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-paknings-problem. 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.

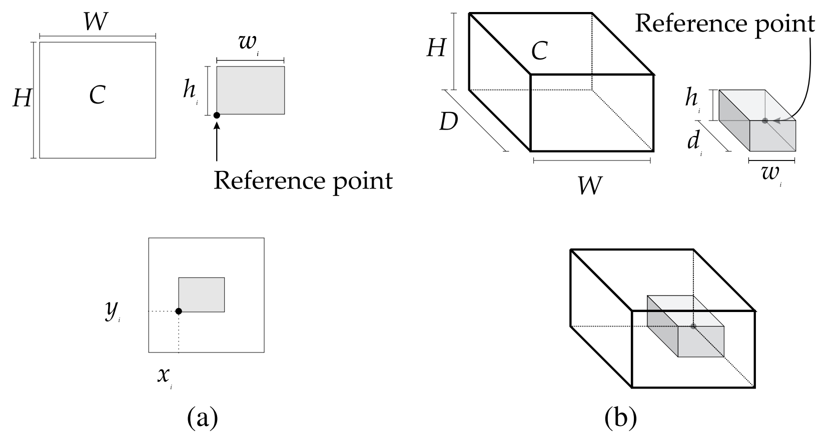


Figure 2: *(a) A two-dimensional container, item, and placement of an item within the container. (b) Three-dimensional variant.*

- **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.

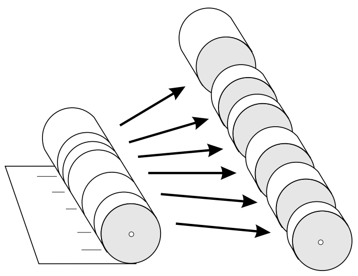
An important part of evaluating solutions to packing problems in general is the *utilization*. The utilization of a placement is the area or volume occupied by the items within the container divided by the the area or volume of the container.

2.1 Problem Types

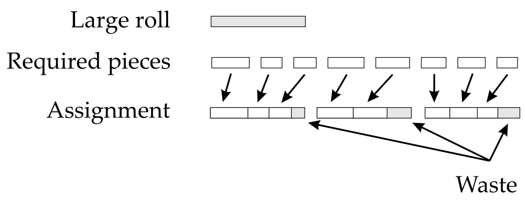
One-dimensional problems are in a sense the simplest type of packing problems, and we will begin by, briefly, considering them, in order to move on to the more complicated higher dimensional problems which are the focus of this text.

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)).



(a)



(b)

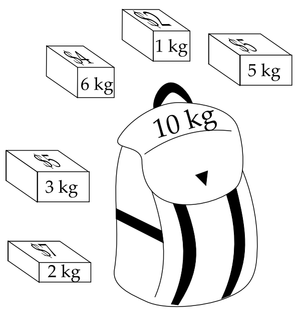
Figure 3: *The 1D cutting stock problem. (a) Large rolls must be cut into several smaller rolls. (b) Given large roll width, determine the number of large rolls required to cut the required pieces.*

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):

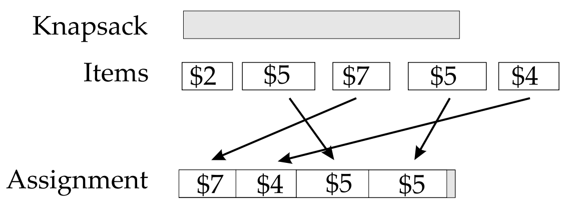
$$\begin{aligned}
\text{min} \quad & \sum_{k=1}^{K} y_k, \\
\text{s.t.} \quad & \sum_{k=1}^{K} x_{ik} \geq 1, \;\; i \in I \\
& \sum_{i \in I} w_i x_{ik} \leq W y_k, \;\; k = 1, \ldots, K \\
\text{where} \quad & x_i \in \{0,1\}, \;\; i \in I \\
& y_k \in \{0,1\}, \;\; k = 1, \ldots, K.
\end{aligned}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

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



(a)



(b)

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.

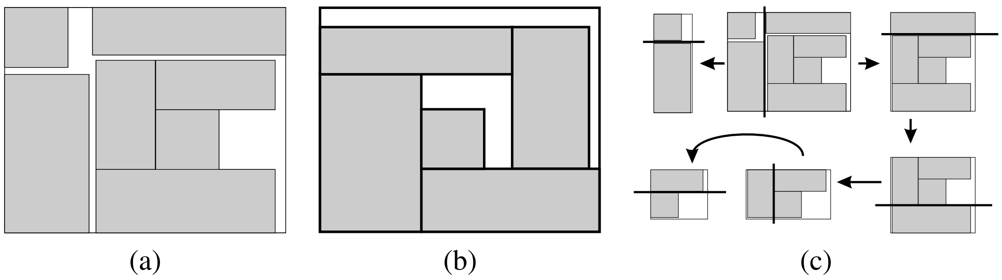


Figure 5: *Guillotine cutting. (a) Guillotine cuttable placement. (b) Non-guillotine cuttable placement. No sequence of cuts can cut all the items from the plate without cutting an item. (c) Cutting sequence for (a).*

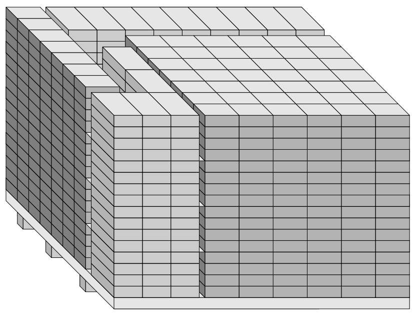


Figure 6: *The manufacturer's pallet loading problem is to determine the optimal loading of a set of identical items on a pallet.*

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 $^\circ$, 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,

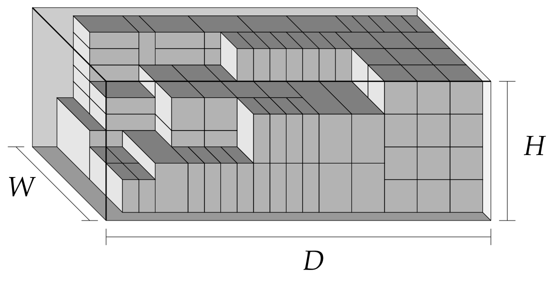


Figure 7: *The container loading problem and the common use of width (W), height (H), and depth (D).*

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

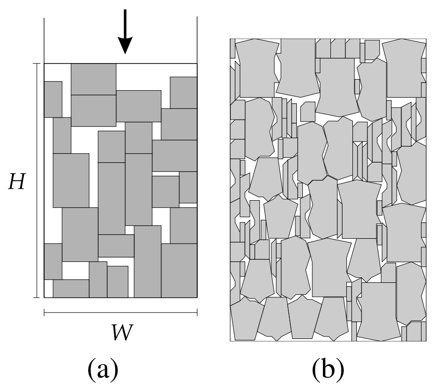


Figure 8: *(a) The strip-packing problem. The height (H), also referred to as the length, of the container must be minimized. (b) Example packing of irregular items.*

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.

