levels of detail with the highest level consisting of complete helices. Massire and Westhof [17] present a tool that builds the tertiary structure based on a library of RNA motifs. Once constructed, the structures can be manipulated interactively. MC-Sym by Major [15] also builds 3D structures from known 3D structures, and allows interactive specification of structural constraints. The 3D structures may be further refined using molecular dynamics simulation which minimize their energy. The RNA2D3D program by Martinez et al. [16] generates helices from the secondary structure and spaces atomic models of nucleotides evenly on a backbone which is used to create the three-dimensional structure by winding. This first order representation may then be further refined interactively and by molecular dynamics.

Das and Baker [3] presents a procedure which is inspired by the Rosetta low-resolution protein structure prediction method. The method assembles RNA fragments controlled by a Monte Carlo method in order to minimize a knowledge based energy function and is able to accurately predict structures consisting of approximately 30 nucleotides. Finally, Ding et al. [4] uses discrete molecular dynamics (DMD) to fold structures consisting of up-to 100 nucleotides although best results are reached for less that 50 nucleotides.

Some geomtric aspects of RNA structures were analyzed by Hyeon et al. [12]. The RNA molecules studied were found to be more aspherical and prolate than proteins. Furthermore the radius of gyration, which determines the compactness of the molecules, was found to be consistently $R_G = 5.5\text{Å}N^{\frac{1}{3}}$ for N nucleotides, which is less than the compactness for proteins.

3.2 Packing and Layout

Since the methods that we use in this paper have been previously applied to packing and layout problems we briefly consider this field. Packing problems of non-rectangular shapes in three dimensions have been considered by several authors. Stoyan et al. [22] considers optimal packing of convex polyhedra within a rectangular container, while a similar problem involving spheres was considered by Stoyan and et al. [21]. Imamichi and Hiroshi [13] also considers packing of spheres and model rigid shapes as collections of spheres. In addition to packing problems, the method is applied to protein-protein docking problems. The methodology closely resembles the strategies described in this paper, since a placement with overlap is continuously refined using a gradient search method until a non-overlapping placement is reached.

Determining a placement of objects given a set of constraints has previously been considered for component layout optimization. Here a given set of items inter-connected by wires must be placed within a container such that some objective is optimized. The objective can be wire-length or the overall center of gravity, and in some cases additional proximity constraints must be met. A survey of layout problems was given by Cagan et al. [1] and recent work is presented by Yin et al. [24].

The methodology used in this paper, as presented in Section 4, originates from work by Faroe et al. [7] who consider a relaxed placement method for the bin-packing problem, where the minimal number of rectangular bins required to contain a set of rectangular items must be found. The method revolves around a procedure which starts with infeasible placements of overlapping rectangular items, that are continuously refined to reach non-overlapping placements. The refinement procedure consists of repeated translations of individual items to less-overlapping positions. The refinement process was also used by Egeblad et al. [A] for two- and three-dimensional strip-packing problems, where a minimal length container capable of encompassing a set of polygons must be found. The method was generalized to three-dimensional problems involving general polyhedra and to higher dimensional

problems by Egeblad and Pisinger [C]. The refinement procedure Faroe et al. [6] uses a similar method for very large circuit (VLSI) layout problems, which is a two-dimensional component layout problem.

4 Solution Method

The solution method follows the work by Faroe et al. [6, 7] and Egeblad et al. [A, B] which was briefly touched upon in Section 3.2. The intuition behind our method is as follows. We begin with a random placement of the capsules which is unlike to meet all of our requirements. As our solution method progresses capsules are allowed to be freely moved around in the coordinate system even that make them overlap with other capsules, extend beyond the envelope, and violate link and angle constraints. Violations of constraints are described by a continuous objective function, such that a "larger" violation of constraints has a higher objective value and a placement of the capsules is feasible with respect to the requirements if and only if the objective value is zero. Our method iteratively approaches a feasible placement where all requirements are met by reducing the objective value in each step. In each iteration exactly one capsule, which contributes positively to, the objective function is selected and all possible axis aligned translations as well as rotations of the capsule are considered. The translation or rotation that reduces the objective value the most is selected.

The objective function we consider is defined as follows:

$$\mathcal{F}(\mathcal{P}) = \sum_{i=1}^{n}\sum_{j=i}^{n} f_C(\mathcal{P}, \overline{\mathbf{v}}_{ij}, i, j) + \sum_{i=1}^{n} f_E(\mathcal{P}, \tilde{\mathbf{v}}_i, i) + \sum_{i=1}^{m} f_L(\mathcal{P}, i) + \sum_{i=1}^{m} f_A(\mathcal{P}, i),$$

where $\mathcal{P}$ is the current placement of capsules, and each of the functions f_C, f_E, f_L and f_A are explained in the following and illustrated on Figure 3. We emphasize that all these values are with respect to the current placement $\mathcal{P}$.

$f_C(\mathcal{P}, \overline{\mathbf{v}}_{ij}, i, j) \geq 0$ is the amount capsule i must be translated along the vector $\overline{\mathbf{v}}_{ij}$ in placement $\mathcal{P}$ in order for it not to overlap with j as illustrated on Figure 3 (a). This value is related to the concept *intersection depth*, and will be described in more detail in Section 5.1. Note that $f_C(\mathcal{P}, \overline{\mathbf{v}}_{ij}, i, j) = 0$ if and only if i and j are not overlapping in $\mathcal{P}$. T

$f_E(\mathcal{P}, \tilde{\mathbf{v}}_i, i) \geq 0$ indicates how far capsule i must be moved in direction $\tilde{\mathbf{v}}_i$ in placement $\mathcal{P}$ in order to be completely contained within the envelope E. This is illustrated on Figure 3 (b). $f_E(\mathcal{P}, \tilde{\mathbf{v}}_i, i) = 0$ if and only if i is completely contained within E.

For link k which connects capsules $i = s(k)$ and $j = e(k)$ the value $f_L(\mathcal{P}, k) = \max(||\mathbf{p}_{s(k)} + \mathbf{v}_{s(k)} - \mathbf{p}_{e(k)}||^2 - b(i)^2, 0) \geq 0$, is a measure of how far capsules $i = s(k)$ and $j = e(i)$ should be moved relative to their placement in $\mathcal{P}$ in order for the distance between the endpoints of the capsules i and j to be feasible. This is illustrated on Figure 3 (c) Note that $f_L(\mathcal{P}, k) = 0$ if and only if the distance between the endpoints of the two capsules linked by link i are within a feasible range.

For link k the value $f_A(\mathcal{P}, k)$ indicates how far the angle between capsules $s(k)$ and $e(k)$, is from the target angle interval $[\alpha_i^-, \alpha_i^+]$ and we set:

$$f_A(i) = \begin{cases} \alpha_i^- - \angle(i) & \text{for } \angle(i) < \alpha_i^- \\ 0 & \text{for } \alpha_i^- \leq \angle(i) \leq \alpha_i^+ \\ \angle(i) - \alpha_i^+ & \text{for } \angle(i) > \alpha_i^+ \end{cases}$$

where $\angle(\mathcal{P}, i)$ is the angle in radians between capsules $s(i)$ and $e(i)$ and the difference is calculated modulo 2π, i.e. it is always positive. This is illustrated on Figure 3 (c).

We will give more details on how to evaluate the different terms of the objective function and explain the choice of the vectors $\overline{\mathbf{v}}_{ij}$ and $\tilde{\mathbf{v}}_i$ in Section 5. As can be seen from the previous description $\mathcal{F}(\mathcal{P}) = 0$ if and only if $\mathcal{P}$ is feasible with respect to our set of requirements.

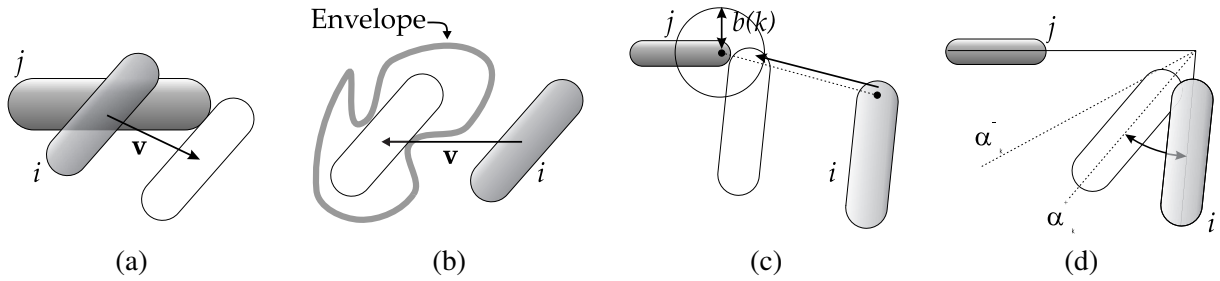


Figure 3: *Illustration of the different terms of the objective function. In all cases the capsule i is at an infeasible position and the white hollow capsule represents a feasible position. (a) Capsules i and j overlap and i must be translated $f_C(\mathcal{P},\mathbf{v},i,j)$ along* $\mathbf{v}$ *(indicated by the arrow) to remove the overlap. (b) i is placed outside E and must be translated $f_E(\mathcal{P},\mathbf{v},i)$ along* $\mathbf{v}$ *(indicated by the arrow), (c) A link k connects capsules i and j, which are placed too far from each other. $f_L(\mathcal{P},k)$ (indicated by the arrow) is the quadratic distance that i must be translated for its endpoint to be within a distance $b(k)$ (represented by the circle) of j's endpoint. (d) Link k connects capsules i and j and i should be rotated $f_A(\mathcal{P},k)$ (represented by the arrow) for the angle to be within the required $[\alpha_k^-,\alpha_k^+]$.*

4.1 Local search overview

Our procedure starts with a random, and likely, infeasible placement $\mathcal{P}_0$. We now seek to minimize $\mathcal{F}$ using a simple local search scheme which from a placement $\mathcal{P}_k$ searches for a new placement $\mathcal{P}_{k+1}$ such that $\mathcal{F}(\mathcal{P}_{k+1}) < \mathcal{F}(\mathcal{P}_k)$. The possible changes we look for with respect to $\mathcal{P}_k$ revolves around a single capsule i and are as follows:

1. Translate i in directions parallel to the three coordinate system vectors.
2. Translate i in direction $\nabla f(x,y,z)$, where $\mathcal{P}_k(x,y,z)$ is the placement $\mathcal{P}_k$ with (x,y,z) added to c_i and $f(x,y,z) = \sum_{j=1}^{m} f_L(\mathcal{P}_k(x,y,z),j)$. The purpose of this change, is that a translation in this direction will reduce the link term of the objective function.
3. Rotate i. The set of feasible angles is limited to the discrete set $\mathcal{A}$.

We refer to this set of possible changes as the *local search neighborhood*. All three possible changes are evaluated, and all possibilities among each type of translation are investigated. The translation or rotation which reduces $\mathcal{F}(\mathcal{P}_k)$ the most is selected and the new placement is $\mathcal{P}_{k+1}$. Evaluation of the objective function for each possible translation is a computationally expensive process, and we will explain how this can be done efficiently in polynomial time in Section 5.

We refer to the set of placements which may arise from one of the changes listed above applied to a placement $\mathcal{P}$ as the local search neighborhood, $\mathcal{N}(\mathcal{P})$. If $\mathcal{F}(\mathcal{P}') \geq \mathcal{F}(\mathcal{P})$ for any placement $\mathcal{P}' \in \mathcal{N}(\mathcal{P})$, we say that $\mathcal{P}$ is a local minimum with respect to $\mathcal{F}$ and the local search neighborhood.

The local search process proceeds until either a placement $\mathcal{P}_k$ with $\mathcal{F}(\mathcal{P}_k) = 0$ or a local minimum placement is found. If $\mathcal{F}(\mathcal{P}_k) = 0$ we have solved the specific instance of the ICPDP and return $\mathcal{P}_k$ as solution. To continue the process from a local minimum we use the metaheuristic Guided Local Search which will be described in Section 4.2.

