Datalogisk Institut
KØBENHAVNS UNIVERSITET

Nesting Problems and Steiner Tree Problems

Benny Kjær Nielsen

Technical Report no. 08-06
ISSN: 0107-8283

Dept. of Computer Science
University of Copenhagen • Universitetsparken 1
DK-2100 Copenhagen • Denmark

Nesting Problems and Steiner Tree Problems

Benny Kjær Nielsen
benny@diku.dk
DIKU, University of Copenhagen

November, 2007

Preface

A large part of research is a search through a maze built of existing literature. Numerous journal papers, conference proceedings, technical reports and books are read in order to figure out what has been done and what has (probably) not been done within a certain research field. Occasionally one thinks of a new idea and a new search begins in the relevant part of the maze to answer the question: Did someone have the same or a similar idea? Working on the idea might reveal that it is a bad idea — a blind alley in the maze — or it may just seem like the path goes on forever and eventually one decides to go back.

At some point, one reaches a level of understanding of the research field which makes it easier to separate good ideas from bad ideas. In a sense, one rises above the maze. The maze is still there, but it is much easier to find a successful path. This thesis contains two major research fields; nesting problems and Steiner tree problems. In the analogy of the research maze, I managed to rise above the maze for some parts of these research fields and I stayed firmly on the ground for other parts. Many blind or seemingly endless alleys were attempted. One of these seemingly endless (and maybe blind) alleys deserves special mentioning. In cooperation with Marco E. Lübbecke[1], Martin Zachariasen and Pawel Winter, it was attempted to solve the "rectilinear Steiner tree problem with obstacles" using integer programming and column generation. A major implementation was done by Marco and I which included an increasing number of advanced strategies in order to improve the results. We never reached the end of the alley, but Marco deserves my thanks for his great efforts, especially since I never had the pleasure of co-authoring any other papers with him.

Also, a special thank you to each of my co-authors in this thesis. They are, in alphabetical order, Marcus Brazil, Jens Egeblad, Stinus Lindgreen, Allan Odgaard, Doreen A. Thomas, Pawel Winter, Christian Wulff-Nilsen, and Martin Zachariasen. In particular, I am very grateful for the guidance of my supervisor, Martin, and for the very inspiring lunch conversations with Marcus in the Lygon Street cafés in Melbourne — especially since this was in a time where every alley seemed to be blind.

Benny Kjær Nielsen
November, 2007

[1] Technische Universität Berlin, Institut für Mathematik

Contents

Introduction

Two major subjects are treated in this thesis; *nesting* problems and *Steiner tree* problems. In short, nesting problems are packing problems in which the utilization of some material (or space) is to be maximized, and Steiner tree problems are tree construction problems in which the length of a tree connecting a set of points/vertices is to be minimized. These subjects are introduced separately in this thesis since they have little in common, but both nesting problems and Steiner tree problems are, in general, $\mathcal{NP}$-hard. Exceptions do exist, e.g., the design of microprocessors involves packing submodules while minimizing wiring length. Each of the introductions is followed by four papers on the subject. The following is a short overview of the two subjects and the related papers.

The emphasis in this thesis is on efficient algorithms and problem specific heuristics which can help solve hard optimization problems. The nesting papers are mainly concerned with variations of a translation algorithm which has proven to be a very useful geometric tool for solving nesting problems. Given some placement of the shapes (represented as polygons in 2D), the translation algorithm solves the problem of moving a shape in an axis-aligned direction to a position which minimizes its overlap with all other shapes.

The first paper by Egeblad et al. [4][A] introduces the translation algorithm and shows that state-of-the-art results can be obtained when it is applied to two-dimensional packing problems with irregular shapes. They also generalize the translation algorithm to efficiently handle three-dimensional packing problems, but they give no proof of its correctness. Nielsen [5][B] showed that the translation algorithm can be modified to handle packing problems in which the solution is going to be repeated (like a tiling), and Egeblad et al. [3][C] formally proved the correctness of the translation algorithm in three dimensions while also generalizing it to an arbitrary number of dimensions. Finally, Nielsen et al. [6][D] described how the translation algorithm can be implemented efficiently for translations in non-axis-aligned directions. This is then utilized for a novel approach for solving nesting problems in which free orientation of the shapes is allowed. The introduction to nesting problems contains additional material on the subject of measures of overlap. An extended version of this material is under preparation for a future nesting related paper. The introduction also discusses other geometric techniques used for solving nesting problems in the existing literature.