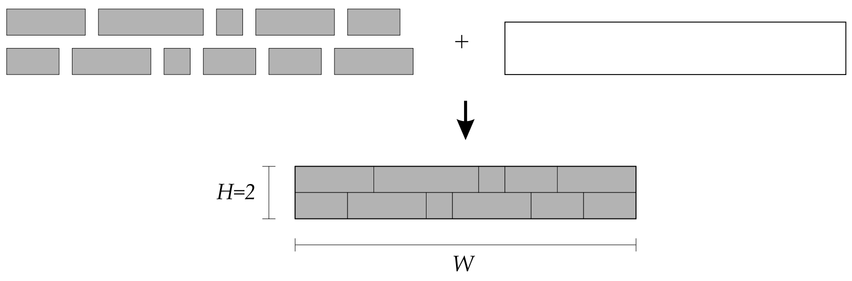


Figure 9: *Illustration of an instance of 2DPDP based on an instance of SPP used to prove that 2DPDP is $\mathcal{NP}$-complete. The height of all rectangles are* 1 *and W is the sum of all the widths divided by two. The solution of the 2DPDP problem, shown on the bottom, is also a solution to the SPP 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 $f(\mathbf{p}) : \mathbb{R}^d \rightarrow \{0,1\}$

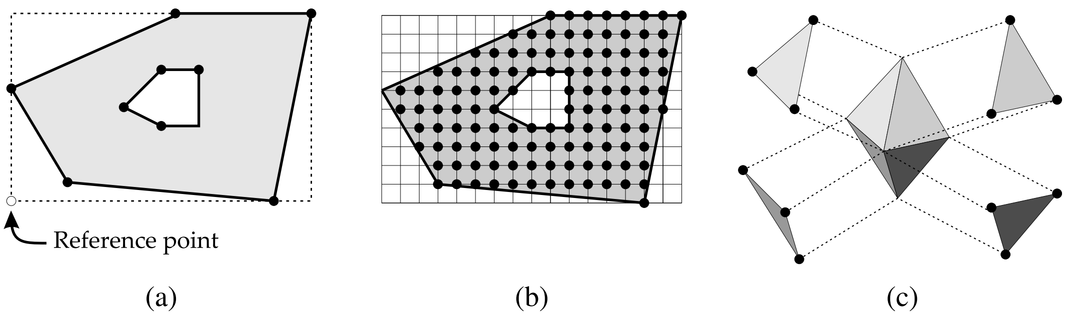


Figure 10: *Polygons and polyhedra. (a) Polygon with hole with its bounding-box. The polygon is represented by the endpoints of the edges (black circles). (b) Raster representation of the polygon. Black circles indicate positions where the raster-function is 1. (c) Polyhedron represented as a number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

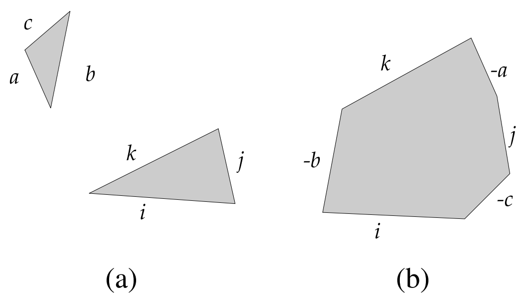


Figure 11: *The clockwise sorting principle for NFP generation. (a) Two convex polygons. The edges of the polygons are a,b,c and i,j,k respectively (sorted by slope). (b) The NFP of the two polygons. The polygon with edges a,b,c has been negated to the polygon with edges $-a,-b,-c$. The NFP can now be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

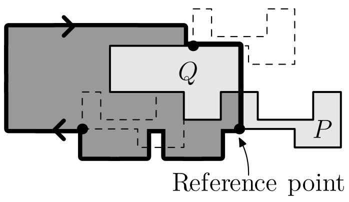


Figure 12: *The NFP of two polygons P and Q. The boundary of the NFP corresponds to the position of P as P is translated around Q.*

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/

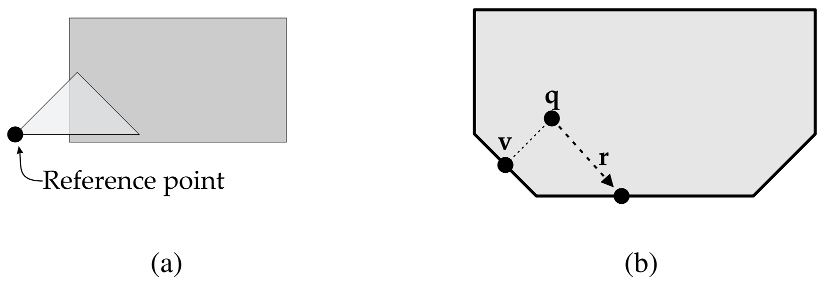


Figure 13: *(a) An overlapping triangle and retangle. (b) Their NFP.* $(\mathbf{p}-\mathbf{q})$ *is the relative position of the two polygons.* $\mathbf{v}$ *is the closest point to* $(\mathbf{p}-\mathbf{q})$ *on the boundary and the translation corresponding to* $\mathbf{v}-(\mathbf{p}-\mathbf{q})$ *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 $\mathrm{int}(s_1(\mathbf{p})) \cap \mathrm{int}(s_2(\mathbf{q})) = \emptyset$ 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 $\mathrm{int}(s_1(\mathbf{p})) \cap \mathrm{int}(s_2(\mathbf{q})) \neq \emptyset$.

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

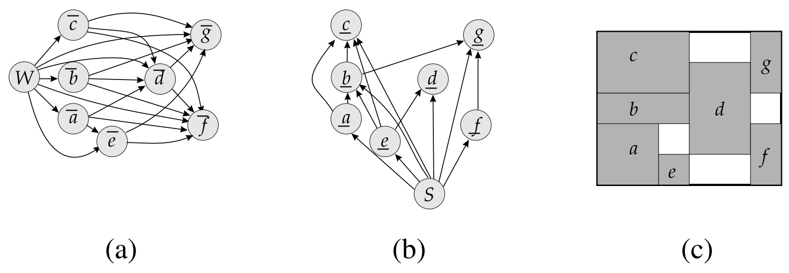


Figure 14: *Constraint graphs for two-dimensional placement of rectangles a, b, c, d, e, f and g.. (a) G_h. (b) G_v. (c) Resulting placement. Edge weights have been removed but are respectively width and height of the rectangles for G_h and G_v.*

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:

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

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

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:

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

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

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 $k = 1,\dots,d$ 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:

$$\min \sum_{i=1}^{M} x_i$$

$$\begin{array}{lll}
s.t. & & \\
& \sum_{i=1}^{M} a_{ij} x_j = 1 & (i = 1,\dots,n) \\
& x_j \in \{0,1\} & (j = 1,\dots,n),
\end{array}$$

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

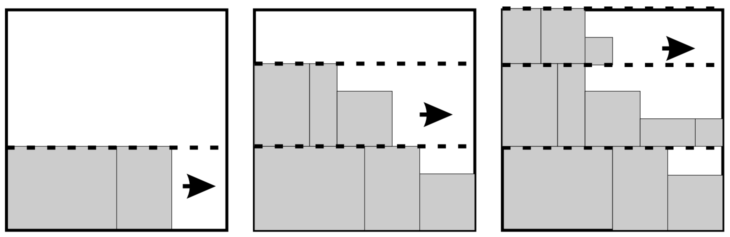


Figure 15: Levels *or* Shelves. *The container is filled from left to right in row, with the rectangles of a row "standing" on the bottom of that row. The height of a row is equal to the height of the tallest item.*

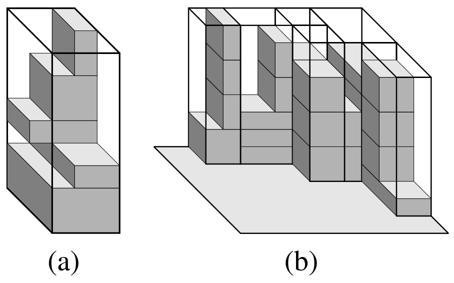


Figure 16: *Stacks. (a) A stack of 5 items. (b) A placement consisting of 6 stacks.*

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

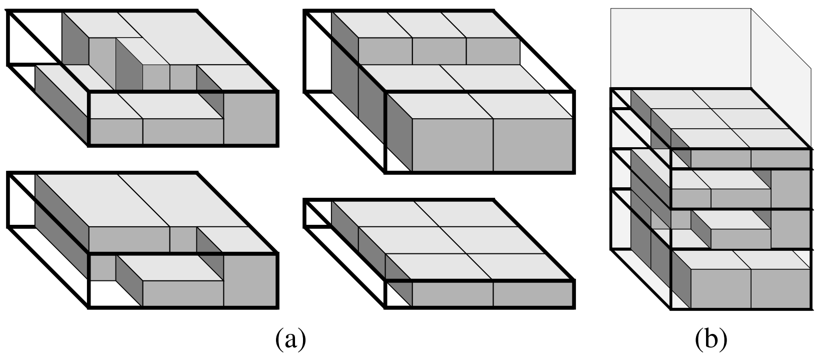


Figure 17: *Layers. (a) 4 layers. (b) A placement consisting of the four layers placed on top of each other.*