Note that when the local search move selected is a translation then we calculate the f_C and f_E terms with respect to the direction of translation as will be described in Section 5. We also set the values of the vectors $\bar{\mathbf{v}}_{ij}$ and $\bar{\mathbf{v}}_{ji}$ for $j = 1,\ldots,n$ and $\tilde{\mathbf{v}}_i$ to the chosen direction. In other words, the

value which determines the translation distance required for a capsule i, in order for i not to overlap, is always with respect to the last translation that changed the overlap of i. Therefore one could argue that during the optimization of $\mathcal{F}(\mathcal{P})$ we also attempt to find the right set of vectors of $\bar{\mathbf{v}}_{ij}$ and $\tilde{\mathbf{v}}_i$. However, for $\mathcal{F}(\mathcal{P}) = 0$ the choice of vectors has no effect on the objective value. Therefore our search is still limited to finding a set of translations and rotations of the capsules such that $\mathcal{F}(\mathcal{P}) = 0$.

4.2 Guided Local Search

One of the main challenges with local search based methods is to ensure that they can continue the search for a global minimum once they encounter a local minimum. A common way to solve this problem is to control the local search with a form of metaheuristic. Metaheuristics consists of general principle used to attack combinatorial optimization problems and some of most succesful and well-known metaheuristics are Simulated Annealing (Monte Carlo Methods) Kirkpatrick et al. [14], Genetic Algorithms Mitchell [18] and Tabu search Glover [9, 10]. Although, many local search based metaheuristics are percieved as generic tools, some metaheuristics are often more suitable than others for a specific local search procedure.

The metaheuristic Guided Local Search (GLS) introduced by Voudouris and Tsang [23] has previously proved successful for packing and layout optimization problems as described by Faroe et al. [6, 7] and Egeblad et al. [A, B] and was therefore found suitable for solving ICPDP as well. The primary element of GLS is to minimize an augmented objective function in which undesirable features of placements are *penalized* by adding a set of additional *penalty*-terms to the objective function one wishes to optimize. Whenever the local search procedure reaches a local minimum placement, the augmented objective function is altered by modifying the penalty-terms, such that the current placement ceases to be a local minimum relative to the modified augmented objective function. An important part of this paradigm is that penalty terms must be carefully added such that any global minimum of the augmented objective function is also a global minimum of the original objective function.

For ICPDP we use the following augmented objective function:

$$\text{minimize } \mathcal{H}(\mathcal{P}) \quad = \quad \mathcal{F}(\mathcal{P}) + \mathcal{Z}(\mathcal{P}),$$

where $\mathcal{Z}(\mathcal{P})$ is value of the penalties in $\mathcal{P}$ and given by

$$\mathcal{Z}(\mathcal{P}) = \lambda_C \sum_{i=1}^{n} \sum_{j=i+1}^{n} I_C(i,j,\mathcal{P})\rho_{i,j} + \lambda_E \sum_{i=1}^{n} I_E(i)\psi_i + \lambda_L \sum_{i=1}^{m} I_L(i)\sigma_i + \lambda_A \sum_{i=1}^{n} I_A(i)\tau_i.$$

Here $I_C(i,j,\mathcal{P}) = 1$ if and only if capsules i and j overlap which is true if and only if $f_C(\mathcal{P},\bar{\mathbf{v}}_{ij},i,j) > 0$, $I_E(i) = 1$ if and only if capsule i is not contained within the envelope which is the case if and only if $f_E(\mathcal{P},\tilde{\mathbf{v}}_i,i) > 0$, $I_L(i) = 1$ if and only if link-distance i is violated and therefore $f_L(\mathcal{P},i) > 0$, and $I_A(i) = 1$ if and only if the angle of link i is not within its required interval which can be true if and only if $f_A(\mathcal{P},i) > 0$. The values $\rho_{i,j}$, ψ_i, σ_i and τ_i are penalty counts for respectively inter-capsule overlap, envelope overlap, link-distance violation, and angle violation and these are explained shortly. The values $\lambda_C \geq 0$, $\lambda_E \geq 0$, $\lambda_L \geq 0$ and $\lambda_A \geq 0$ are parameters that determines the weight of each of the penalties in $\mathcal{H}$ and are used to fine-tune the behavior of the heuristic. Note that with this definition $\mathcal{F}(\mathcal{P}) = 0$ if and only if $\mathcal{H}(\mathcal{P}) = 0$.

Initially all $\rho_{i,j}$, ψ_i, σ_i and τ_i are set to 0 and $\mathcal{H} = \mathcal{F}$. However, whenever a local minimum placement $\mathcal{P}$ with $\mathcal{H}(\mathcal{P}) > 0$ is encountered, one of the penalty terms are modified. Note that such

a placement must contain either overlap, envelope overlap, link-distance violation, or incorrect link-angles. The heuristic randomly selects one among these four contributions to change. If the overlap penalty term is chosen we calculate the utility of this feature as:

$$\mu_{i,j} = \frac{f_C(\mathcal{P}, \overline{\mathbf{v}}_{ij}, i, j)}{1 + \rho_{i,j}}, \qquad i, j = 1, \ldots, n$$

and increase the value $\rho_{i,j}$ by 1 for the pair of capsules i and j with maximal $\mu_{i,j}$. If the envelope overlap term is chosen we calculate:

$$\xi_i = \frac{f_E(\mathcal{P}, \tilde{\mathbf{v}}_i, i)}{1 + \psi_i}, \qquad i = 1, \ldots, n$$

and increase ψ_i for the capsule i with largest ξ_i. If the link-distance term is chosen we calculate:

$$\nu_i = \frac{f_L(\mathcal{P}, i)}{1 + \sigma_i}, \qquad i = 1, \ldots, m$$

and increase the value σ_i for the link i with the highest ν_i. Finally, if the angle term is chosen we calculate:

$$\gamma_i = \frac{f_A(\mathcal{P}, i)}{1 + \tau_i}, \qquad i = 1, \ldots, m$$

and increase the value τ_i for the link i with the highest γ_i.

After this change of objective function the local search heuristic continues with the modified objective. The effect of the modification is that the undesirable features, e.g. large overlap of two specific capsules, are "emphasized" in subsequent optimization and the local search heuristic will move towards placements without this particular overlap.

4.3 Fast Local Search

A very important aspect of the outlined local search procedure, is the selection of the capsule in each step. Searching for new placements with respect to *every* capsule in each iteration of the local search is computationally expensive. Rather, we use a concept referred to as *Fast Local Search* (FLS).

The details of FLS are as follows: We maintain a list $\mathcal{L}$ of capsules and in each step the local search procedure searches only for improving changes to the first capsule in the list. Initially the list $\mathcal{L}$ contains all the capsules. Whenever the local search procedure has attempted to change the placement of a capsule, it is inactivated and removed from $\mathcal{L}$ and the procedure continues with the next capsule in $\mathcal{L}$. If an improving change of the placement for a capsule is found, all capsules connected to it via links and capsules overlapping with it before or after the move are activated and put in $\mathcal{L}$.

If $\mathcal{L}$ is empty, we assume the current placement is a local minimum placement, and we proceed to change the penalties as described in the previous section. Afterwards, the capsule(s) associated with the penalized feature are inserted in $\mathcal{L}$ and will be considered in subsequent steps by the local search heuristic.

5 Neighborhood search

In this Section we will show how to evaluate the local search neighborhood described in Section 4.1 efficiently. The computationally most expensive element of the neighborhood is to determine the

translation of a capsule i in a direction $\mathbf{a}$ that minimizes $\mathcal{H}$. In this section, we will consider a specific capsule i and let $\mathcal{P}(t)$ denote the placement that arises from $\mathcal{P}$ when i is translated $t\mathbf{a}$ units relative to its current position.

A piecewise quadratic function $f(t)$ consists of k second order polynomials each constrained to a specific interval (pieces):

$$f(t) = \begin{cases} a_1t^2 + b_1t + c_1 & \text{for } t \in [t_1, t_2] \\ \vdots & \\ a_kt^2 + b_kt + c_k & \text{for } t \in [t_{2k-1}, t_{2k}] \end{cases},$$

where each of the coefficient a_i, b_i, $c_i \in \mathbb{R}^3$ and $[t_{2i-1}, t_{2i}] \cap [t_{2j-1}, t_{2j}] = \emptyset$ for $i, j = 1, \ldots, k$.

Rather than probing $\mathcal{H}(\mathcal{P}(t))$ for a discrete set of values t and selecting the best translation from this set, we will present an efficient polynomial time algorithm that returns the minimal value of $\mathcal{H}(\mathcal{P}(t))$. The algorithm determines a piecewise quadratic function which describes $\mathcal{H}(\mathcal{P}(t))$ for any value of $t \in \mathbb{R}$, given $\mathbf{a}$ and capsule i. It has asymptotic running time $O((n+m+|E_s|)\log(n+m+|E_s|))$, where n is the number of capsules in the instance, m is the number of links, and $|E_s|$ is the number of surface triangles from the envelope. Specifically the algorithm can be used to find t for $\min_t \mathcal{H}(\mathcal{P}(t))$ in the same asymptotic time, by analyzing each piece for its minimum.

$\mathcal{H}$ consists of three components that depend on t; capsule overlap (f_C) envelope overlap (f_E), and link violations (f_L). In the following we discuss how we may determine piecewise quadratic functions $f_C(\mathcal{P}(t), \mathbf{a}, i, j)$, $f_E(\mathcal{P}(t), \mathbf{a}, i)$, and $f_L(\mathcal{P}(t), \mathbf{a}, k)$ over t for capsules i and j and link $k \in \{1, \ldots, m\}$.

5.1 Capsule Intersection

The first terms of $\mathcal{F}$ are the f_C terms. $f_C(\mathcal{P}, \mathbf{a}, i, j)$ describes the minimum amount capsule i must be translated in a specific direction in order for i not to overlap with j as illustrated on Figure 3 (a).

To determine if two capsules overlap for different translations t along $\mathbf{a}$ we will evaluate the minimum quadratic distance between two capsules. One with the line segment endpoints $\mathbf{p}_1 + t\mathbf{a}$ and $\mathbf{p}_1 + \mathbf{v}_1 + t\mathbf{a}$, and one with the line segment endpoints $\mathbf{p}_2$ and $\mathbf{p}_2 + \mathbf{v}_2$ as a function in t. The capsules overlap, if the distance between these segments is less than the sum of the capsules' radii.

For a given translation t and direction $\mathbf{a}$ let $f(t, \mathbf{a}, s, u) = ||\mathbf{p}_1 + s\mathbf{v}_1 + t\mathbf{a} - \mathbf{p}_2 - u\mathbf{v}_2||^2$ be the distance between specific points on the infinite lines that are coincident with the two line segments, let

$$d_{LL}(t, \mathbf{a}, \mathbf{p}_1, \mathbf{v}_1, \mathbf{p}_2, \mathbf{v}_2) \quad = \quad \min_{s,u \in \mathbb{R}} f(t, \mathbf{a}, s, u),$$

be the minimal quadratic distance between the two infinite lines that are coincident with the line segments, and let

$$d_{S,S}(t, \mathbf{a}, \mathbf{p}_1, \mathbf{v}_1, \mathbf{p}_2, \mathbf{v}_2) = \min_{s,u \in [0,1]} f(t, \mathbf{a}, s, u)$$

be the minimal quadratic distance between the two line segments.