In all of the nesting papers, the same meta-heuristic technique is applied (*guided local search*). This works well, but it does not mean that it is the only possible choice. It emphasizes the fact that all of the papers are concerned with algorithms and heuristics which are closely related to the geometric challenges of the problem. The subject of optimizing the parameters of a sophisticated meta-heuristic has not been addressed in any of the papers. This is partly motivated by the belief that the choice of meta-heuristic is much less important than the development of good problem specific algorithms and heuristics.

The Steiner tree related papers treat more diverse problems. The first paper by Nielsen et al. [7][E] is the only paper in this thesis which does not contain any computational experiments. The paper is concerned with the location of so-called *Steiner points* in uniform orientation metrics. In short, the problem is to construct a minimum length tree connecting a set of points in the plane with the restriction that all edges adhere to a finite set of uniformly distributed orientations. The results in this and other papers were later used to implement an exact algorithm for solving very large problem instances, see Nielsen et al. [8]. The length of such minimum length Steiner trees change when the given set of points is rotated. Finding

the optimal orientation of a given set of points is the subject of the paper by Brazil et al. [2]^{F}. The motivation of studying Steiner trees in uniform orientation metrics is an application in VLSI design, i.e., the design of integrated circuits for which the routing wires are restricted to some small set of orientations.

Another application of Steiner trees can be found in the area of computational biology. Given a set of species, the problem is to construct a tree describing their evolutionary history. This is most often described as a phylogenetic tree. It is not necessarily a Steiner tree problem, because the exact problem definition depends on the model of evolution assumed. Nevertheless, it can be stated as a Steiner tree problem. Based on DNA sequences from the species, Nielsen et al. [9]^{G} assume that the problem can be modeled as the problem of finding a Steiner tree in a very large graph. The nodes of the graph represent all possible DNA sequences up to some fixed length and the edges represent possible mutations of DNA sequences. The problem is then to find the shortest tree connecting the nodes in the graph which represent the species. Brazil et al. [1]^{H} approach the problem in a completely different manner, but they also end up with a Steiner tree problem. Based on some measure of pairwise distances between the species, a transformation is done to represent the species by points in a high-dimensional Euclidean space. The problem is then to find a minimum length Steiner tree connecting all of the points.

Eight papers are included in the thesis; four nesting related papers and four Steiner tree related papers. Six of these papers are either published or in press and two papers have been submitted recently. The papers have been adapted to follow a uniform style for this thesis. This includes changes to the size of figures, the layout of tables and the style of the references. Some obvious textual corrections have also been done such as corrections of spelling errors. Note that whenever a paper included in this thesis is cited, the index number of the citation is followed by a superscript capital letter, e.g., [3]^{C}. This can be used to identify the paper in the table of contents.

References

[1] M. Brazil, B. K. Nielsen, D. Thomas, P. Winter, C. Wulff-Nilsen, and M. Zachariasen. A novel approach to phylogenetic trees: d-dimensional geometric Steiner trees. *Networks*. In Press.

[2] M. Brazil, B. K. Nielsen, P. Winter, and M. Zachariasen. Rotationally optimal spanning and Steiner trees in uniform orientation metrics. *Computational Geometry*, 29(3):251–263, 2004.

[3] J. Egeblad, B. K. Nielsen, and M. Brazil. Translational packing of arbitrary polytopes. Submitted.

[4] 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.

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

[6] B. K. Nielsen, M. Brazil, and M. Zachariasen. Using directed local translations to solve the nesting problem with free orientation of shapes. Submitted.

[7] B. K. Nielsen, P. Winter, and M. Zachariasen. On the location of Steiner points in uniformly-oriented Steiner trees. *Information Processing Letters*, 83(5):237–241, 2002.

[8] B. K. Nielsen, P. Winter, and M. Zachariasen. An exact algorithm for the uniformly-oriented Steiner tree problem. *Algorithms - ESA 2002: 10th Annual European Symposium, Rome, Italy, September 17-21*, pages 237–266, 2002.

[9] B. K. Nielsen, S. Lindgreen, P. Winter, and M. Zachariasen. Deferred path heuristic for phylogenetic trees revisited. In M.-F. Sagot and K. S. Guimarães, editors, *CompBioNets 2005: Algorithmics and Computational Methods for Biochemical and Evolutionary Networks*, volume 5 of *Texts in Algorithmics*, pages 75–92, King's College London, Strand, London, 2005. College Publications.