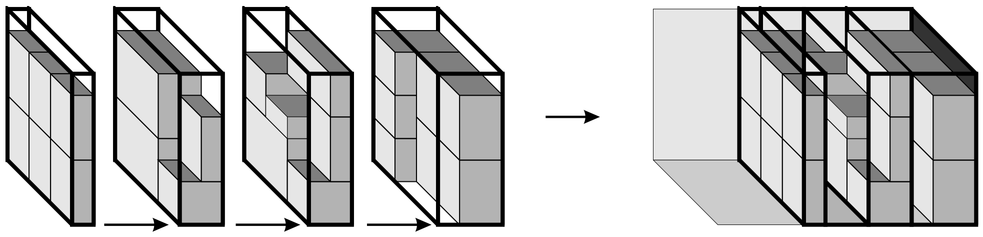


Figure 18: *Wall-building. 4 Walls are constructed and placed in a container.*

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-

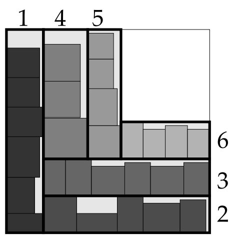


Figure 19: *When filling each wall during wall-building Pisinger recursively fills the wall with horizontal and vertical strips using tree-search. Here six strips are used; first a vertical, then two horizontal strips, then two vertical strips and finally one horizontal strip.*

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 $a,\dots,h$ as indicated on Figure 20 (a). It should be noted that the values of $a,\dots,h$ can be chosen from a subset of the values $1,\dots,W$ and $1,\dots,H$ 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.

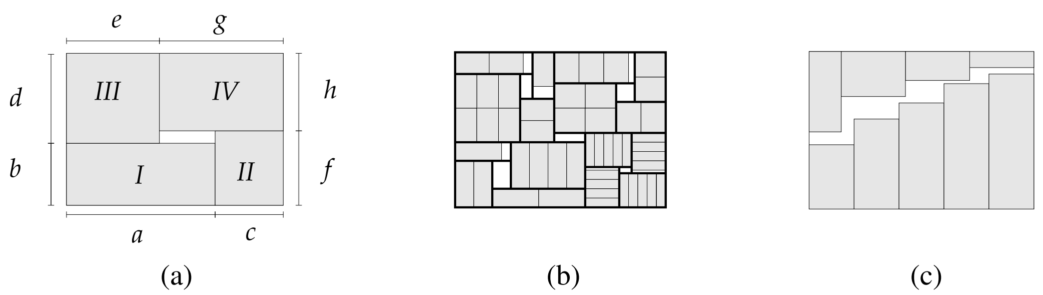


Figure 20: *(a) G4 structure consisting of 4 blocks. (b) Placement using the G4 structure with four item types. (c) Placement which cannot be represented by a G4 structure.*

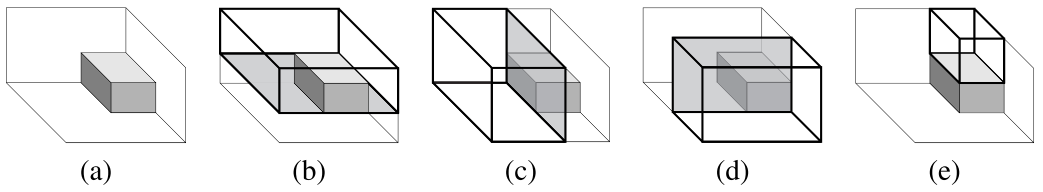


Figure 21: *Eley's space representation. (a) A single item is positioned in the container. (b,c,d) The three residual spaces generated with the insertion of the item. (e) Alternative to the residual space above the item, which will ensure that any new item positioned above, will be place at a stable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

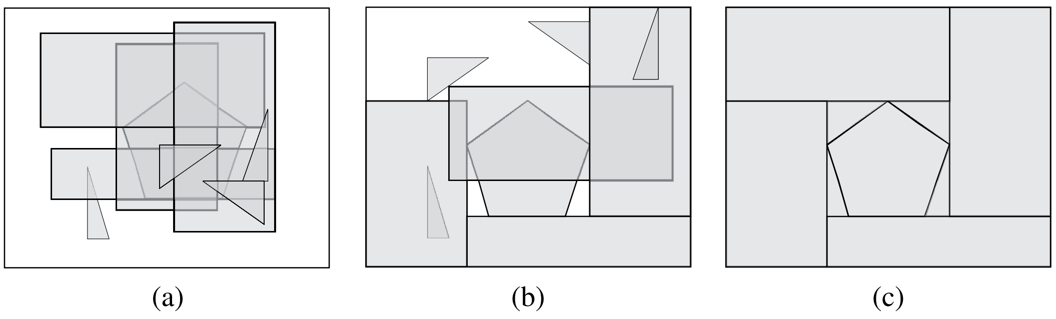


Figure 22: *Relaxed placement. (a) A placement with overlap. (b) Overlap has been reduced significantly by moving items. (c) A feasible placement without overlap.*

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.

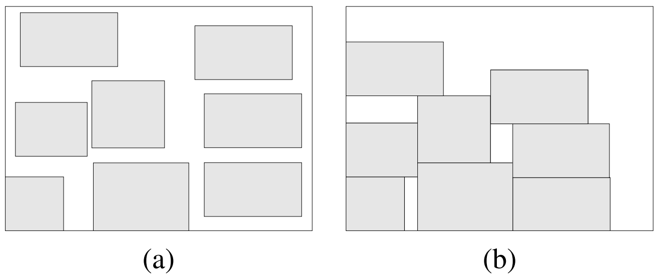


Figure 23: *Normalized placement. (a) A placement of a set of rectangles. (b) Normalized placement, where non of the rectangles can be moved left or down without causing overlap.*

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

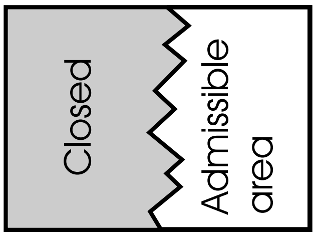


Figure 24: *The notion of an envelope. As the placement is constructed a part of the placement area is inaccessible (closed) and the admissible area for a shape is the remaining area of the placement.*

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

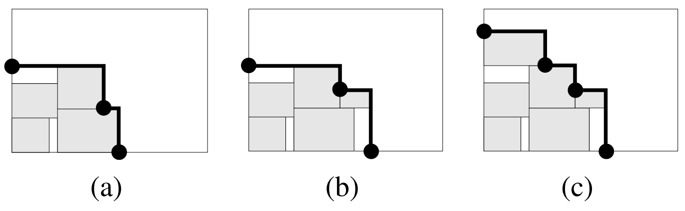


Figure 25: *Rectangular envelope. The shape of an envelope as a placement is constructed by sadding one rectangle at a time. Rectangles are placed in normalized fashion and may not be placed lower or left of the envelope which is indicated by thick lines. Circles indicate feasible positions of the lower-left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

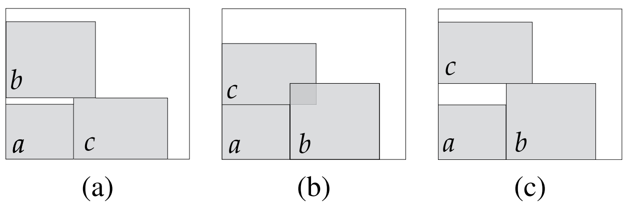


Figure 26: *(a) A placement of rectangles a,b,c. (b) Overlap will appear if b and c are exchanged directly. (c) No overlap will appear if the rectangles are exchanged in the sequence pair representation and then positioned using a decoding algorithm (see text).*

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 $i = 1, \dots, d$ – 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 $E_1, \dots, E_d$ for the d graphs as a *packing class* if it satisfies the following properties:

- P1: The graphs $G_i = (V, E_i)$ for $i = 1, \dots, d$ are interval graphs.
- P2: Each stable set of S of G_i is x_i-feasible for $i = 1, \dots, d$. 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 $i = 1, \dots, d$. 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 $E_1, \dots, E_d$ 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.