For fixed values of t and $\mathbf{a}$ f is a two dimensional quadratic function in s and u, and the minimum distance between the two infinite lines occurs for values of $s'(t)$ and $u'(t)$ where:

$$\left(\frac{\partial f}{\partial s}(t, s'(t), u'(t)), \frac{\partial f}{\partial u}(t, s'(t), u'(t)) \right) = (0, 0). \tag{2}$$

It can be deduced that:

$$\left(\frac{\partial f}{\partial s}(t,s,u),\frac{\partial f}{\partial u}(t,s,u)\right) = 2\mathbf{M}\begin{pmatrix} s \\ u \end{pmatrix} + 2V + 2t\begin{pmatrix} \mathbf{v}_1\cdot\mathbf{a} \\ -\mathbf{v}_2\cdot\mathbf{a} \end{pmatrix},$$

with

$$\mathbf{M} = \begin{pmatrix} ||\mathbf{v}_1||^2 & -\mathbf{v}_1\cdot\mathbf{v}_2 \\ -\mathbf{v}_1\cdot\mathbf{v}_2 & ||\mathbf{v}_2||^2 \end{pmatrix} \quad \text{and} \quad \mathbf{V} = \begin{pmatrix} \mathbf{v}_1\cdot(\mathbf{p}_1-\mathbf{p}_2) \\ \mathbf{p}_2\cdot(\mathbf{p}_2-\mathbf{p}_1) \end{pmatrix}.$$

Which allows us to define $s'(t)$ and $u'(t)$ as

$$\begin{pmatrix} s'(t) \\ u'(t) \end{pmatrix} = -M^{-1}V - tM^{-1}\begin{pmatrix} \mathbf{v}_1\cdot a \\ -\mathbf{v}_2\cdot\mathbf{a} \end{pmatrix},$$

which shows that the values $s'(t)$ and $u'(t)$ are linear in t.

If we let $S = \{t \mid s'(t) \in [0,1]\}$ and $U = \{t \mid u'(t) \in [0,1]\}$ then both $s'(t) \in [0,1]$ and $u'(t) \in [0,1]$ for $t \in S \cap U$. This shows, that for $t \in S \cap U$ the two nearest points of the infinite lines are on the line segments and for $t \in S \cap U$ we can evaluate the distance between the two line segments as the distance $d_{LL}(t,\mathbf{a},\mathbf{p}_1,\mathbf{v}_1,\mathbf{p}_2,\mathbf{v}_2)$ between two infinite lines going through the endpoints of the capsules. The distance between two infinite lines can be determined as:

$$d_{LL}(t,\mathbf{a},\mathbf{p}_1,\mathbf{v}_1,\mathbf{p}_2,\mathbf{v}_2) = \frac{(t(\mathbf{a}\cdot(\mathbf{v}_1\times\mathbf{v}_2)) + (\mathbf{p}_1-\mathbf{p}_2)\cdot(\mathbf{v}_1\times\mathbf{v}_2))^2}{||\mathbf{v}_1\times\mathbf{v}_2||^2}.$$

For $t \notin S \cap U$ at least one of the two nearest points of the line segments, is an endpoints of a line segment and we can evaluate the minimum distance between the two line segments as the minimum distance between all of the four line segment endpoints and the opposite line segment:

$$\begin{aligned} d_{PPSS}(t,\mathbf{a},\mathbf{p}_1,\mathbf{v}_1,\mathbf{p}_2,\mathbf{v}_2) = \quad \min\big(\quad & d_{PS}(t,\mathbf{a},\mathbf{p}_1,\ \mathbf{p}_2,\mathbf{v}_2), d_{PS}(t,\mathbf{a},\mathbf{p}_1+\mathbf{v}_1,\mathbf{p}_2,\mathbf{v}_2) \\ & d_{PS}(t,-\mathbf{a},\mathbf{p}_2,\mathbf{p}_1,\mathbf{v}_1),\ d_{PS}(t,-\mathbf{a},\mathbf{p}_2+\mathbf{v}_2,\mathbf{p}_1,\mathbf{v}_1)\ \big), \end{aligned}$$

where

$$\begin{aligned} d_{PS}(t,\mathbf{a},\mathbf{p},\mathbf{p}_2,\mathbf{v}_2) &= ||\mathbf{p}+t\mathbf{a}-\mathbf{p}_2-r(t,\mathbf{p},\mathbf{p}_2,\mathbf{v}_2)\mathbf{v}_2||^2, \\ r(t,\mathbf{a},\mathbf{p},\mathbf{p}_2,\mathbf{v}_2) &= \max\big(0,\min(1,r'(t,\mathbf{p},\mathbf{p}_2,\mathbf{v}_2))\big), \\ r'(t,\mathbf{a},\mathbf{p},\mathbf{p}_2,\mathbf{v}_2) &= \frac{t\mathbf{a}\cdot\mathbf{p}_2+\mathbf{p}_1\cdot\mathbf{v}_2}{||\mathbf{v}_2||^2}. \end{aligned}$$

Here $r(t,\mathbf{a},\mathbf{p},\mathbf{p}_2,\mathbf{v}_2)$ is the value of s for the point on the line segment $\mathbf{p}_2+s\mathbf{v}_2$, $s \in [0,1]$ that is closest to $\mathbf{p}$. By analyzing the interval for t where $r'(t,\mathbf{p}_1,\mathbf{p}_2,\mathbf{v}_2) \in [0,1]$, we can describe $d_{PS}(t,\mathbf{a},\mathbf{p},\mathbf{p}_2,\mathbf{v}_2)$ as a piecewise quadratic equal to the quadratic distance between points $\mathbf{p}+t\mathbf{a}$ and $\mathbf{p}_2$ for $r'(t,\mathbf{p},\mathbf{p}_2,\mathbf{v}_2) < 0$, the point-line distance between $\mathbf{p}$ and the line-segment $\mathbf{p}_2,\mathbf{p}_2+\mathbf{v}_2$ for $0 \le r'(t,\mathbf{p}_i,l) \le 1$, and the distance between points $\mathbf{p}_1+t\mathbf{a}$ and $\mathbf{p}_2+\mathbf{v}_2$ for $r'(t,\mathbf{p}_i,l) > 0$. In total the distance between the two line segments can be calculated as:

$$d_{SS}(t,\mathbf{a},\mathbf{p}_1,\mathbf{v}_1,\mathbf{p}_2,\mathbf{v}_2) = \begin{cases} d_{LL}(t,\mathbf{a},\mathbf{p}_1,\mathbf{v}_1,\mathbf{p}_2,\mathbf{v}_2) & \text{for } t \in S\cap U. \\ d_{PPSS}(t,\mathbf{a},\mathbf{p}_1,\mathbf{v}_1,\mathbf{p}_2,\mathbf{v}_2) & \text{otherwise .} \end{cases}$$

Since both d_{LL} and d_{PS} are piecewise quadratic in t, d_{SS} may be represented as a single piecewise quadratic function. Note that extra attention must be given for orthogonal lines and that evaluating $d_{PPSS}(t,\mathbf{a},\mathbf{p}_1,\mathbf{v}_1,\mathbf{p}_2,\mathbf{v}_2)$ can be simplified with further boundary analysis.

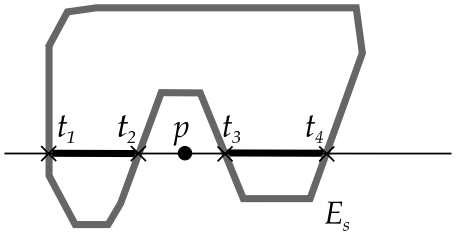


Figure 4: *Illustration of the feasible translations of a point* $\mathbf{p}$*, such that it is contained within the envelope E (see text). For a point* $\mathbf{p}+t\mathbf{a}$ *to be feasible, the number of intersections between the envelope surface* E_s *and the ray* $\mathbf{p}+t\mathbf{a}-u\mathbf{a}$*, for* $u \in [0,\infty]$ *must be odd. Here translations with t in the intervals* $[t_1,t_2]$ *or* $[t_3,t_4]$ *are feasible.*

Having the minimum distance for any translation along a vector $\mathbf{a}$ of two line segments allows us to evaluate the interval $O = [t_1,t_2]$, where the capsules i and j overlap. The endpoints of O are the up-to two values of t for which $d_{SS}(t,\mathbf{a},\mathbf{p}_1,\mathbf{v}_1,\mathbf{p}_2,\mathbf{v}_2) = (r_i+r_j)^2$.

We now set the value $f_C(\mathcal{P}(t),\mathbf{a},i,j) = \min(-t_1,t_2)$ if $t \in O$ and 0 otherwise since we must translate i either $-t_1$ or t_2 units along $\mathbf{a}$ in order for the two capsules not to overlap. We can also use this to calculate the value of $f_C(\mathcal{P}(t),\mathbf{a},i,j)$ by creating the piecewise linear-function:

$$f_C(\mathcal{P}(t),\mathbf{a},i,j) = \begin{cases} t-t_1 & \text{for } t \in [t_1, \frac{t_2-t_1}{2}] \\ t_2-t & \text{for } t \in [\frac{t_2-t_1}{2}, t_2] \\ 0 & \text{otherwise} \end{cases} .$$

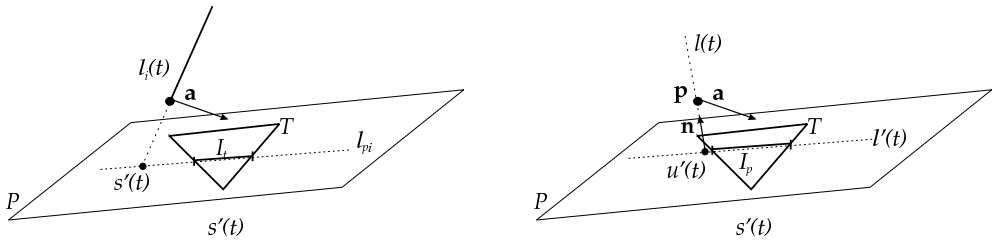
Thus $f_C(\mathcal{P}(t),\mathbf{a},i,j)$ for all $t \in \mathbb{R}^3$ can be determined in $O(1)$ asymptotic time and the resulting piecewise linear function consists of no more than 4 pieces.

5.2 Surface Intersection

The second term f_E of $\mathcal{F}(t)$ concerns placement of capsules outside the envelope as illustrated on Figure 3 (b). First we note that a capsule can only be inside the envelope if both its endpoint are inside the envelope, it does not cross the envelope, and the minimal distance from the line segment to any triangle on the envelope is larger than the radius of the capsule. Our strategy is to determine the set of intervals of t where all these conditions are met. The intersection of those intervals constitute the feasible translations of capsule i along vector $\mathbf{a}$.

To determine if an endpoint $\mathbf{p}$ is within the envelope we may simply cast a ray from $\mathbf{p}$ in direction of $-\mathbf{a}$. If the number of intersections between the ray, and the envelope is odd, $\mathbf{p}$ is inside the envelope. This concept can be used to return the set of intervals of t where $\mathbf{p}$ is within the envelope. This is done by determining all intersections between triangles of the envelope and the line $\mathbf{p}+t\mathbf{a}$. Denote the distinct values of t for the intersections as $t_1,\ldots,t_k$ and assume, WLOG, that they are sorted such that $t_1 < t_2 < \ldots < t_k$. Since translations of $\mathbf{p}$ with an odd number of intersections are feasible translations, we know that values of t within the intervals $[t_1,t_2]$, $[t_3,t_4]$, $\ldots$, $[t_{k-1},t_k]$ are feasible (see Figure 4).

Since multiple intersections can occur for equal values of t when triangle edges are coincident we ensure that the distance between to subsequent intersections must be larger than a small value ε. Calculating these intervals may be done in $O(|E_s|\log|E_s|)$ time, since it takes $O(|E_s|)$ time to calculate all the line-triangle intersections, $O(|E_s|\log|E_s|)$ time to sort them and $O(|E_s|)$ time to traverse them and generate the intervals. The set of intervals representing feasible translations of both endpoints



(a) (b)

Figure 5: *Illustration of the formulae required to calculate the distance from a line segment to a triangle as described in Section 5.2.*

can be calculated in $O(|E_s|\log|E_s|)$ time by determining the intersection of the intervals from each endpoint.

We now consider the problem of determining the distance between the line segment of capsule i, $l_i(t) : \mathbf{p}_i + s\mathbf{v}_i + t\mathbf{a}$, $s \in [0,1]$ and a triangle T. If the line segment intersects T, the distance is 0 otherwise the minimal distance is the minimum of the distances between $l_i(t)$ and any of T's edges or the minimal distance from one of $l_i(t)$'s endpoints to T.