Nesting Problems

1 Introduction

1.1 The problem

Cutting and packing problems are classic subjects in computer science research — literally thousands of papers [31] have been written on this subject. The problems range from the relatively simple one-dimensional *cutting stock problem* to high-dimensional packing problems involving arbitrary shapes. The word "simple" should be taken with a grain of salt since even the one-dimensional cutting stock problem is $\mathcal{NP}$-hard. Applications of the cutting stock problem can be found in, e.g., the paper industry. In this case, stock items could be rolls of paper with identical lengths. Given a number of orders of various lengths, the problem is then to decide how to cut pieces from the stock items matching the lengths of the orders. Most often the optimization goal is to minimize the number of stock items needed to satisfy all orders, i.e., the goal is to minimize the waste (or maximize the utilization) of the stock items. A small example is given in Figure 1a.

There are numerous variations of the cutting stock problem, e.g., the lengths of the stock items might not be identical or some maximum of different cutting patterns is desired. These variations are relatively simple when compared to generalizations of the cutting stock problem to higher dimensions. In higher dimensions, geometry is often the most important part of the problem description. Simple problems only involve rectangular items to be cut from a rectangular material. Most often the items are then also required to being oriented such that their sides are parallel with the borders of the material (see Figure 1b). Geometrically, problems with non-rectangular shaped items are much harder to handle. The word "irregular" is often used to described such shapes although this word does not really mean non-rectangular. A regular polygon is a polygon with congruent angles and sides with identical lengths; this definition does not even include rectangles (except squares). Problems which allow free orientation of shapes are much harder to handle even if the shapes are rectangular.

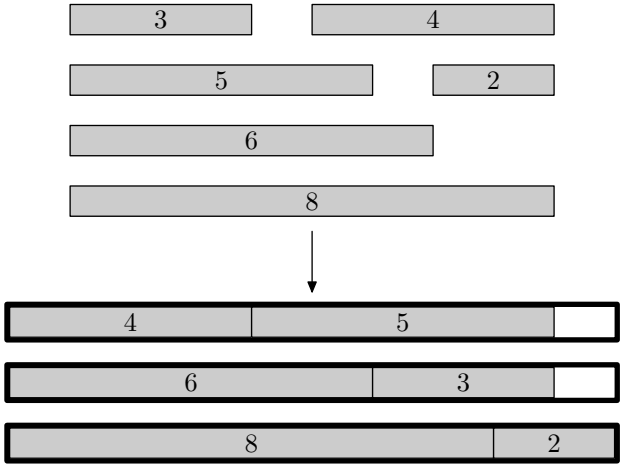
Applications in two dimensions include cutting problems in the textile, animal hide, glass and metal industries. In three dimensions, applications include packing problems related to container shipping and truck loading. Note that the terminology suddenly changed from cutting to packing. This is because three dimensional problems most often involve solid objects which are to be packed within the boundaries of some container. Although the wording changed, the problem is the same — instead of maximizing the utilization of some material, one maximizes the utilization of space. That said, three dimensional problems in the field of *rapid prototyping* [33] does actually involve the utilization of a material. Two-dimensional problems with irregular shapes are often referred to as *nesting problems* and this is the term used in this introduction. This emphasizes the fact that we are dealing with irregular shapes.

A very general definition of a nesting problem can be stated as follows.

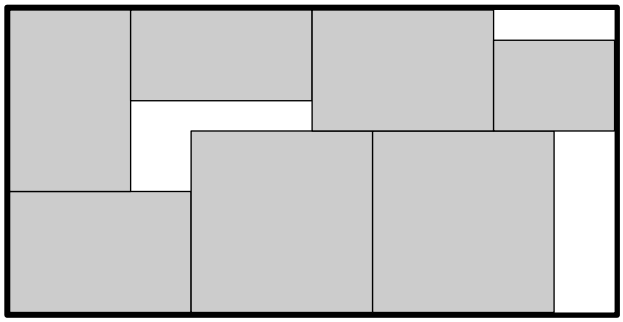
Nesting Decision Problem. *Given a set of shapes $\mathcal{S}$ and a container C in d-dimensional space, determine whether the shapes can be placed (possibly including orientation) such that there is no overlap between the shapes and they are all inside the container.*

The container can be thought of as a shape which is inside-out.