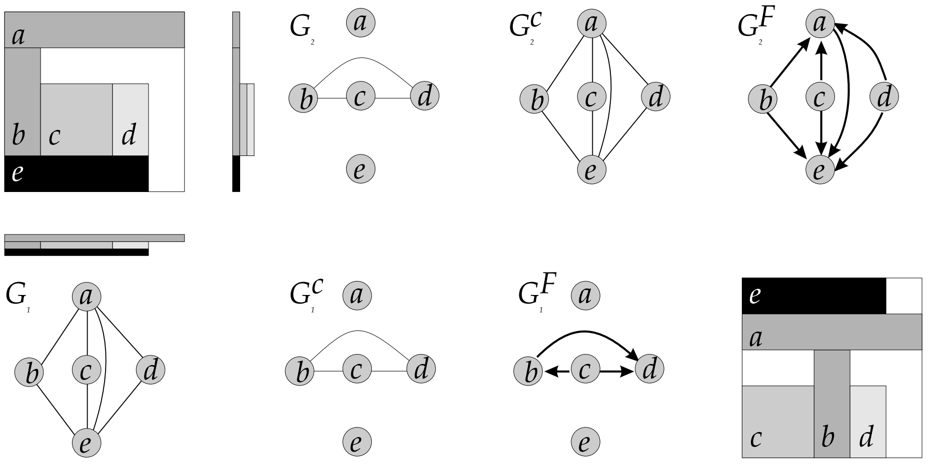


Figure 27: *Packing classes. A placement of rectangles in the upper-right corner is used to generate the interval graphs G_1 and G_2. The edges of G_1 and G_2 constitute the packing class that the placement is part of. The edges of the complementary graphs G_1^C and G_2^C are given an orientation to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

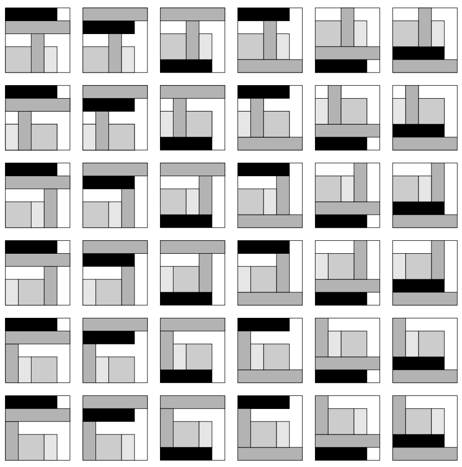


Figure 28: *The 36 placements which arise from the packing class represented by the edges of G_1 and G_2 from figure 27.*

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 $u : [0,1] \to [0,1]$ 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.

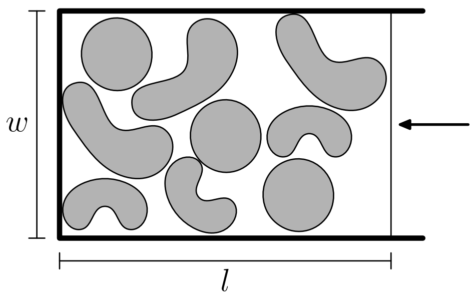


Figure 1: The Strip Nesting Problem. Place a number of stencils on a strip with width w such that no two stencils overlap and the length l of the strip is minimized.

- *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.

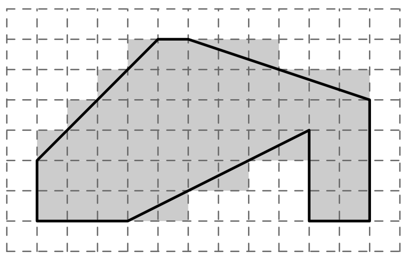


Figure 2: The raster model requires all stencils to be defined by a set of grid squares. The drawing above is an example of a polygon and its equivalent in a raster model.

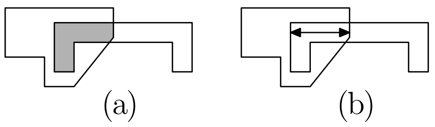


Figure 3: The degree of overlap can be measured in various ways. Here are two examples: (a) The precise area of the overlap. (b) The horizontal intersection depth.

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

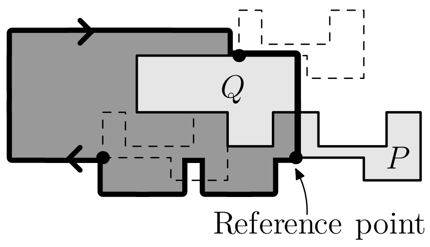


Figure 4: Example of the *No-Fit-Polygon* (thick border) of stencil *P* in relation to stencil *Q*. The reference point of *P* is not allowed inside the NFP if overlap is to be avoided.

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.

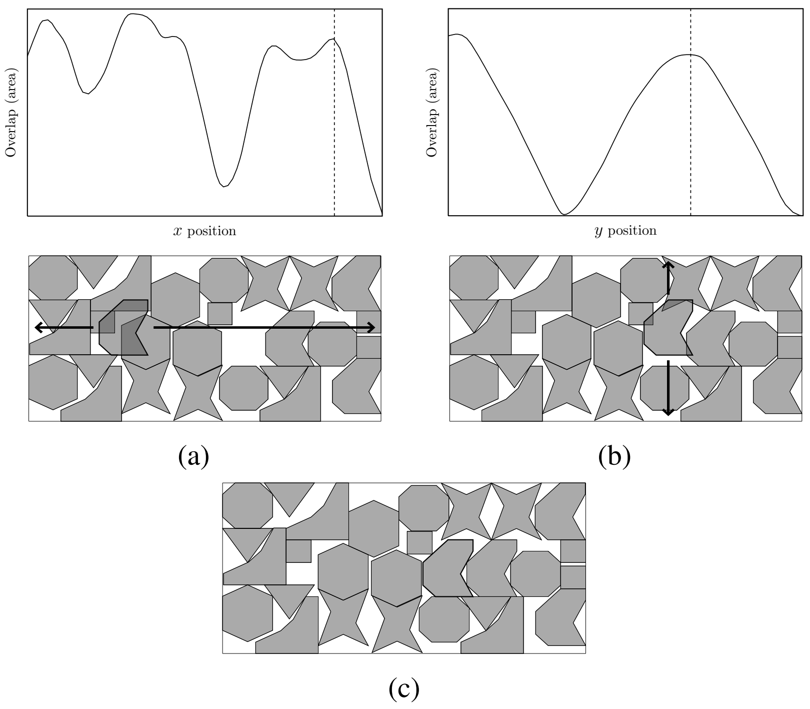


Figure 5: Example of a local search. (a) One polygon overlaps with several other polygons and is selected for optimization. In the top row we have drawn the amount of overlap as a function of the leftmost *x*-coordinate of the polygon. The positions beyond the dashed line are illegal since the polygon 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:

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

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 $\mathcal{S}$;
Generate initial placement $p$;
foreach pair of polygons $s_i, s_j \in \mathcal{S}$ do
    Set $\phi_{ij} = 0$;
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 $s_i, s_j \in \mathcal{S}$ do
        Compute $\mu_{ij}(p)$;
    foreach pair of polygons $s_i, s_j \in \mathcal{S}$ such that $\mu_{ij}$ is maximal do
        Set $\phi_{ij} = \phi_{ij} + 1$;
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).