To determine if $l_i(t)$ intersects T we first need to determine the point of intersection between the infinite line coincident with $l_i(t)$ and the plane of T as a a parameter of t (see Figure 5). Let the plane of T be defined as the set of points $P : \{\mathbf{p} \in \mathbb{R}^3 | \mathbf{n} \cdot \mathbf{p} + q = 0\}$ where $\mathbf{n}$ is the normal of the plane and $q \in \mathbb{R}$. Let $s'(t)$ be the value of s for the point of intersection between the infinite line going through $l_i(t)$ and P, then we require that:

$$\mathbf{n} \cdot (\mathbf{p}_i + s'(t)\mathbf{v}_i + t\mathbf{a}) + q \quad = \quad 0,$$

which implies that:

$$s'(t) \quad = \quad \frac{\mathbf{n} \cdot \mathbf{p}_i + q}{\mathbf{n} \cdot v_i} + t\frac{\mathbf{n} \cdot a}{\mathbf{n} \cdot v_i}.$$

If $\mathbf{n} \cdot \mathbf{v}_i = 0$, $l_i(t)$ and the P are parallel and no intersection occurs. Otherwise, the points of intersection are represented by the line:

$$l_{pi} : t\frac{\mathbf{n} \cdot \mathbf{a}}{\mathbf{n} \cdot \mathbf{v}_i}\mathbf{v}_i + \mathbf{p}_i + \frac{\mathbf{n} \cdot \mathbf{p}_i + q}{n \cdot v_i}v_i,$$

which lies in the plane P. Let $I_t = \{t \mid l_{pi}(t) \in T\}$ be the interval of intersection between l_{pi} and T, then I_t is the interval of t for which the infinite line going through l_i intersects T. Let $I_s = \{t \mid s'(t) \in [0,1]\}$, then the interval $I_T \cap I_s$ is the set of values for t where $l_i(t)$ intersects the triangle T (see Figure 5 (a)).

The problem of determining the minimal distance between the line segment and T's edges as a parameter in t is similar to the problem of determining the distance between two line segments which was covered in Section 5.1.

To determine the minimum distance from the line segment endpoints to the interior of T, consider a point $\mathbf{p} + t\mathbf{a}$ (which can be either endpoint of $l_i(t)$), then the distance from $\mathbf{p} + t\mathbf{a}$ to P is $\mathbf{n} \cdot (\mathbf{p} + t\mathbf{a})/||\mathbf{n}|| + q$. This distance is valid as a distance to T when the point closest to $\mathbf{p} + t\mathbf{a}$ on P is within T. When this is not the case, the closest point of T is on one of T's edges and this situation

is handled by the line segment edge distance evaluation mentioned above. To determine for which values of t the closest point on P is within T we calculate the line (in t) representing the closest point on P to $\mathbf{p}+t\mathbf{a}$ for any value of t as follows. First, we define a line $l(t):\mathbf{p}+u\mathbf{n}+t\mathbf{a}$, $u\in\mathbb{R}$ which is the line going through $\mathbf{p}+t\mathbf{a}$ in direction $\mathbf{n}$. Let $u'(t)$ be the value of u where $l(t)$ intersects P, then

$$u'(t)=-\frac{\mathbf{p}\cdot\mathbf{n}+q+t\mathbf{a}\cdot\mathbf{n}}{||\mathbf{n}||^2}.$$

Inserting this into the equation for $l(t)$ we get a line representing the closest point of P to p:

$$l'(t):\mathbf{p}-\frac{\mathbf{p}\cdot\mathbf{n}+q}{||\mathbf{n}||^2}+t(\mathbf{a}-\frac{\mathbf{a}\cdot\mathbf{n}}{||\mathbf{n}||^2}\mathbf{n}).$$

Now, the interval of t I_p where the closest point on P is within T is the interval where l' intersects the triangle T which is determined by calculating the values of t for l''s entry and exit point of T (see Figure 5 (b)).

The minimal quadratic distance between l_i and T is calculated by combining the different distance measures over their respective valid intervals; For the interval where l_i and T intersects we set the distance to 0. For the remaining parts we calculate the minimum distance of either of the endpoint distances within their valid intervals and the line segment distances between l_i and the edges of T. Since each individual distance is composed of piecewise quadratic functions, we may combine them into one single piecewise quadratic function which gives the distance from l_i to T for any translation t along $\mathbf{a}$. Denote this piecewise quadratic function $d_{ST}(t,\mathbf{a},\mathbf{p}_i,\mathbf{v}_i,T)$.

We now generate a set of intervals containing feasible translations of i along $\mathbf{a}$. This may be generated by subtracting the intervals where $d_{ST}(t,\mathbf{a},\mathbf{p}_i,\mathbf{v}_i,T)<r_i^2$ for $T\in E$ from the set of intervals representing feasible translations of both endpoints of l_i, which may be done in $O(|E_s|\log|E_s|)$ time. Assume the resulting list of intervals with feasible translations is represented as a sorted list of t-values, $t_1,t_2,\ldots,t_n$, where each interval is represented as a pair of t-values $[t_j,t_{j+1}]$, $j\equiv 1 \mod 2$, then we create a piecewise linear function

$$f_E(\mathcal{P}(t),\mathbf{a},i)=\begin{cases} t_1-t & \text{for } t\le t_1 \\ t-t_2 & \text{for } t_2<t\le\frac{t_2+t_3}{2} \\ t_3-t & \text{for } \frac{t_2+t_3}{2}<t\le t_3 \\ t-t_4 & \text{for } t_4<t\le\frac{t_4+t_5}{2} \\ t_5-t & \text{for } \frac{t_4+t_5}{2}<t\le t_5 \\ \vdots & \\ t-t_n & \text{for } t>t_n \\ 0 & \text{otherwise} \end{cases},$$

which describes the amount we need to translate i along $\mathbf{a}$ for i to be placed feasibly within E for every value of t.

5.3 Link Constraints

The third contribution to $\mathcal{F}(t)$ is the set of link terms. Specifically we need to determine the value of $\sum_{j=k}^{m} f_L((\mathcal{P},k)$ for any translation of capsule i along vector $\mathbf{a}$. To do this we first note that the sum of all link-distances for links which are not incident with i, can be described as a constant. For each remaining link k incident to i we determine the value $f_L(\mathcal{P}(t),k)$ which is the value $f_L(\mathcal{P}(t),k)$ for the placement $\mathcal{P}(t)$ where i is translated t units in direction $\mathbf{a}$. Assume $i=s(k)$ ($i=e(k)$ is equivalent) and

let $d_b(k,t,\mathbf{a}) = ||\mathbf{p}_{s(k)} + \mathbf{v}_{s(k)} + t\mathbf{a} - \mathbf{p}_{e(k)}||^2$, then $f_L(\mathcal{P}(t),k) = \max(d_b(k,t,\mathbf{a}) - b(k)^2, 0)$ is a piecewise quadratic function:

$$f_L(\mathcal{P}(t),k) = \begin{cases} d_b(k,t,\mathbf{a}) - b(k)^2 & \text{for } d_b(k,t,\mathbf{a}) > b(k)^2 \\ 0 & \text{otherwise,} \end{cases}$$

where the interval $d_b(k,t,\mathbf{a}) > b(k)^2$ can be found by solving the quadratic equation $d_b(k,t,\mathbf{a}) = b(i)^2$.

5.4 Fast Neighborhood Search

As stated earlier, computation of $\mathcal{H}(\mathcal{P}(t))$ can be done by probing with each value t from a discrete set of values W. However, the size of W would depend on the desired resolution and dimensions of the coordinate system used for placements and it would impose limits on the set of feasible positions. Additionally, it would be inefficient since each probe naively would require asymptotic linear time in the number of capsules and size of envelope. Assuming that we consider translations where $t \in W$, then such an algorithm would have asymptotic running time $O(W(|E_s| + n + m))$. Instead we will present an algorithm which is independent of coordinate system resolution and dimensions and has running time $O((n + |E_s| + m)\log(n + |E_s| + m))$. The algorithm takes advantage of the fact that, since each of the individual terms of the objective function may be represented as piecewise quadratic functions over t, $\mathcal{H}(\mathcal{P}(t))$ may also be represented by a piecewise quadratic function over t.

For a capsule i, to be translated, the function $f_C(\mathcal{P}(t),\mathbf{a},i,j)$ for another capsule j, can be calculated in $O(1)$ time and results in a piecewise quadratic function with less than 5 pieces. For each link k incident with i, $f_L(\mathcal{P}(t),k)$ can be calculated in $O(1)$ time, and results in a piecewise quadratic function with less than 3 pieces. $f_E(\mathcal{P}(t),\mathbf{a},i)$ can be calculated in $O(|E_s|\log|E_s|)$ time and results in a piecewise quadratic function of less than $|E_s|+1$ pieces. The values $f_C(\mathcal{P}(t),\overline{\mathbf{v}}_{ij},k,j)$, $f_E(\mathcal{P}(t),\tilde{\mathbf{v}}_k,k)$ for $j,k \neq i$ and $f_L(\mathcal{P}(t),k)$ where link k is not incident with i, as well as the values $f_A(\mathcal{P}(t),k)$, for $k = 1,\ldots,m$ are constant for all values of t.

A set of piecewise quadratic functions with a total of k intervals may be summed to a global piecewise quadratic function $f(t)$ in asymptotic time $O(k\log k)$; First, sort the interval endpoints, enumerate the sorted endpoints as $t_1 \leq t_2 \leq \ldots \leq t_k$ and let (a_i,b_i,c_i) be the coefficients corresponding to the quadratic function that endpoint t_i arose from. Now, construct $f(t)$ by visiting each interval endpoint t_i in order $i = 1,\ldots,k$ and maintaining values values $(a,b,c)^T$, initially $\mathbf{0}$. As t_i is visited create a quadratic function over the interval $[t_{i-1},t_i]$ with coefficients a,b,c, and update a,b,c by either adding or subtracting a_i,b_i,c_i depending on whether t_i is an interval end or start.

Since the number of pieces of the piecewise quadratic functions which arise from evaluating the individual terms of $\mathcal{F}(\mathcal{P}(t))$ is less than $5n+3m+|E_s| = O(n+m+|E_s|)$, we can sum the complete piecewise quadratic function $\mathcal{F}(\mathcal{P}(t))$ in $O((n+|E_s|+m)\log(n+|E_s|+m))$ time. Analyzing the piecewise quadratic function $\mathcal{F}(\mathcal{P}(t))$ for its global minimum with respect to t may be done in the $O(n+m+|E_s|)$ time by analyzing each individual piece for local minimum and selecting the global minimum. Finally, the penalty function $\mathcal{Z}(\mathcal{P}(t))$ may be included, by considering where each of the piecewise quadratic functions are larger than zero and which violations they arise from. In total, we can calculate $\mathcal{H}(\mathcal{P}(t))$ and select the value of t which results in the global minimum of $\mathcal{H}(\mathcal{P}(t))$ in $O((n+|E_s|+m)\log(n+|E_s|+m))$ time.