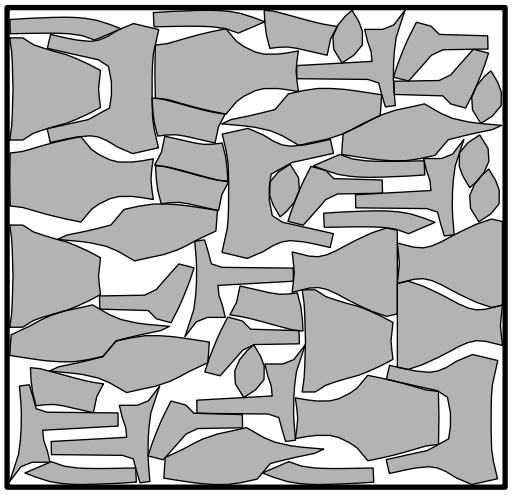
We also define the following optimization problem.



(a)



(b)



(c)

Figure 1: (a) A one-dimensional cutting stock problem and a solution. (b) A solution to a two-dimensional cutting stock problem with rectangular items and a single piece of rectangular material. (c) A solution to a two-dimensional cutting stock problem with non-rectangular shapes (a nesting problem).

Strip Nesting Problem. *Given a set of shapes $\mathcal{S}$ and a rectangular axis-aligned parallelepiped C in d-dimensional space, determine the smallest length of C for which the shapes can be placed (possibly including orientation) such that there is no overlap between the shapes and they are all inside the parallelepiped.*

Here, the length of the container is its size with respect to the x-coordinate. The other coordinates are assumed to be fixed. The two-dimensional variant of this problem is the one that is most often addressed in the literature when the subject is on nesting irregular shapes. This is partly due to the fact that this problem matches the problem most often faced in the textile industry where the "infinite" strip is a roll of fabric. The glass industry and parts of the metal industry are closer to a bin-packing variant of the problem, but this has received little attention in the existing literature. The one-dimensional variant of the strip nesting problem is trivially solved. The same is not true for the nesting decision problem since we have not yet specified any restrictions on the shape of a container. It could be several disconnected parts and thus be equivalent to the one-dimensional cutting stock problem.

The quality of a given non-overlapping placement is measured by the *utilization* of the volume (area in 2D) of the container, i.e., if $\nu(S)$ denotes the volume of a shape S and this is generalized for sets such that $\nu(\mathcal{S}) = \sum_{S \in \mathcal{S}} \nu(S)$ then a non-overlapping placement has a utilization (of the container) of $\nu(\mathcal{S})/\nu(C)$.

In this introduction, the focus is primarily on two-dimensional nesting problems. Nevertheless, most of the discussions in this introduction and the solution methods presented in the following four papers can be generalized to three and even more dimensions. This is described in detail in the paper by Egeblad et al. [13][C] and in a subsection of the paper by Nielsen et al. [27][D].

1.2 The shapes

In two dimensions, a shape can be described as a set of points in the plane ($\mathbb{R}^2$). It is convenient to require that this set of points is an *open* set, i.e., every point in the set must be an *interior* point. Arbitrary shapes can be very difficult to handle and some kind of approximation is often used. Imamichi and Nagamochi [20] discuss an interesting approach in which they approximate two-dimensional shapes by sets of circles and three-dimensional shapes by sets of spheres. A large number of circles or spheres may be necessary, but they should then be easier to handle than the original shapes. This is a quite unique approach to the problem of approximating shapes and most papers in the literature about the nesting problem simply use polygons to approximate shapes (illustrated in Figure 2). This is also the case in this introduction and for the following papers on the nesting problem — with a generalization to higher-dimensional variants of polygons (polytopes) in the paper by Egeblad et al. [13][C].

Some solution methods for the nesting problem can handle more complicated shapes than polygons, for example, Burke et al. [5] can handle polygons with circular arcs and Stoyan et al. [29] construct objects from a set of primary objects which includes circles.

1.3 The solution methods

The definition of the nesting decision problem, as it is presented above, is easy to understand, but it lacked a lot of details. Even when the shapes have been defined more precisely, it is not defined what is meant by "no overlap" or "inside the container". It can be defined formally

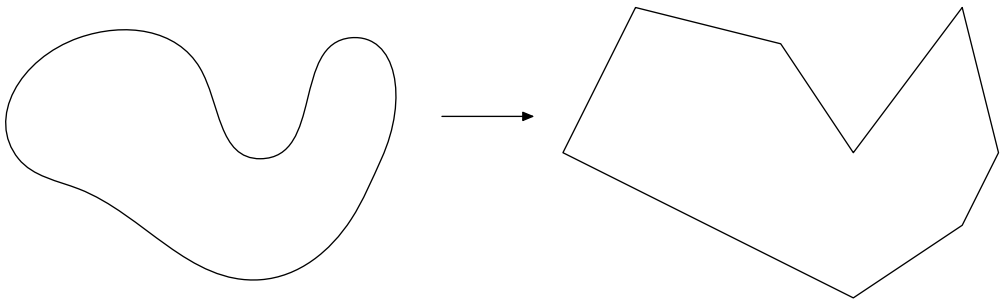


Figure 2: Any shape can be approximated by a polygon. In this case, the approximation is very crude. Using a sufficient number of edges, an approximation can be arbitrarily close to the original shape.