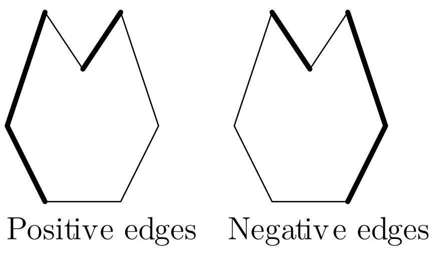


Figure 6: Positive and negative edges of a polygon according to Definition 6.

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*

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

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.

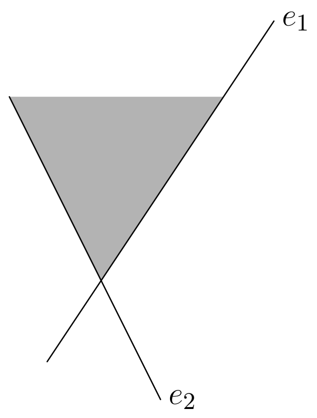


Figure 7: The edge region $R(e_1,e_2)$ of two edges e_1 and e_2 (see Definition 8).

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 $\mathbf{C}$ 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).

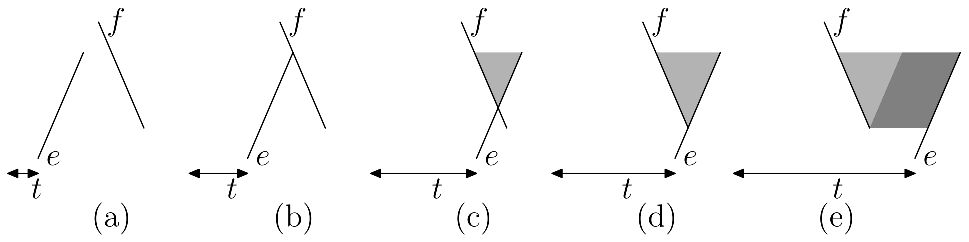


Figure 8: The edge region $R(e,f)$ of two edges as e is translated t units from left to right. (a) e is completely left of f. (b) e adjoins f. (c) e crosses f. (d) e and f are adjoined again. (e) e is to the right of f. Notice that $R(e,f)$ is either $\emptyset$, a triangle or a triangle combined with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