6 Optimization Variants

The problem described in the previous sections is a decision problem, i.e. we wish determine a feasible placement under the given set of constraints. One may also consider optimization variants

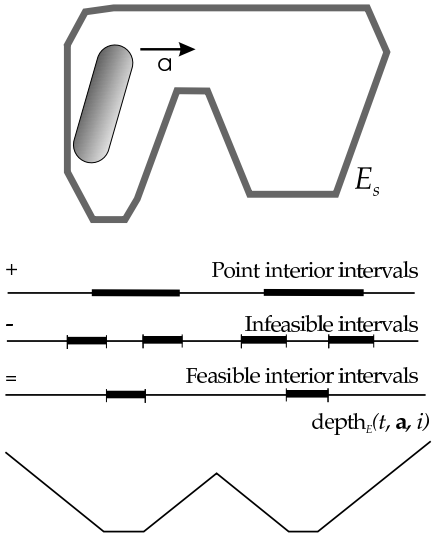


Figure 6: *Illustration of the evaluation of the f_E term. First, the intervals corresponding to translations where both points are within E are determined. Then the set of intervals where the distance between the capsule and E_s are subtracted which gives the feasible intervals. Finally, the f_E function is determined based on the feasible intervals.*

revolving around ICPDP, which, since ICPDP is $\mathcal{NP}$-complete, are $\mathcal{NP}$-hard. Here we will discuss two types of problems; The first group consists of problems where the objective can be optimized by solving a series of decision problems. The second group consists of problems where the objective is optimized by adding it to the decision objective function $\mathcal{H}(\mathcal{P})$. The examples we present here for the first group are container compaction problems and in the second group we consider adding surface placement requirements to the capsules.

6.1 Compaction Problems

In this section we consider three simple optimization variants in which container dimensions are minimized. The procedure we outline in the following was also used by Egeblad et al. [A, B] to solve two- and three-dimensional strip-packing problems.

Unlike the problem considered previously, we will assume in the following that we are given a convex container C instead of an envelope E. In the previous variant we allowed capsules outside the envelope E during the solution process, albeit at a cost of increased objective value. Here we will only consider translations within C during the solution process, so the f_E term is omitted from the objective function in this section.

The three container minimization problems we consider are as follows and are illustrated on Figure 7:

- Given two dimensions W and H determine a minimum value L such that a feasible placement of the capsules with respect to overlap, link and angle constraints can be found within the box-container $W \times H \times L$ (strip packing).
- Minimize L such that a feasible placement can be found within the cube $L \times L \times L$ (minimal cube packing).

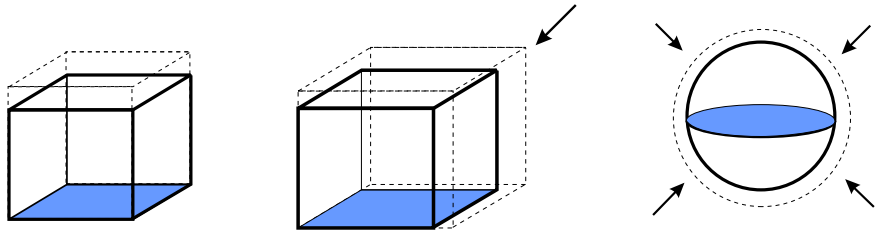


Figure 7: *The three different compaction objectives. (a) Minimize box-height. (b) Minimize cube. (c) Minimize sphere.*

- Minimize L such that a feasible placement can be found within a sphere of radius L (minimal sphere packing).

All three problems may be solved similarly. We begin with a sufficiently large value for the free container dimension, L_0. Once a solution has been found for the ICPDP problem with the free container dimensions L_0, we consider a problem with smaller container dimensions where $L_1 = L_0 - \varepsilon$, where ε is a step-size. We repeat this process so that the free container dimension in iteration i is $L_1 = L_0 - i \cdot \varepsilon$, although more complex strategies mimicking binary search may be applied. When the container dimensions are reduced, all capsules not within the new container are translated into the new smaller container, while the remaining capsules remain at their current position. This way, a large part of the placement is kept intact and less time is spend on finding a feasible placement. The search may end after a specific time-limit, where the smallest value L_i found so far, is returned as a solution to the problem.

6.2 Exposure Optimization

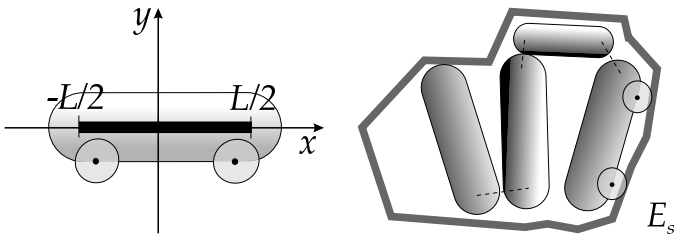
It may be possible to identify residues of the molecule that are exposed through experimental techniques. This can be used to aid the search, since they can be used to indicate which portions of the capsules must be near the surface.

To model this, we may create a set of spheres S corresponding to the identified residues. Each sphere is then assigned to the capsule that corresponds to the helix of the residue, by specifying a coordinate in the local coordinate system of the capsule (see figure 8 (a)). Now, during local search, the same transformation that applies to the capsule as described in the beginning of Section 2 also applies to the sphere, so that when the capsule is moved the sphere follows it. The objective is to ensure that each sphere $i \in S$, with radius r_i overlaps with the molecular surface E_s (see figure 8 (b)).

Rather than modeling this as hard constraint, we can formulate the problem as an optimization problem. In this variant our objective is to minimize the number of spheres not within a certain distance of the molecular surface. The objective function is as follows:

$$\text{minimize} \quad \mathcal{G}(\mathcal{P}) = \sum_{i \in S} w_i f_X(\mathcal{P}, \tilde{\mathbf{v}}_{c(i)}, i),$$

where w_i is a weight assigned to each sphere that can model its importance, $c(i)$ is capsule sphere i is attached to, and $f_X(\mathcal{P}, \tilde{\mathbf{v}}_{c(i)}, i)$ is the amount $c(i)$ must be translated for the center of i to be within r_i units of E_s.



(a) (b)

Figure 8: *(a) Illustration of two spheres which represent residues that are known to be exposed. These are described in the local coordinate system of the capsule they are attached to. (b) The residue-spheres follow the capsules under transformation. Here the two spheres attached to the capsule overlaps with the surface, as required.*

The complete optimization problem, where minimization of $\mathcal{G}(\mathcal{P})$ is combined with ICPDP and extra penalty terms are added, can now be formulated as follows:

$$\text{minimize } \mathcal{H}'(\mathcal{P}) = \mathcal{H}(\mathcal{P}) + \mathcal{G}(\mathcal{P}) + \lambda_P \sum_{i \in S} \eta_i I_P(i)$$

where η_i is a penalty value for sphere i, λ_P is used to fine-tune the heuristic, and $I_P(i)$ is an indicator function:

$$I_P(i) = \begin{cases} 1 & \text{for } f_X(\mathcal{P}, \tilde{\mathbf{v}}_{c(i)}, i)) > 0 \\ 0 & \text{otherwise} \end{cases}.$$

Whenever GLS reaches a local minimum and the penalty terms are modified (see Section 4.2), we also consider the sphere penalties, η_i, and increase the penalty for the sphere i with highest utility:

$$\chi(i) = \frac{f_X(\tilde{\mathbf{v}}_{c(i)}, i)}{\eta_i + 1}.$$

At the end of the search the feasible placement, with respect to the ICPDP, with least found value $\mathcal{G}(\mathcal{P})$ is returned.

Just as for the components of $\mathcal{F}(\mathcal{P})$, we determine $f_X(\mathcal{P}, \tilde{\mathbf{v}}_{c(i)}, i)$ for any translation of $c(i)$ along a vector $\mathbf{a}$. Let $q_i + t\mathbf{a}$ be the center of i, when translated t units along $\mathbf{a}$. Let $d_{S,E}(\mathbf{q} + t\mathbf{a})$ be the distance from $q_i + t\mathbf{a}$ to any triangle $T \in E_s$, then we determine feasible intervals $[t_1, t_2], \ldots, [t_{k-1}, t_k]$ (sorted in ascending order), such that $d_E(\mathbf{q} + t\mathbf{a}) < r_i$, for $t \in [t_{2j-1}, t_{2j}]$, $j \leq \frac{k}{2}$. Now the intervals $[t_1, t_2], \ldots, [t_{k-1}, t_k]$, represent translations of sphere i relative to $\mathcal{P}$ where sphere i overlaps with E_s. We can evaluate $d_E(\mathbf{q} + t\mathbf{a})$ and determine $t_1, \ldots, t_k$ in $O(|E_s| log |E_s|)$ time using the same strategy used to determine the distance between segment endpoints and E_s outlined in Section 5.2. We now let $f_X(\mathcal{P}(t), \tilde{\mathbf{v}}_{c(i)}, i)$, be:

$$f_X(\mathcal{P}(t), \tilde{\mathbf{v}}, i) \begin{cases} t_1 - t & \text{for } t \leq t_1 \\ 0 & \text{for } t \in [t_1, t_2] \\ t - t_2 & \text{for } t \in [t_2, \frac{t_3 + t_2}{2}) \\ t_3 - t & \text{for } t \in [\frac{t_3 + t_2}{2}], t_3) \\ \vdots & \\ t - t_k & \text{for } t \geq t_k \end{cases}$$

Since $f_X(\mathcal{P}(t), \tilde{\mathbf{v}}, i)$ is piecewise linear we can use the same strategies as mentioned in Section 5.4 to evaluate $\mathcal{H}'(\mathcal{P}(t))$ for all values of t. We must evaluate $f_X(t, \tilde{\mathbf{v}}, i)$ for each sphere i attached to the capsule of translation $c(i)$ so the total asymptotic running time required to evaluate $\mathcal{H}'(\mathcal{P}(t))$ for all values of t is $O((n+m+l|E_s|)\log(n+m+l|E_s|))$, where l is the number of spheres attached to the capsule $c(i)$ for which the set of possible translations are being examined.

It should be noted that, for this optimization variant rotations around the capsule axis must also be represented and included in the local search neighborhood, in order to ensure that the all solutions can be reached.

Hidden residues It is also straightforward to extend the model to include residues which are hidden. Let S^- be the set of sphere representing hidden residues, then we may solve:

$$\text{minimize} \quad \mathcal{G}(\mathcal{P}) = \sum_{i \in S} w_i f_X(\mathcal{P}, \tilde{\mathbf{v}}_{c(i)}, i) + \sum_{i \in S^-} w_i f_X^-(\mathcal{P}, \tilde{\mathbf{v}}_{c(i)}, i),$$

where $f_X^-(\mathcal{P}, \tilde{\mathbf{v}}_{c(i)}, i)$ is the amount $c(i)$ must be translated for the center of i to be more than r_i units from of E_s. Penalties are added in the same fashion as for exposed residues and the neighborhood can be searched in similar fashion.

7 Experimental Results

To investigate the performance of the heuristic it was implemented in C++ and compiled with GCC 4.2.3. All experiments were run on a computer with two Intel Xeon 5355 2.66 GHz quad core processors (8 cores total) and 8 GB RAM. No advantage was taken of the 8 core system. The set of rotation angles $\mathcal{A}$ was set to $\{0, \frac{1}{32}\pi, \ldots, \frac{31}{32}\pi\}$.

Suitable values of λ_C, λ_L, and λ_E were found using parameter tuning, and was set as follows $\lambda_C = 0.1 \cdot S$, $\lambda_L = 0.1 \cdot S$, and $\lambda_E = 0.5 \cdot S$ for $S = \sum_{i=1}^{n}(c_i + r_i)$.

Both compaction and decision variants of the problem were tested and will be describe in Section 7.1 and Section 7.2 respectively.

7.1 Results for Compaction

To test the method's ability to work as an heuristic for the compaction optimization problems listed in Section 6.1, three different types of problem instances were constructed; Homogeneous problems consisting of only one type of capsule, heterogeneous problems consisting of different types of capsules, and problems with capsule links.

For each major type of problem, instances for each of the three compaction variants ($V = \{\text{strip}, \text{cube}, \text{sphere}\}$) from Section 6.1 were randomly generated. Capsule radii were all set to 1 and lengths for the problems are taken from the list $L = \{0, 2, 8, 32\}$. The number of capsules in each instance were taken from the list $N = \{5, 10, 15, 25, 35, 50, 150\}$.

For all compaction instances, the heuristic was set to report the best result found within 250,000 iterations, which was found to deliver adequate convergence. To test the stability of the heuristic, each instance was run with 5 different random seeds for the random number generator.