by, e.g., stating that no point must be inside more than one shape (no overlap) and that no point must be inside some shape and outside the container shape (inside the container), but this does not really improve the intuition of the problem. Humans are quite comfortable with the notions of "no overlap" and "inside the container" — even with very complicated shapes because this is a natural part of the world we live in.

Most people handle three-dimensional packing problems on a daily basis. For example, when we try to pack groceries in as few bags as possible or when we try to pack all of the groceries into a refrigerator. The problem also arises when packing as much as possible into a suitcase or when packing suitcases into the trunk of a car.

In general, humans are very good at solving these problems. Intuitively, we rotate items to improve packings and we easily handle restrictions, such as not turning an open milk carton upside down. We consider the consequences of gravity and we can make sure that fragile objects are not put below heavy objects. Faced with difficult packing problems we try different heuristic approaches, e.g., placing similar objects close to each other. We can even squeeze or bend soft objects to make more room for other objects. The list of problem variations, we encounter, is endless, and so is the list of (heuristic) techniques we apply to solve them.

Computer scientists often find inspiration in human problem solving. For nesting problems this is especially interesting, because these problems (in two and three dimensions) have very physical counter parts in the real world. For example, in the textile industry, manual nesting is done by having a set of *stencils* representing the shapes on a large table. The goal is then to find a non-overlapping placement of the stencils with as little waste of fabric as possible. Most humans would probably use one of two techniques (or a combination); placing one stencil at a time without ever introducing overlap or place all stencils randomly with overlap and then move (and swap) them until a non-overlapping solution has been found. In the computer science literature these approaches are represented by the bottom-left placement heuristics [12, 16, 28] and the heuristics which focus on minimizing overlap by moving one shape or swapping two shapes until a non-overlapping solution is found [4, 14, 32]. Other approaches exist though, e.g., solution methods in which all shapes are moved in each iteration [17, 21] — a less likely approach to be taken by a human.

Humans are also very good at handling three dimensional shapes, but they loose the ability to handle intermediate overlapping placements since the physical world does not allow

such a degree of freedom. A computer program is not concerned with the physical world and some heuristics minimizing overlap can actually be generalized to higher dimensions [13]^{C}. In other words, it is hard for a computer program to compete with the packing skills of a human in three dimensions, but interestingly this changes completely when moving beyond three dimensions. Such objects have no obvious representation in the physical world and it is even hard for a human to grasp what, e.g., a four dimensional shape is. Packing boxes in four dimensions is an almost impossible task for a human, while (at least for some solution methods) it is just another coordinate to handle for a computer program.

Still, in two and three dimensions, the nesting problem is a very intuitive and visual problem. It has many applications and numerous commercial nesting applications exist — especially for the two-dimensional case. Unfortunately, benchmarks for these applications do not exist and it is an open question how well the solution methods in the literature compare to the commercial applications. It is known though that some solution methods in the literature, e.g., the one by Heckmann and Lengauer [19], are now used in commercial nesting libraries.

The above is a short introduction to the subject of nesting problems. In the remaining part of this introduction, the focus is mainly on the geometrical challenges faced when trying to solve these problems. In Section 2, various techniques are discussed with references to some of the solution methods using these techniques. All four papers about the nesting problem in this thesis are based on the same geometric technique; a translation algorithm which can move a shape to a position with "less" overlap in a given direction. In all four papers, area (or volume) is used to measure overlap. Other measures are possible, but these are not discussed in any of the papers. This is an important subject for future research and a discussion of overlap measures is therefore given in Section 3, especially in relation to the translation algorithm.

Heuristic and meta-heuristic techniques used for the nesting problem which are not related to the geometric aspect of the problem are not discussed in this introduction. For a discussion of complete solution methods for the two-dimensional nesting problem, we refer to the introductions of the papers by Egeblad et al. [14]^{A}, Burke et al. [5], and Gomes and Oliveira [17]. For three-dimensional nesting problems, we refer to Egeblad et al. [13]^{C} and Stoyan et al. [30].