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

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 $\mathcal{S}$ of polygons and a polygon $P \notin \mathcal{S}$;
foreach edge $e$ from polygons $\mathcal{S} \setminus \{P\}$ do
    foreach edge $f$ from $P$ do
        Create breakpoints for edge pair $(e, f)$;
Let $\mathbf{B}$ = breakpoints sorted;
Define area-function $\alpha(t) = \tilde{A}t^2 + \tilde{B}t + \tilde{C}$;
Set $\tilde{A} = \tilde{B} = \tilde{C} = 0$;
foreach breakpoint $b \in \mathbf{B}$ do
    Modify $\alpha(t)$ by changing $\tilde{A}$, $\tilde{B}$ and $\tilde{C}$;
    Look for minimum on the next interval of $\alpha(t)$;
return $t$ with smallest $\alpha(t)$
```

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.

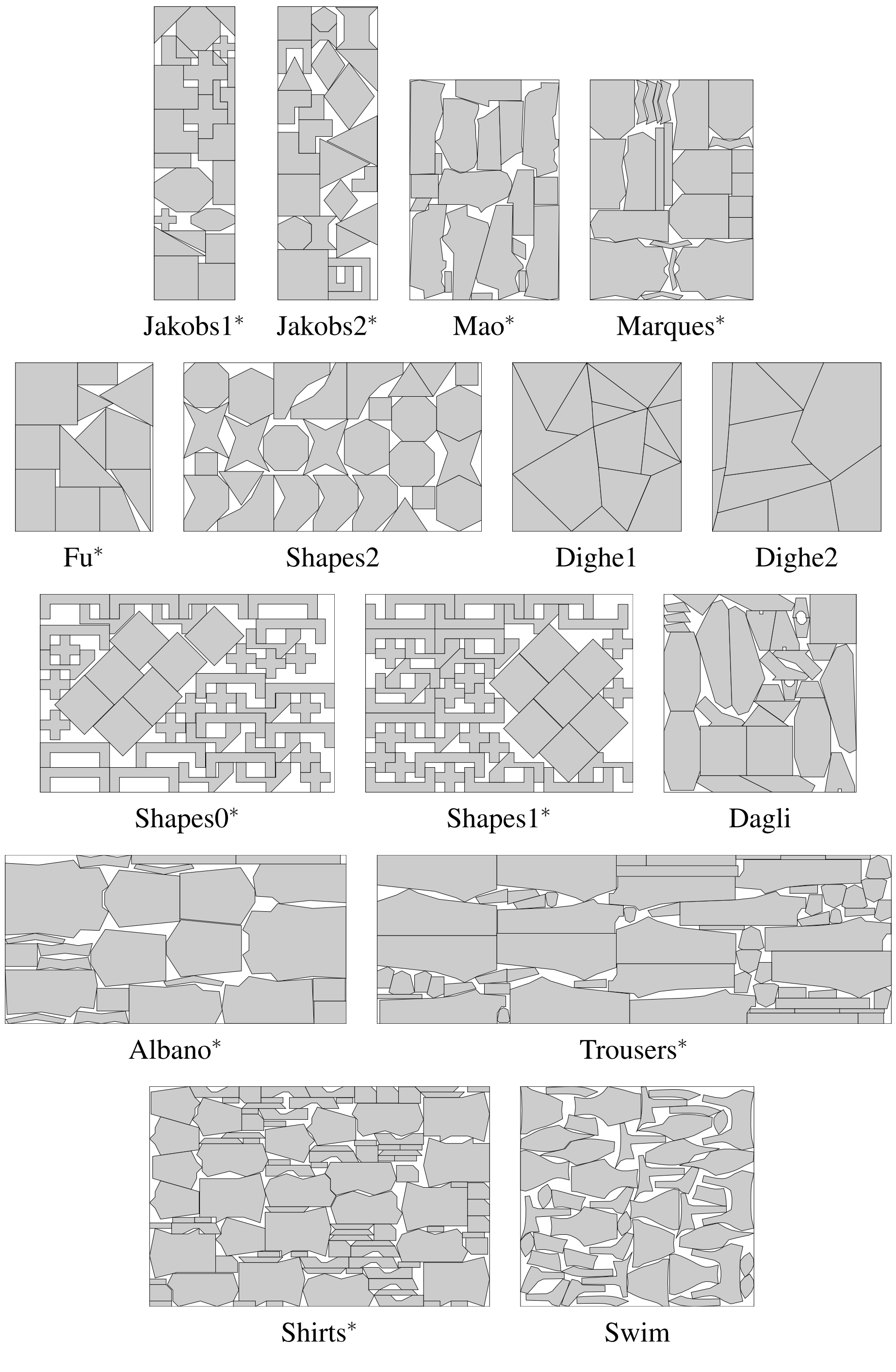


Figure 9: The best solutions found by 2DNEST easily comparable with the ones shown in Gomes and Oliveira [20].

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

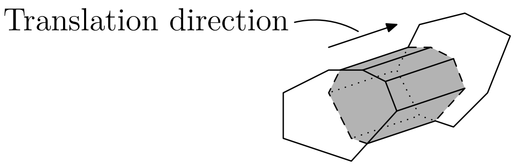


Figure 10: An illustration of the face region between two faces. The faces are not necessarily parallel, but the sides of the face region are parallel with the translation direction. The face region would be more complicated if the two faces were intersecting.

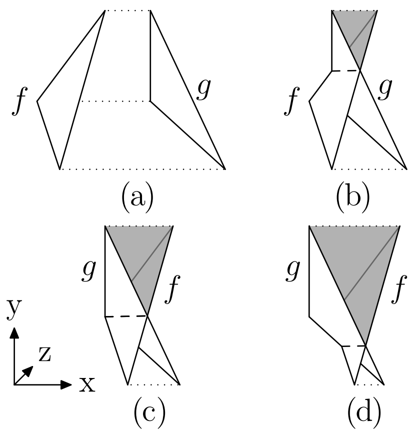


Figure 11: Translation of a triangle f through another triangle g along the x-axis, where the triangles have the same projection onto the yz-plane. The face region $\mathbf{R}(f,g)$ changes shape each time two corner points meet.

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.

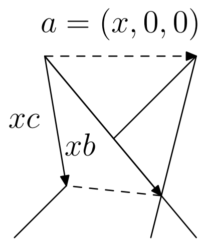


Figure 12: The volume of the above tetrahedron can be calculated from the three vectors **a**, **b** and **c**. In our case **b** and **c** are linearly dependent on x which is the length of **a** (and the translation distance since the tetrahedron started growing).

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:

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

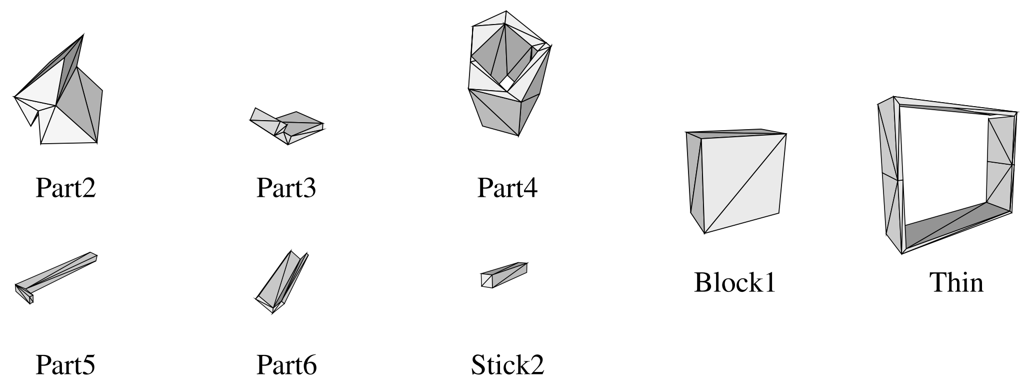


Figure 13: The Ikonen data set.

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

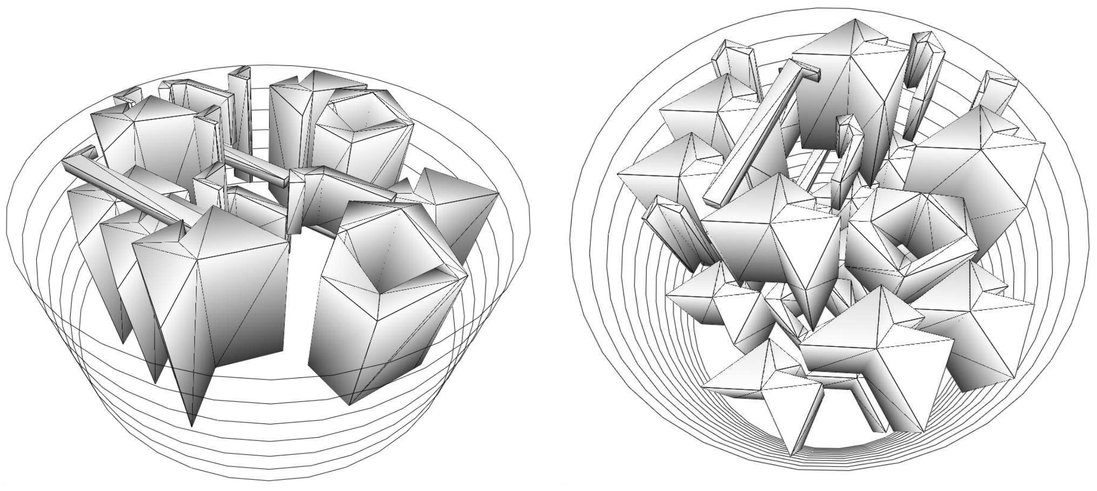


Figure 14: The above illustrations contain twice as many objects as originally intended in Ikonens Case 1 and 2. They only took a few seconds to find.

Pack 10 objects into a cylinder of radius 3.4 and height 3.0. The 10 objects were chosen as follows: 3 × Part2, 1 Part4 and 2 × 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			
			Old	Current			Old	Current		
	Size	Deg.	2DNEST	NEST2D	ILSQN	SAHA	2DNEST	NEST2D	ILSQN	SAHA
Albano	24	180°	86.96	$\underline{87.50}$	87.14	84.70	87.44	87.90	$\underline{88.16}$	87.43