7.1.1 Homogeneous Problems

For each length from $l \in L$, each number of capsules $n \in N$, and each optimization variant from V an instance was generated with n capsules and all capsule lengths set to l. This results in $3 \times 4 \times 7 = 84$

Homogeneous													
Length		0			2			8			32		
	n	Avg.	Std.	Time	Avg.	Std.	Time	Avg.	Std.	Time	Avg.	Std.	Time
Strip	5	31.4	0.71	42.7	42.3	0.00	37.4	43.3	0.00	0.3	34.1	0.36	43.7
	10	34.7	0.17	61.8	37.1	0.53	109.6	54.5	0.00	51.5	46.8	0.00	74.9
	15	39.1	0.19	83.0	38.8	0.19	164.0	45.7	0.41	123.0	41.6	0.69	119.1
	25	44.2	0.33	116.6	42.3	0.29	264.0	50.6	0.00	182.3	43.0	0.70	197.5
	35	44.9	0.15	139.2	45.1	0.40	357.0	42.0	0.39	302.1	42.8	0.61	225.8
	50	41.9	0.35	165.0	47.5	0.34	491.9	44.8	0.13	482.5	46.4	0.60	368.1
	150	43.1	0.17	312.4	47.9	0.38	1161.5	48.9	0.37	1066.7	51.0	2.26	874.1
Cube	5	37.4	1.41	42.6	37.5	0.00	68.1	39.8	1.64	59.7	42.4	1.06	50.8
	10	35.5	0.00	61.0	38.3	0.00	110.2	44.2	1.46	112.4	51.1	0.00	96.5
	15	39.7	0.00	76.0	41.9	1.38	155.8	49.3	0.00	144.7	48.1	1.10	135.8
	25	40.7	1.17	95.8	42.8	0.00	223.3	45.5	0.00	234.7	48.3	1.20	225.0
	35	42.5	1.12	120.9	44.7	1.12	317.8	47.5	0.89	311.8	49.8	0.87	296.0
	50	42.8	1.00	151.9	45.6	0.82	434.2	46.7	0.94	431.4	53.1	0.67	421.2
	150	44.8	0.00	338.0	47.2	0.60	1266.1	48.9	0.55	1253.2	51.1	0.00	1163.9
Sphere	5	35.5	0.00	39.0	39.3	0.00	73.7	44.2	2.94	63.8	37.2	1.42	33.7
	10	38.7	0.00	65.0	40.2	0.00	150.3	42.4	0.00	125.0	39.5	2.45	71.7
	15	43.3	0.00	86.1	43.3	0.00	200.2	42.8	0.00	169.3	41.7	0.00	126.9
	25	40.6	0.00	109.9	43.1	0.00	296.2	43.6	0.00	263.1	42.1	1.26	204.9
	35	42.1	1.75	134.4	44.1	1.40	387.3	44.5	1.60	343.4	42.8	0.00	281.6
	50	42.5	1.34	167.1	45.1	0.00	541.4	44.5	0.00	467.8	43.0	0.00	405.4
	150	44.2	0.93	363.2	46.3	0.00	1434.7	47.9	0.86	1303.9	43.8	0.00	1175.7

Table 1: Results of experiments with homogeneous instances. Results are shown for each of the four capsule lengths 0, 2, 8, 32, and each of three compaction variants (Strip, Cube, and Sphere). 'n' is the number of capsules. Each instance was run five times with 5 different random seeds. 'avg.' is the average utilization of the container in percent. 'std.' is the standard deviation of the utilization over the five runs. 'time' is the average running time in seconds over the five runs.

homogeneous instances. The results for the homogeneous instances are presented in Table 1.

In the table, the average *utilization* [Volume of items]/[Volume of container] from the 5 different runs are presented for each instance along with the standard deviation. Utilization levels are generally between 40 and 50 % even for instances with as many as a 150 capsules, although only between 30 and 40 % for the instances with 5 capsules. The high utilization in instances containing as much as 150 spheres indicates that the placement method scales well. Running times are between 30 seconds for the smallest instances and up-to 20 minutes for the largest. The standard deviation is generally between 0 and 2 utilization percentage points, which shows a high level of stability. The utilization is equivalent across the different compaction types, which demonstrates that the heuristic works well even for different types of containers. Examples of homogeneous solutions are displayed on Figure 9.

The instances where the length is zero are homogeneous sphere-packing instances. Johannes Kepler's conjecture, which was recently proved by Hales [11], states that an optimal packing of homogeneous spheres in an infinitely large box has a utilization of $\frac{\pi}{3\sqrt{2}} \approx 74.048\%$. However, for a low number of spheres such a packing may be impossible and the heuristic is not geared specifically towards homogeneous sphere packing so the utilization levels are promising.

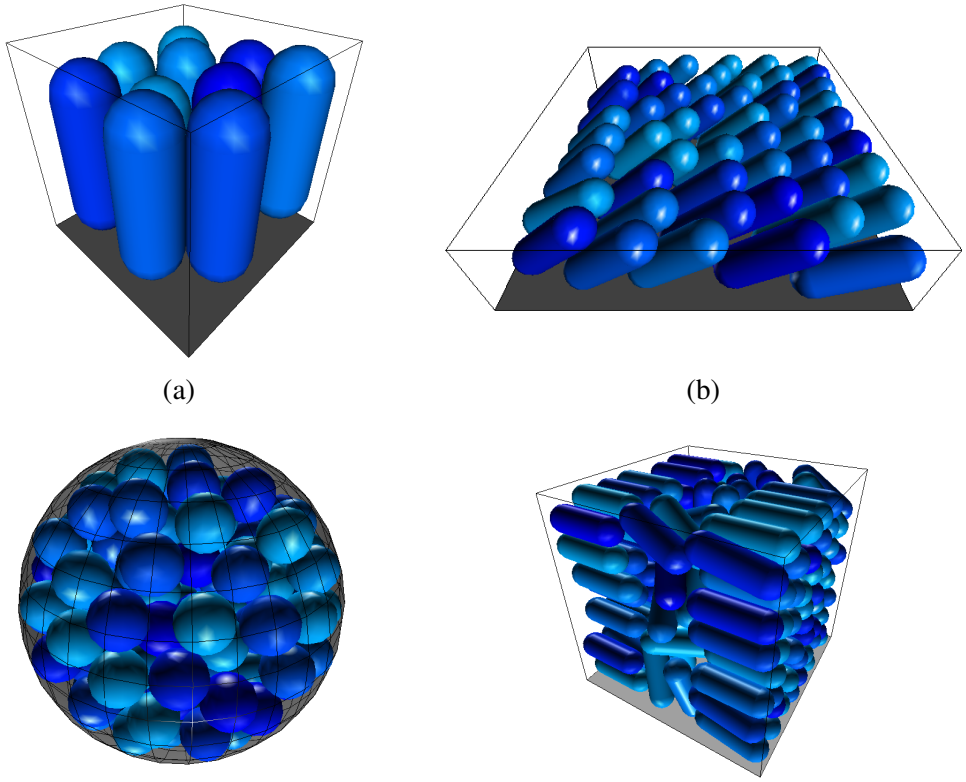


Figure 9: *Examples of homogeneous solutions. (a) 10 capsules of length 32 in minimal cube solution (utilization* 51%*). (b) 50 capsules of length 32 in minimal strip solution (utilization* 53%*). (c) 150 capsules of length 0 (spheres) in a minimal sphere (utilization* 45% *). (d) 150 capsules of length 32 in minimal cube (utilization* 51%*).*

Heterogeneous										
Iterations		50,000			125,000			250,000		
	n	Avg.	Std.	Time	Avg.	Std.	Time	Avg.	Std.	Time
Strip	5	43.0	0.00	0.3	43.0	0.00	0.3	43.0	0.00	0.3
	10	49.7	1.81	16.9	52.8	1.03	34.3	54.2	0.35	51.0
	15	43.0	0.42	28.3	43.6	0.32	69.8	44.1	0.37	138.7
	25	43.5	0.27	45.7	44.1	0.29	114.9	44.4	0.18	229.6
	35	43.9	0.41	61.5	44.6	0.28	161.0	44.9	0.19	323.9
	50	42.9	0.32	76.7	45.0	0.36	229.6	45.7	0.34	468.3
	150	17.8	0.00	159.7	34.8	0.15	409.5	47.3	0.28	1097.2
Cube	5	42.3	0.86	13.3	42.5	1.21	33.6	42.5	1.21	66.8
	10	43.1	0.35	23.2	43.3	0.65	58.2	44.0	1.39	117.0
	15	45.5	0.90	32.2	45.8	0.96	80.0	46.5	1.08	160.2
	25	45.3	0.79	50.0	45.5	0.56	123.4	45.7	0.51	246.1
	35	45.8	0.67	67.4	46.2	0.22	166.1	46.6	0.75	331.5
	50	46.3	0.23	93.3	46.6	0.19	230.8	46.7	0.39	456.6
	150	47.3	0.30	205.8	47.5	0.50	625.4	48.3	0.31	1288.4
Sphere	5	42.2	1.20	11.1	42.8	1.47	27.2	42.8	1.47	54.1
	10	43.8	0.93	22.6	43.8	0.93	57.4	44.1	0.47	116.4
	15	43.5	1.01	33.6	44.1	0.83	85.0	44.3	0.42	170.8
	25	43.4	0.70	51.2	44.5	0.89	127.6	44.7	0.81	256.8
	35	44.0	0.39	69.4	45.0	0.32	172.7	45.2	0.00	344.5
	50	44.6	0.65	95.9	45.2	0.60	236.5	45.3	0.67	476.1
	150	46.4	0.46	237.0	46.6	0.47	660.8	46.8	0.22	1331.0

Table 2: Results for the heterogeneous instances. Results are shown after 50,000, 100,000, and 250,000 iterations to illustrate the convergence of the heuristic. For each instance the average of the five runs on the three different instances is reported. See Table 1 for a description of labels.

7.1.2 Heterogeneous Problems

For the heterogeneous problems, instances were generated random, with capsule radii set to 1 and lengths from L. Four instances were generated for each optimization variant from V and each value of $n \in N$ given a total of $3 \times 3 \times 7 = 63$ heterogeneous instances. Results of the heterogeneous instances are presented in Table 1

The results of the heterogenous instances for 250,000 iterations matches those of the homogeneous instances. Utilization is generally between 40 and 50 % and even matches 50 %. Standard deviation remains below 2 utilization percentage points. The results also show that the heuristic converges rapidly. For the small instances containing 5-15 capsules there is little improvement between 50,000 to 250,000 iterations. For the larger instances containing up-to 50 capsules the improvement between 125,000 and 250,000 iterations is less than a single percentage point. For a 150 capsules good results are only reached with 250,000 iteration in the strip-packing variant, while the last 125,000 iterations for the other variants show little improvement. Example solutions are shown on Figure 10.