The computational experiments by Egeblad et al. [14]^{A} have been repeated with the latest version of our nesting program Nest2D. The new results are compared with the old results in Section 4. They are also compared with some of the latest results from the literature.

2 Geometric Techniques

One of the greatest challenges for nesting solution methods is to handle the geometric aspects efficiently. This can, e.g., be in the form of efficient methods for determining if and how much two shapes overlap, but the specific needs depend on the heuristic approach to the problem. In this section we present the relatively few techniques used in the existing literature to handle the problems related to the geometry of nesting problems.

2.1 No-Fit-Polygons

Most solution methods can be put into one of two groups. Either it is a *legal placement method* or it is a *relaxed placement method* [14]^{A}. The end result of any solution method should be a non-overlapping placement of the shapes. The difference between the two groups

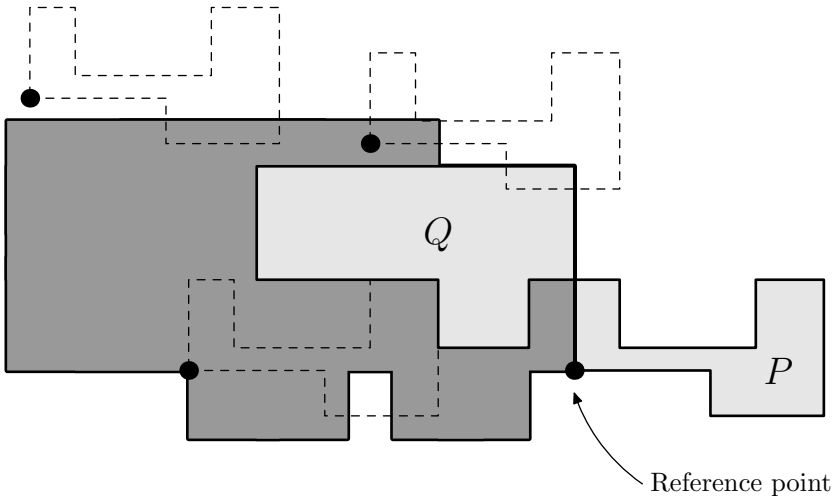


Figure 3: Two polygons, P and Q, and their corresponding no-fit-polygon (dark area). Note the reference point of P in its lower left corner. Copies of P are drawn with dashed lines at various positions. When the reference point is inside the no-fit-polygon, the polygons overlap and when it is outside the no-fit-polygon, the polygons do not overlap. Given a no-fit-polygon, a test for intersection can be reduced to a point-in-"polygon" test.

is whether overlap is allowed in intermediate steps of the solution process — in other words, whether the non-overlap constraint is relaxed.

The majority of solution methods which construct placements without allowing overlap are based on the geometric concept of a Minkowski sum. At least this is the term most often used for it in the mathematical literature. It is also known as hodograph, configuration space, convolution and envelope, but in the nesting literature it is primarily known as the *no-fit-polygon* (Adamowicz and Albano [1]). In the following, no-fit-polygon (NFP) is preferred even though it is not always a polygon with respect to any standard definitions. Given two polygons P and Q and a reference point on P, an NFP describes the set of locations of the reference point for which P and Q overlaps. A simple example with two non-convex polygons is shown in Figure 3.

Given two convex polygons in $\mathbb{R}^2$ with n and m edges, the NFP can be computed in time $O(n+m)$ [18]. Given two star-shaped polygons, an algorithm exists which can solve it in time $O(nm\log(nm))$ [23]. In the general case of two non-convex polygons, the NFP is not necessarily a polygon since it can include singular points and other degeneracies. Its complexity is bounded by $O(n^2m^2)$ and this bound is tight [7]. Construction algorithms are, e.g., described by Dean et al. [8], Burke et al. [6], and Bennell and Song [3]. No running times for these algorithms are stated, but worst case running time cannot be better than the worst case size of the output. These papers also contain comprehensive surveys of existing techniques for the construction of NFPs.

The NFP is often used in solution methods which use some placement heuristic to place the shapes one by one. These methods differ with regard to the order of the shapes placed and where the shapes are placed. The order of the shapes can then be changed using some meta-heuristic method [5, 16, 24]. Another usage of NFPs is described in the following subsection. Usually, NFPs are preprocessed for all pairs of polygons and if some set of rotation angles

is allowed then any combinations of these are also needed. This is a problem for problem instances with a large number of different shapes. Burke et al. [6] give an efficient algorithm for computing NFPs and one of their examples has 75 shapes where multiples of 90° rotations are allowed. Even though they can compute 634 NFPs per second for this problem instance, it takes 141.9 seconds to do it.