Dagli	30	180°	85.31	$\underline{86.17}$	85.80	85.38	85.98	$\underline{87.51}$	87.40	87.15
Dighe1	16		93.93	$\underline{99.99}$	90.49	82.13	99.86	$\underline{100.00}$	99.89	$\underline{100.00}$
Dighe2	10		93.11	$\underline{99.99}$	84.21	84.17	99.95	$\underline{100.00}$	99.99	$\underline{100.00}$
Fu	12	90°	90.93	91.03	87.57	87.17	91.84	$\underline{91.94}$	90.67	90.96
Jakobs1	25	90°	88.90	$\underline{89.05}$	84.78	75.79	89.07	$\underline{89.09}$	86.89	78.89
Jakobs2	25	90°	80.28	$\underline{80.71}$	80.50	74.66	80.41	82.44	$\underline{82.51}$	77.28
Mao	20	90°	82.67	$\underline{83.08}$	81.31	80.72	$\underline{85.15}$	84.23	83.44	82.54
Marques	24	90°	88.73	$\underline{89.12}$	86.81	86.88	89.17	$\underline{89.85}$	89.03	88.14
Shapes0	43		65.42	66.07	66.49	63.20	67.09	66.74	$\underline{68.44}$	66.50
Shapes1	43	180°	71.74	72.35	72.83	68.63	73.84	73.83	$\underline{73.84}$	71.25
Shapes2	28	180°	79.89	80.69	$\underline{81.72}$	81.41	81.21	82.32	$\underline{84.25}$	83.60
Shirts	99	180°	85.73	86.61	$\underline{88.12}$	85.67	86.33	87.35	$\underline{88.78}$	86.79
Swim	48	180°	70.27	72.00	$\underline{74.62}$	72.28	71.53	73.04	$\underline{75.29}$	74.37
Trousers	64	180°	89.29	$\underline{89.64}$	88.69	89.02	89.84	$\underline{90.04}$	89.79	89.96
			83.54	$\underline{84.93}$	82.74	80.12	85.25	85.75	$\underline{85.89}$	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`.

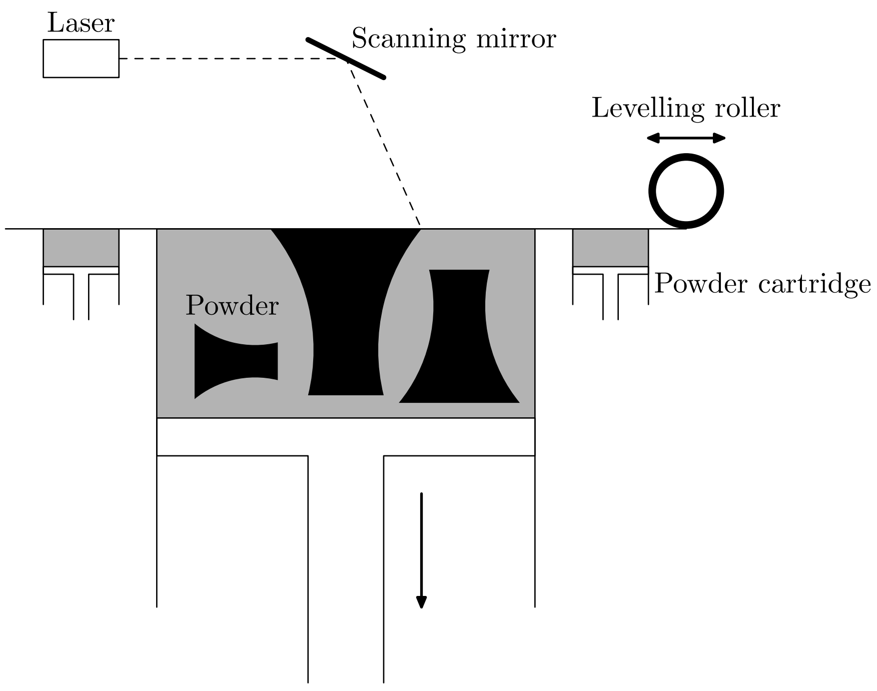


Figure 1: An illustration of a typical machine for rapid prototyping. The powder is added one layer at a time and the laser is used to sinter what should be solidified to produce the desired objects.

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*

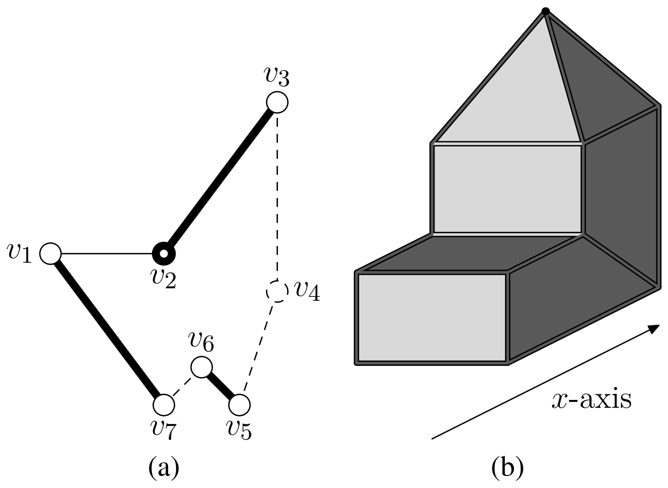


Figure 2: (a) A polygon with three positive (thick) edges, (v_1, v_7), (v_2, v_3), and (v_5, v_6), three negative (dashed) edges, (v_3, v_4), (v_4, v_5), and (v_6, v_7), and one neutral (thin) edge, (v_1, v_2). Also note that the end-points v_1, v_3, v_5, v_6, and v_7 are neutral (thin), v_2 is positive (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 $p \notin \operatorname{int}(P)$ 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 $p \in \operatorname{int}(P)$ 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.

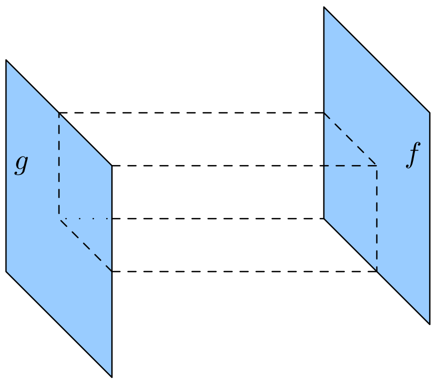


Figure 3: A simple example of the inter-facet region $R(f,g)$ of two vertical faces f and g in $\mathbb{R}^3$. The dashed lines delimit the region. Note that the inter-facet region $R(g,f)$ is empty.

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

$$\mathbf{C}(f,g,p') = \begin{cases} 1 & \text{if } \exists t_1,t_2 \in \mathbb{R} \,:\, t_2 < 0 < t_1,\ p' + t_2 e_1 \in g, \text{ and } p' + t_1 e_1 \in f \\ 0 & \text{otherwise.} \end{cases}$$