No other published results exists for packing problems involving capsules, but the best known results for three-dimensional strip-packing of polyhedra yields a utilization of between 40 and 55 %

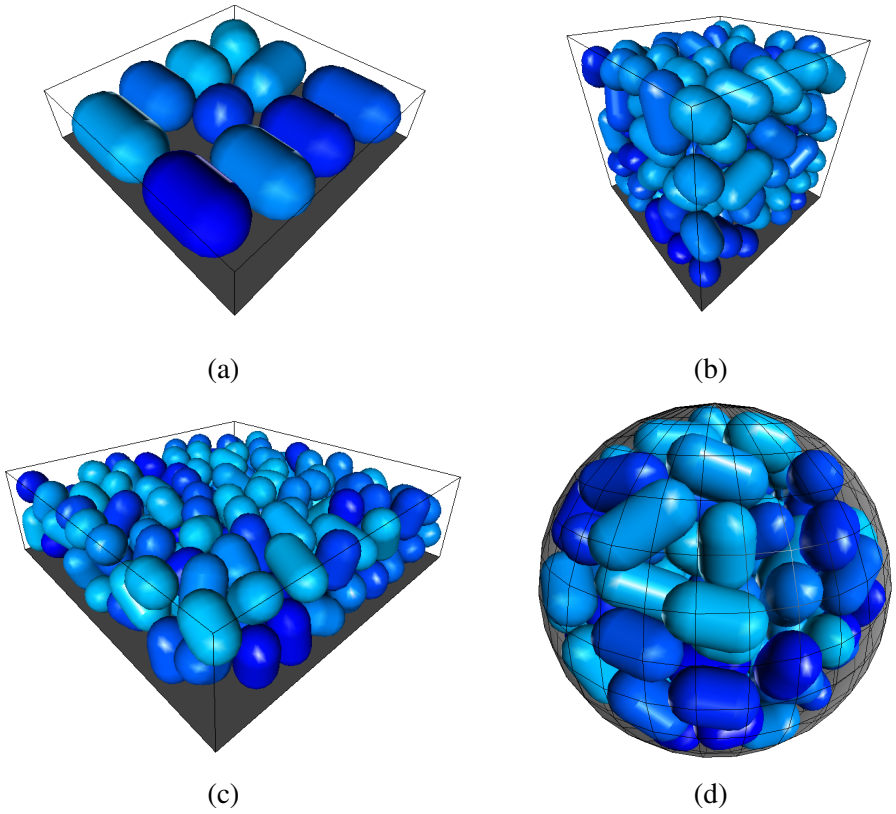


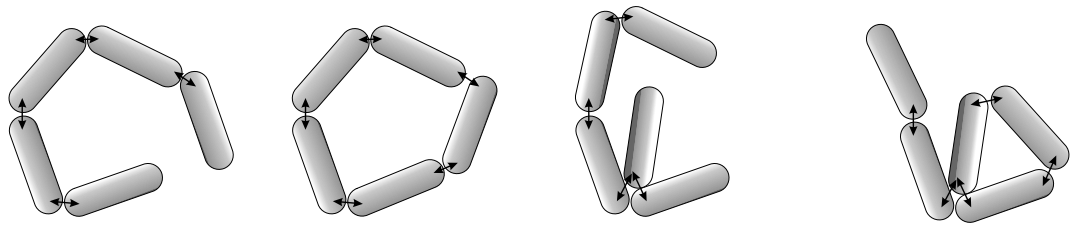
Figure 10: *Examples of heterogeneous solutions. (a) 10 capsules in minimal height container (utilization* 55%*). (b) 150 capsules in minimal cube container (utilization* 58%*). (c) 150 capsules in minimal height container (utilization* 47% *). (d) 150 capsules in minimal sphere container (utilization* 47%*).*

as presented by Egeblad et al. [B], so the utilization levels reached by our method are promising.

7.1.3 Problems with Links

A number of instances with linked capsules were randomly generated and tested to investigate the method's ability to find feasible placements under compact conditions. Capsules were linked in four different ways (See Figure 11):

- As an open chain of capsules where capsule i is linked to capsule $i+1$ (Figure 11 (a)).
- As a closed chain of capsules where capsule i is linked to capsule $i+1$ and capsule n is linked to capsule 1. (Figure 11 (b)).
- As an open chain consisting of single links or 'T'-intersection links, where one endpoint of at least one capsule is connected to two other capsules. The chain is acyclic. (Figure 11 (c)).
- As a closed chain consisting of single links or 'T'-intersection links, where capsule i may be connected to both capsule j and capsule k. The chain consist of at least one cyclic sub-chain. (Figure 11 (d)).



(a) (b) (c) (d)

Figure 11: *The different types of instances for experiments. (a) Open chain. (b) Closed chain. (c) Open T-chain. (d) Closed T-chain.*

The capsules were generated as the heterogeneous instances described in Section 7.1.2 for each of the four types of links. This gives a total of $4 \times 63 = 252$ instances with linked capsules. The results of the instances with linked capsules are given in Table 3.

Results for all of the four different types of chains are promising. Generally, utilization levels of over 40 % are reached which matches the instances without links, and shows that the heuristic handles the extra constraints imposed by adding links extremely well.

Although the randomly generated instances with closed T-chains may be infeasible, i.e. it is unknown if a valid solution for the links exists, close inspection of the data revealed that the heuristic was able to find feasible placements for all instances, and only failed in 8, 10, and 11 runs for respectively strip, cube, and sphere packing of the instances containing 150 capsules. There the heuristic handles difficult link constraints for instances with up-to 50 capsules well and in compact placements, while open and closed chain compaction problems are dealt with even for 150 capsules.

7.2 Decision Problems

The RNA structure P4-P6 RNA was modeled to test the performance of the heuristic for problems where only a feasible placement must be found within an envelope. The structure consists of *158 nucleotides* and 8 helical regions (See figure 14 (a)). The 8 helical regions were contracted into *7 helices* and converted into an instance of the capsule placement problem as illustrated on Figure 14 (b).

A crystal structure was used to identify the actual position of each nucleotide in the RNA molecule. The nucleotides of each helix were identified and the center axis of each helix was found by linear least square fitting of the positions of the nucleotides in the crystal structure. The radius of each helix was determined as the maximum distance from the axis to the center of any of the involved nucleotides. Links were added between capsule for which the associated helices were neighbors in the backbone of the RNA, and the required distance between two capsules i and j was set to the total distance between the last nucleotide of i and the first nucleotide j on the backbone.

Additionally, an molecular surface was generated using Small-angle X-ray scattering (SAXS) and converted into a triangle mesh consisting of *880* triangles which was used to represent an envelope.

The generated instances was tested with 375 different random seeds. The results of the 375 test-runs are summarized in Table 4. In 318 (85%) of the 375 test runs an actual placement within the envelope was found. Each run took on average 445 seconds, but with the fastest run taking less than 4 minutes and the slowest almost 105 minutes.

With Proximity Constraints													
		Strip				Cube				Sphere			
	n	Avg.	Std.	Time	Fail	Avg.	Std.	Time	Fail	Avg.	Std.	Time	Fail
Open chain	5	43.0	0.00	0.3	-	45.2	0.52	75.3	-	42.1	0.00	62.8	-
	10	49.9	1.12	95.2	-	43.0	0.47	131.0	-	43.7	0.00	126.3	-
	15	42.2	0.26	166.6	-	44.4	1.13	183.9	-	41.7	1.17	186.7	-
	25	42.0	0.44	276.6	-	43.6	0.94	277.5	-	42.0	0.00	283.3	-
	35	41.3	0.32	378.2	-	43.7	0.84	366.5	-	42.0	0.00	376.3	-
	50	41.3	0.34	508.7	-	43.6	0.57	498.6	-	42.5	0.81	513.2	-
	150	38.4	0.43	1163.9	-	42.6	0.47	1352.8	-	42.0	0.86	1424.3	-
Closed chain	5	43.0	0.00	0.3	-	45.2	0.57	76.3	-	42.1	0.00	68.6	-
	10	48.2	1.23	98.6	-	42.8	0.84	134.3	-	43.7	0.00	128.2	-
	15	42.1	0.34	169.0	-	44.1	0.77	185.5	-	41.7	0.50	186.9	-
	25	41.5	0.41	277.3	-	43.4	0.94	280.6	-	41.8	1.31	284.0	-
	35	40.9	0.31	379.1	-	43.6	0.83	370.1	-	42.0	0.80	374.2	-
	50	41.0	0.35	509.5	-	43.1	0.93	500.1	-	42.5	0.81	513.4	-
	150	38.4	0.48	1165.6	-	42.0	0.60	1352.4	-	41.5	0.92	1419.6	-
Open T-chain	5	43.0	0.00	0.5	-	44.7	0.46	75.0	-	42.1	0.00	62.4	-
	10	48.1	1.03	97.8	-	43.0	0.47	132.1	-	43.4	0.61	126.6	-
	15	42.2	0.32	167.9	-	44.6	1.21	184.5	-	42.2	0.56	188.3	-
	25	42.0	0.43	279.5	-	43.2	0.85	279.3	-	42.7	0.54	286.4	-
	35	41.3	0.47	381.5	-	43.7	0.32	370.9	-	42.6	0.92	381.0	-
	50	40.9	0.48	512.0	-	43.6	0.46	505.4	-	42.7	1.18	519.9	-
	150	38.0	0.53	1175.8	-	42.5	0.48	1370.6	-	42.0	0.55	1442.7	-
Closed T-chain	5	43.0	0.00	1.1	-	43.0	2.79	76.6	-	42.1	0.00	67.0	-
	10	39.7	0.94	116.4	-	40.6	0.90	135.0	-	41.4	2.03	133.6	-
	15	39.2	0.56	178.9	-	42.4	1.28	187.3	-	40.4	0.49	188.3	-
	25	37.6	0.53	295.3	-	41.4	1.05	287.1	-	40.1	0.43	291.9	-
	35	38.4	0.63	392.4	-	40.8	1.11	378.6	-	40.9	1.14	390.9	-
	50	36.9	1.24	518.0	-	40.7	0.92	518.7	-	40.1	1.06	528.0	-
	150	17.6	1.18	1403.6	8	26.3	2.48	1477.8	10	24.4	0.84	1528.7	11

Table 3: Results of experiments with instances with link constraints. Results are presented for each of the four different types of chains and each of the three different compaction goals. Results for each instance type covers the average result of three instances where each instance has been run 5 times. See table 1 for a description of labels. The column 'Fail' contains the number of the 15 runs of each instance type where no placement could be found within the maximum iteration limit. '-' indicates that a feasible placement was found for all instances in all runs of the designated instance type.

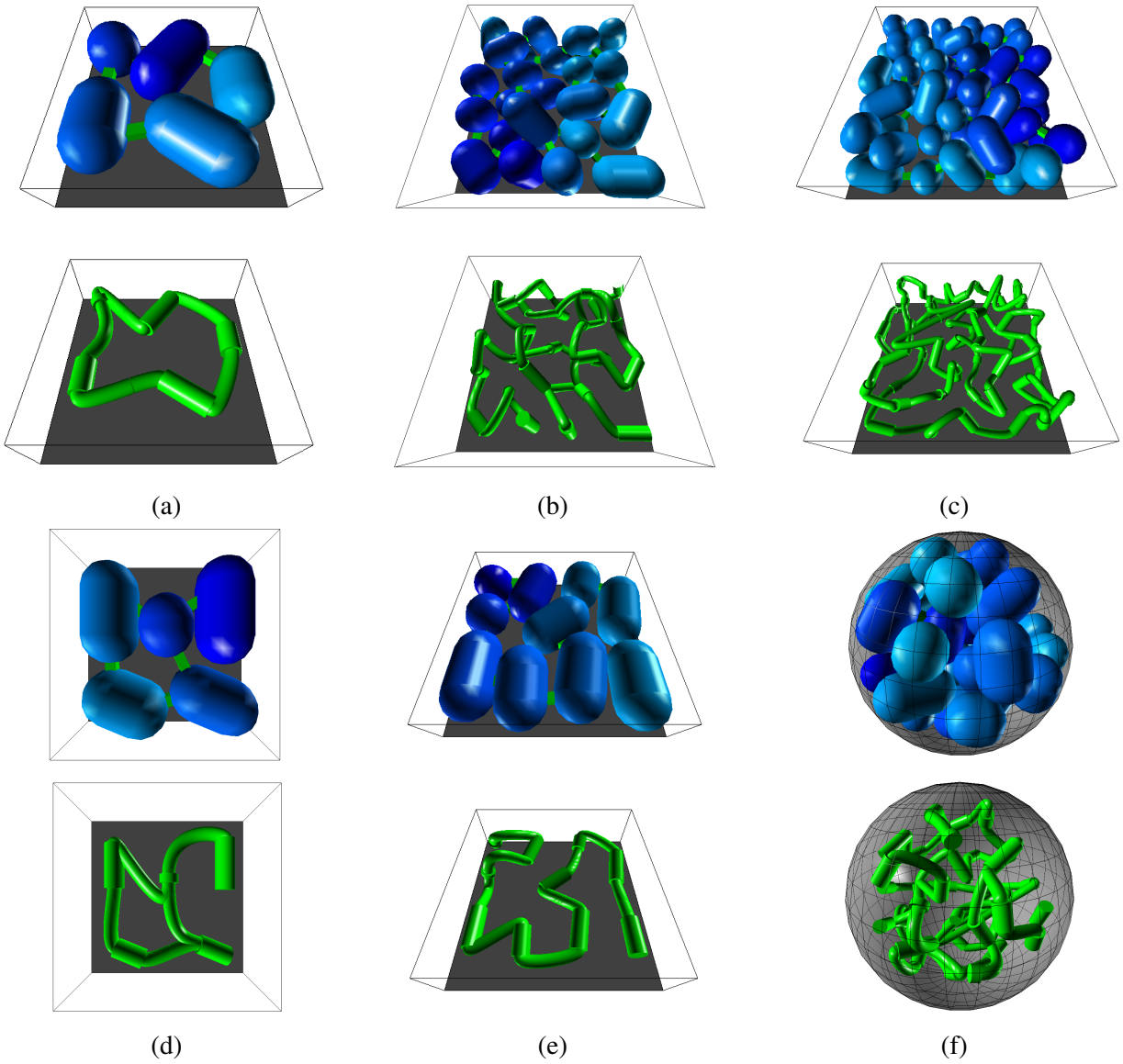


Figure 12: *Example results for problems with capsule links. First row of each example illustrates the capsule placement, and the second row illustrations connectivity. (a) 5 capsules in a closed chain (minimal height). (b) 25 capsules in a closed T-chain (minimal height). (c) 50 capsules in a closed loop (minimal height). (d) 5 capsules in a closed T-chain (minimal height). (e) 10 capsules in an open loop (minimal height). (f) 35 capsules in an open T-chain (minimal sphere).*