2.2 Compaction and separation

Li and Milenkovic [23] were the first to use linear programming to solve the nesting problem. Based on a non-overlapping placement, a set of translation vectors for the polygons are computed using linear programming such that the new placement of the polygons is more *compact* than before — without introducing any overlap. After some number of iterations (claimed to be less than 5 in practice [23]), a solution to the *translational compaction problem* is found. This problem is defined as a motion planning problem for which the goal is to minimize the area of the container. The container has fixed left, top and bottom borders while the right border is allowed to move inwards (thereby decreasing the area of the container). The polygons are moved simultaneously towards their new positions and they are not allowed to overlap each other or the boundary of the container during this movement.

The linear programs are based on NFPs for all pairs of polygons. A very similar problem, the *translational separation problem*, is solved almost identically. For this problem it is assumed that some overlap exists in a given placement and the goal is to find a set of translations vectors which eliminates all overlaps while minimizing the increase of the area of the container. Li and Milenkovic [23] use compaction and separation for a database oriented solution method. Using a database of known nestings of good quality, an initial placement is created which can contain overlaps. A separation algorithm is then used to eliminate the overlaps and subsequently compaction is used to improve this solution.

Bennell and Dowsland [2] and Gomes and Oliveira [17] use compaction and separation combined with a bottom-left placement heuristic. Results from the latter are included in the comparison of results in Section 4 and they show that this strategy can produce very good results. The results reported by Imamichi et al. [21] in a technical report are even better. They use a separation algorithm based on nonlinear programming combined with an algorithm which swaps two polygons such that their overlap with the other polygons is as small as possible (with respect to *penetration depth*, see Section 3.2).

All of the above methods make extensive use of preprocessed NFPs.

2.3 Translations

The four papers about nesting problems in this thesis are all related to an alternative to the use of NFPs; overlap minimizing translations. The measure of overlap is area (or volume in higher dimensions), but as noted in the introduction there are alternatives to this measure which are described in Section 3. For now the focus is on the area measure only.

The basic idea is to solve the following problem:

Horizontal Translation Problem. *Given a polygon Q with a fixed position, a fixed polygonal container C, and a polygon P with fixed y-coordinate, find an offset x' for P such that the (area of) overlap between P and Q is minimized (and P is within the bounds of the container C).*

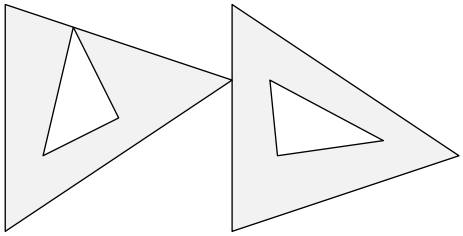


Figure 4: A very general definition of polygons is allowed. The above can be interpreted as one polygon and it can be described by a set of cycles of edges. Necessary requirements are that the cycles do not intersect each other (but they can touch) and that each cycle has no self-intersections.

Very few assumptions about the polygons P and Q are needed. They can, e.g., be in multiple separate parts and contain holes as illustrated in Figure 4 — thus they are not really polygons in the standard sense (similar to no-fit-polygons). When solving the nesting problem, a translation algorithm solving the problem above can be used to repeatedly move polygons to positions which reduces the total sum of overlap. In this setting, the polygon Q in the problem definition is actually the union of all other polygons than P.

A detailed description of an algorithm which can solve the translation problem and a proof of correctness is given by Egeblad et al. [14]$^{\mathrm{A}}$. The following is a short informal description of this algorithm.

First, assume that the container is a rectangle and thus we are looking for a value x' within an interval $[a, b]$. This makes the algorithm a bit simpler to state and understand. The basic idea of the translation algorithm is to initially place P to the left of Q and then move it through Q while iteratively maintaining a function which describes the overlap of P and Q with respect to the horizontal offset of P. This is similar to standard sweep-line algorithms, but we are sweeping with a polygon instead of a line. Initially the overlap function is simply 0 corresponding to no overlap.