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.

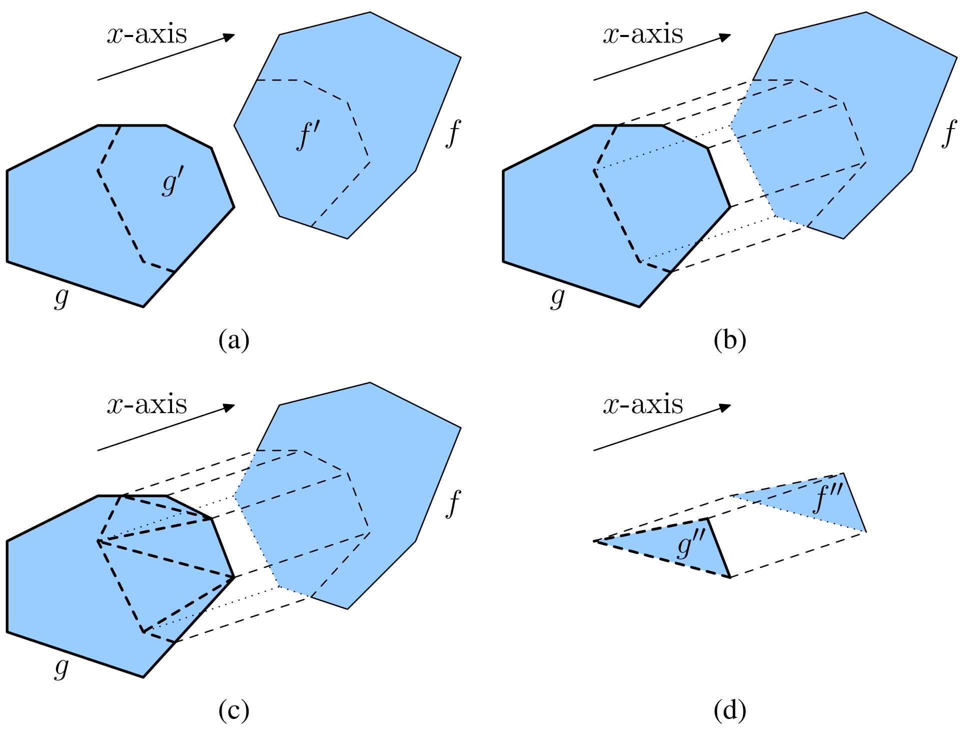


Figure 4: An example of the inter-facet region $R(f,g)$ between two faces, f and g, in three dimensions, where g is in front of f in the x-axis direction. In general the two facets are not necessarily in parallel planes and they can also intersect. (a) The dashed lines indicate the boundary of the projections 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.

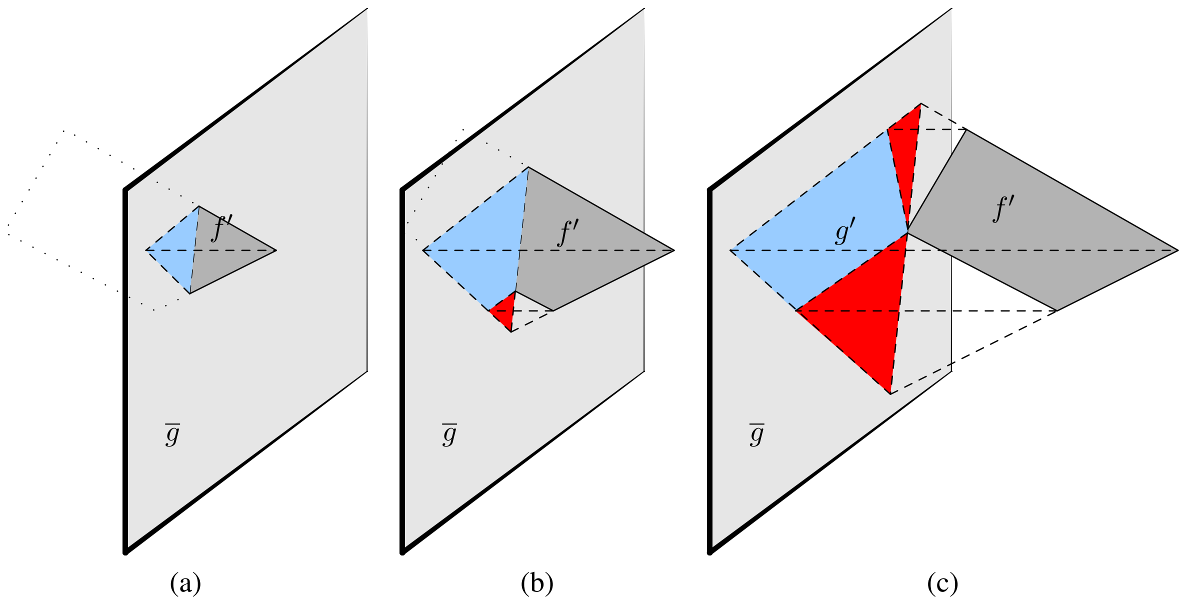


Figure 5: An illustration of the growing tetrahedron volumes needed when a face f' passes through a face g, or equivalently its plane $\overline{g}$, in order to calculate the volume of $R(f',g')$. (a) The initial growing tetrahedron after the first breakpoint. (b) After the second breakpoint, at growing 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:

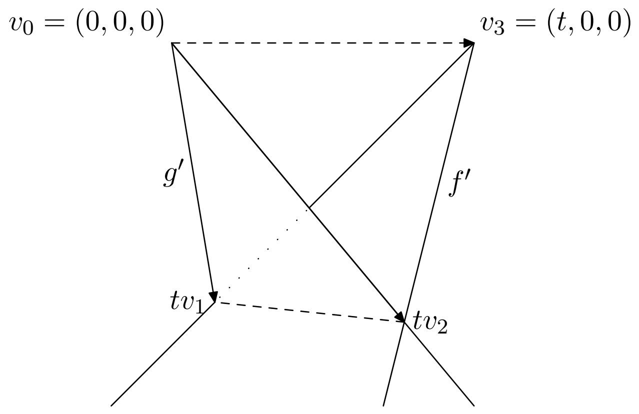


Figure 6: In 3D, it is necessary to calculate the volume function for a growing tetrahedron based on two points, v_1 and v_2, and an overlap distance t. The coordinates of the points change linearly with respect to a change of the distance t. In d dimensions $d-1$ neighboring points are used.

$$V_0(f(t),g) = \frac{1}{d!}|\det([tv_1,...,tv_d])|.$$

This is illustrated in Figure 6.

Since $v_d = (1,0,\dots,0)$, 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:

$$\begin{array}{rrcl} 2\text{D}: & \frac{1}{2}|\det(v_2')t^2| & = & \frac{1}{2}|v_2^y t^2| \\ 3\text{D}: & \frac{1}{6}|\det([v_2'v_3'])t^3| & = & \frac{1}{6}|(v_2^y v_3^z - v_2^z v_3^y)t^3| \end{array}$$

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

$$V(f(t),g) = V(\Delta(p_1,t)) - \sum_{i=2}^{k} V(\Delta(p_i,t)) \text{ where } k = \max_{1 \le i \le n} \{i | t_i < t\}. \quad (3)$$

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 $t_1 \le t_2 \le \ldots \le t_n$. 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.

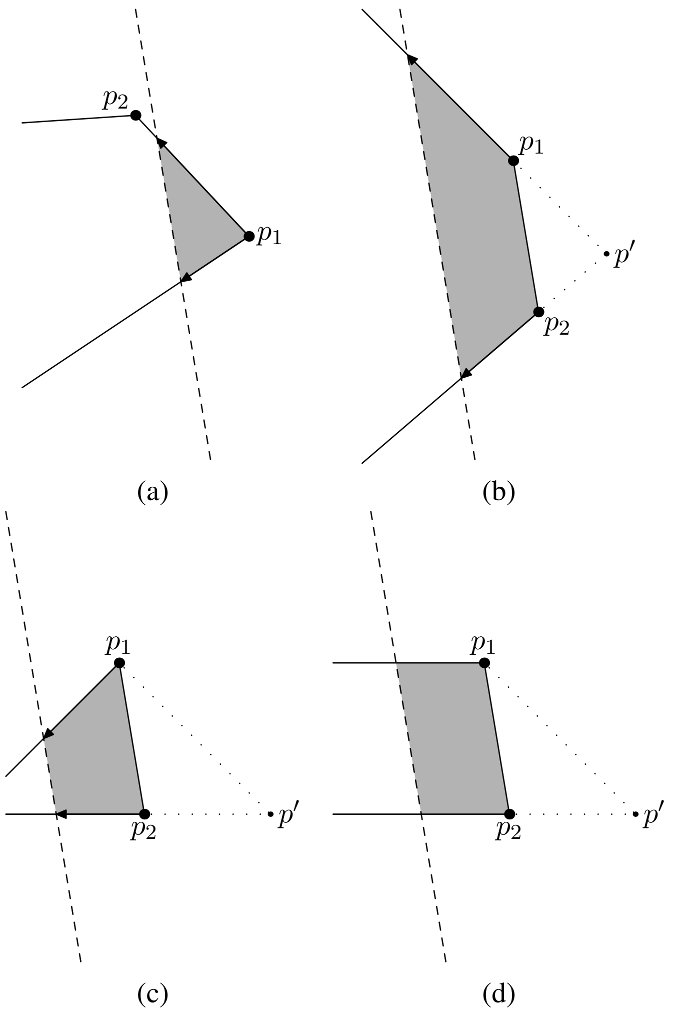


Figure 7: Various special cases can occur when the two first breakpoint distances are equal. Here it is illustrated in 2D using a dashed line to illustrate the intersection line of the two faces during the translation. (a) The standard case in which the first breakpoint distance is unique. (b) The two 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.

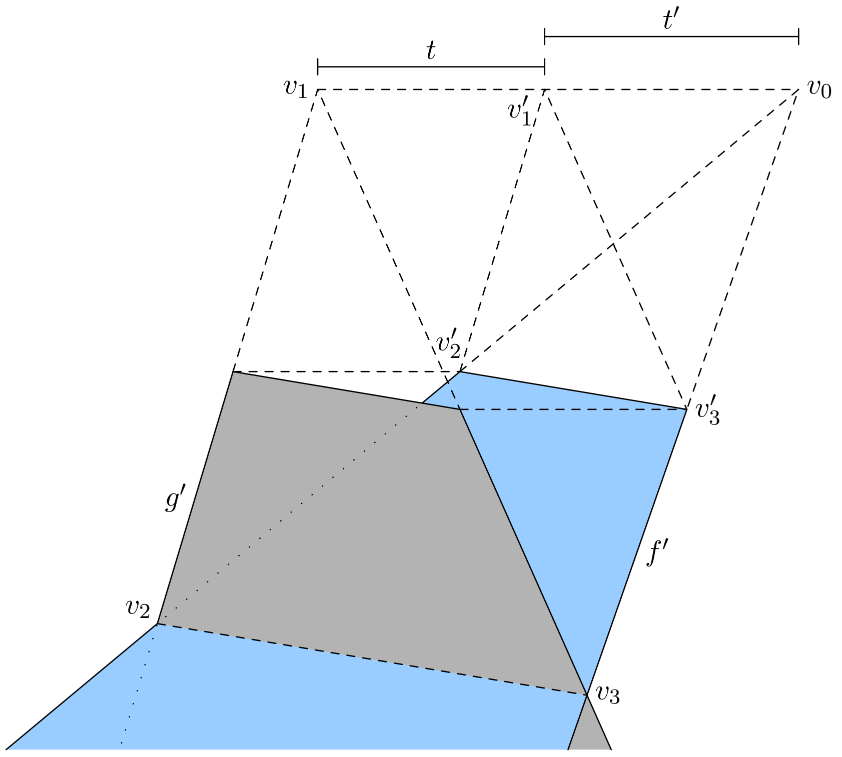


Figure 8: To calculate the volume between the faces f' and g', one can base the calculations on the extended faces illustrated with dashed lines. The volume is a growing tetrahedron (v_0, v_1, v_2, v_3) subtracted by a constant volume (the tetrahedron (v_0, v_1', v_2', v_3')) and subtracted by a linearly 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