P4-P6 RNA (PDB ID: 1GID)				
Utilization	Avg. RMSD	Min. RMSD	Max. RMSD	Std. RMSD
30.5%	19.07	10.74	25.44	2.73
Success	Avg. Time	Min. Time	Max. Time	Std. Time
318/375 (85%)	445.0	221.7	6303.9	570.0

Table 4: Overview of the results from the envelope decision test. RMSD values are in Å, and time values are in seconds. 'Std. RMSD' and 'Std. Time' are standard deviations of the time and RMSD.

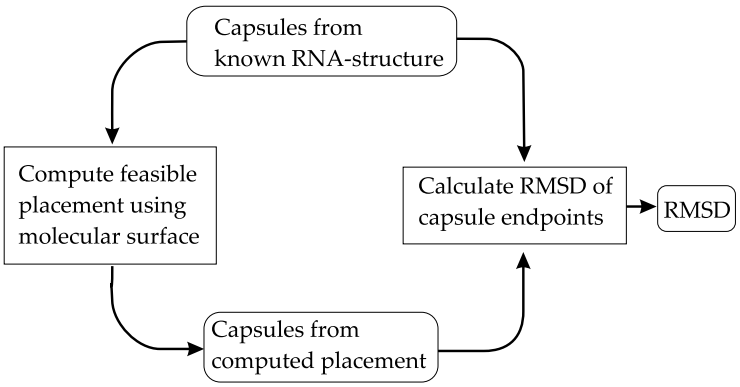


Figure 13: *An accurate known RNA molecular is converted into a ICPDP and a placement is found within the molecular envelope using our procedure. The endpoints of the capsule line-segments of the resulting placement were compared to the endpoints of the capsule line-segments from the known RNA structure, and the resulting RMSD reported.*

The resulting placements were compared with the input-structure, by measuring the root mean square deviation (RMSD) between the endpoints of the capsules from the crystal structure and the endpoints of the capsules from the each solution. The method is illustrated on Figure 13.

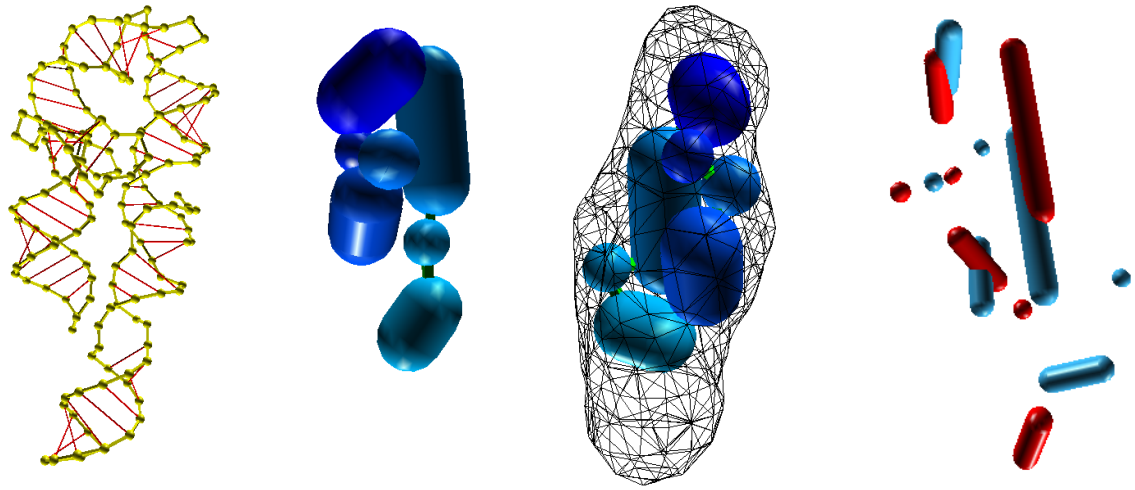
The placements had an average RMSD distance from the input structure of approximately 23 $\mathring{A}$, while the minimal RMSD distance was 10.74$\mathring{A}$ and the maximal 29.6$\mathring{A}$. The placement with minimal RMSD found is illustrated on Figure 14 (c) and showed with the target structure on Figure 14 (d).

8 Conclusion

We have introduced a simple coarse grained model for RNA tertiary structure prediction in which helical regions are converted into interconnected capsules. An efficient method capable of finding feasible layouts of the capsules within a molecular envelope was described. The method is based on a local search scheme in which each capsule is translated in one of four directions or rotated such that the feasibility of the placement is increased with each change. Finding an improving position is done efficiently using a polynomial time algorithm.

The resulting paradigm can be used not only for finding a feasible placement of the capsules, but also for solving optimization variants of the problem in which a compact placement is desired.

The compaction heuristics reveal promising results with utilization levels around 50% for minimal height box packing, minimal cube packing, and minimal sphere packing problems. This matches



(a) (b) (c) (d)

Figure 14: *(a) The nucleotides and base-pairs of the P4-P6 RNA used for testing the coarse grained model. (b) Layout of the capsules determined from the P4-P6 RNA structure. (c) Placement of capsules from structure with minimal RMSD. (d) Overlay of the structures from (b) and (c) with capsule radii divided by 4 for a clearer comparison.*

previous publicized work from the literature for non-rectangular shapes in three dimensions. The heuristic handles additional connection constraints between capsules well and is able to find highly compact placements with connection constraints of up-to 150 capsules within 20 minutes.

Experiments with modeling an RNA structure consisting of more than a 150 nucleotides as a set of capsules within a molecular envelope reveals promising results and further refinement of the resulting placement may lead to a more accurate prediction of the actual structure. This shows that the method has the potential to become a valuable tool for tertiary RNA structure prediction.

Further analysis of the procedure presented in this paper with RNA structures may reveal if the procedure is capable of accurately prediction structures of hundreds of nucleotides. Additionally, the model may be extended to include energy potentials or other information which may be increase the accuracy of this coarse grained method.

References

[A] J. Egeblad, B. K. Nielsen, and A. Odgaard. Fast neighborhood search for two- and three-dimensional nesting problems. *European Journal of Operational Research*, 183(3):1249–1266, 2007.

[B] J. Egeblad, B. K. Nielsen, and M. Brazil. Translational packing of arbitrary polytopes. *CGTA. Computational Geometry: Theory and Applications*, 2008. accepted for publication.

[1] J. Cagan, K. Shimada, and S. Yin. A survey of computational approaches to three-dimensional layout problems. *Computer-Added Design*, 34:597–611, 2002.

[2] E. Capriotti and M. A. Marti-Renom. Computational rna structure prediction. *Current Bioninformatics*, 3:32–45, 2008.

[3] R. Das and D. Baker. Automated de novo prediction of native-like rna tertiary structures. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 104(37), pages 14664–14669, 2007.

[4] F. Ding, S. Sharma, P. Chalasani, V. V. Demidov, N. E. Broude, and N. V. Dokholyan. Ab initio rna folding by discrete molecular dynamics: From structure prediction to folding mechanisms. *RNA*, 14:1164–1173, 2008.

[5] C. B. Do, D. A. Woods, and S. Batzoglou. Contrafold: Rna secondary structure prediction without physics-based models. *Bioinformatics*, 22(14):e90–e98, 2006.

[6] O. Faroe, D. Pisinger, and M. Zachariasen. Guided local search for final placement in vlsi design. *Journal of Heuristics*, 9(3):269–295, 2003. ISSN 1381-1231.

[7] 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.

[8] M. Garey and D. Johnson. *Computers and intractability. A Guide to the theory of NP-completeness.* W. H. Freeman and Company, New York, 1979.

[9] F. Glover. Tabu search - part 1. *ORSA Journal on computing*, 1(3):190–206, 1989.

[10] F. Glover. Tabu search - part 1i. *ORSA Journal on computing*, 2(1):4–32, 1990.

[11] T. C. Hales. A proof of the kepler conjecture. *Annals of Mathematics*, 162:1065–1185, 2005.

[12] C Hyeon, R. I. Dima, and D. Thirumalai. Size, shape, and flexibility of rna structures. *The Journal of Chemical Physics*, 125(19):194905, 2006.

[13] T. Imamichi and N. Hiroshi. *A Multi-sphere Scheme for 2D and 3D Packing Problems*, volume 4638/2007, pages 207–211. 2007.

[14] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[15] F. Major. Building three-dimensional ribonucleic acid structures. *Computating in Science and Engineering*, 5:44–53, 2003.

[16] H. M. Martinez, J. V. Maizel, and B. Shapiro. Rna2d3d: A program for generating, viewing andccomparing 3-dimensional models of rna. *Journal of Biomolecular Structure and Dynamics*, 25(6):669–683, 2008.

[17] C. Massire and E. Westhof. Manip: an interacivve tool for modelling rna. *Journal of Molecular Graphics and Modelling*, 16:197–205, 255–257, 1998.

[18] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA., 1996.

[19] Z. C. Muller. Three-dimensional comparative modeling of rna. *Nucleic Acids Symposium Series*, 36:69–71, 1997.

[20] B. A. Shapiro, T. G. Yingling, W. Kasprzak, and Bindewald E. Bridging the gap in rna structure prediction. *Current opinion in Structural Biology*, 17:157–165, 2007.

[21] Y. Stoyan and et al. Packing of various radii solid spheres into a parallelepiped, 2001. URL `citeseer.ist.psu.edu/stoyan01packing.html`.

[22] Y. Stoyan, N. I. Gil, G. Scheithauer, A. Pankratov, and I. Magdalina. Packing of convex polytopes into a parallelepiped. *Optimization*, 54(2):215–235, 2005.

[23] C. Voudouris and E. Tsang. Guided local search. Technical Report CSM-147, Department of Computer Science, University of Essex, Colchester, C04 3SQ, UK, August 1995.

[24] S. Yin, J. Cagan, and P. Hodges. Layout optimization of shapeable components with extended pattern search applied to transmission design. *Journal of Mechanical Design*, 126(1):188–191, 2004. doi: 10.1115/1.1637663.

[25] M. Zuker, D. H. Mathews, and D. H. Turner. Algorithms and thermodynamics for rna secondary structure prediction: A practical guide. In Clark BFC Edited by Barciszewski J, editor, *RNA Biochemistry and Biotechnology*. NATO ASI Series Kluwer Academic Publishers, 1999.