The efficiency of the algorithm is based on the following fact; the area of the overlap of two polygons can be computed as a sum of signed areas based on pairs of edges, i.e., one does not need to explicitly compute the intersection of polygons. This is presented and proven by Egeblad et al. [14]$^{\mathrm{A}}$ as the Intersection Area Theorem. Now, finding the minimum area of overlap under translation, is a question of computing how much each pair of edges, one from P and one from Q, contribute to the overlap while translating the edge from P. This can be expressed as a simple function which — in the worst case — is a quadratic polynomial. Initially such a function is always 0. When the two edges start intersecting, the function is growing quadratically and when they stop intersecting it changes to a linear function. The algorithm generates a *breakpoint* whenever this function changes. Each breakpoint contains information about the distance traveled so far for the edge from P and a function describing the change in the contribution to the overlap function. The various states of the area function of two edges is illustrated in Figure 5. Note that some areas contribute negatively to the overlap function. This depends on the orientation of the edges involved [14]$^{\mathrm{A}}$.

All of the breakpoints are then sorted according to the distances and the final step is to iterate through all of the breakpoints while adding the functions. At each breakpoint the overlap function is analyzed in order to find the minimum overlap within the interval $[a, b]$.

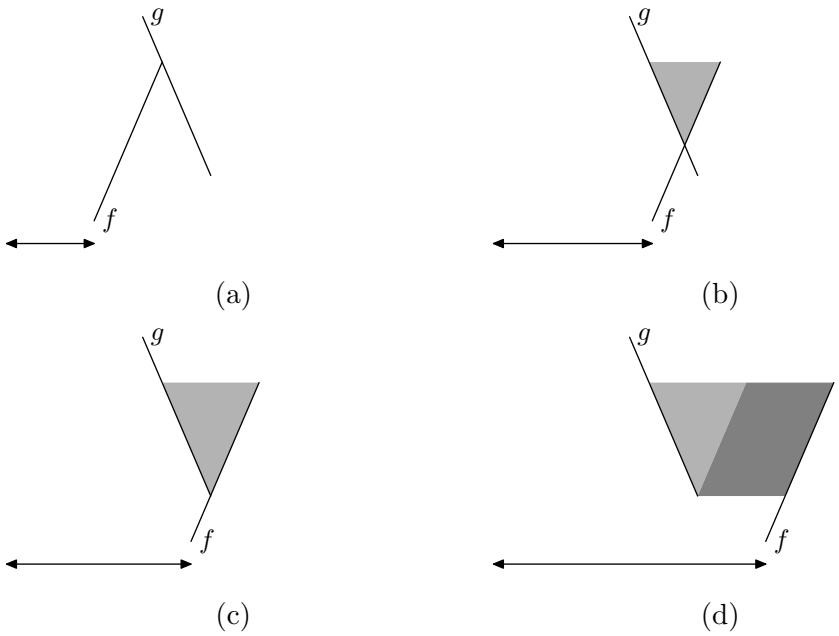


Figure 5: Two edges, f and g, where f is from the moving polygon and g is from a static polygon. Each drawing illustrates the (signed) area contributed to the sum specified in the Intersection Area Theorem [14]$^{\mathrm{A}}$. The sign depends on the orientation of the edges. The double arrows illustrate the translation distance. (a) When f is to the left of g, the contribution is 0. (b-c) When f and g intersects, the contribution is the area of a triangle. As f moves to the right, this area grows quadratically. (d) When f is to the right of g the contribution is the area of a quadrilateral. As f moves to the right, this area grows linearly.

The functions needed to make the translation algorithm work for the area measure are presented by Egeblad et al. [14][A]. It is generalized to higher dimensions by Egeblad et al. [13][C]; here the functions are polynomials of a degree matching the dimension of the problem.

If P has n edges and Q has m edges then the running time of the algorithm is $O(nm \log nm)$. Faster variants are possible if P and Q are convex, but for the solution method described for the nesting problem, Q is not going to be convex even if all of the polygons to be nested are convex. This is due to the fact that Q is the union of all other polygons than P.

Naturally, an algorithm which can solve this horizontal translation problem can also be changed to solve the symmetric vertical translation problem. A non-trivial generalization to arbitrary directions is given by Nielsen et al. [27][D].

Unlike other solution methods, no preprocessing (such as NFPs) is necessary for a solution method using the translation algorithm. This is a major advantage if handling a large number of unique shapes and/or a number of rotation angles.

It is interesting to note that there exists an algorithm to solve the more general problem of finding any position (not just in a given direction) for a given polygon which minimizes its overlap with other polygons (Mount et al. [25]) Given polygons with n and m edges, the worst case running time is $O((mn)^2)$. Their approach is based on an *arrangement* of line segments which is used to compute a two-dimensional area of overlap function which is very similar to the functions described above. Mount et al. [25] did not use this to minimize overlap, but rather to maximize overlap. Nevertheless, their approach works for both problems. No solution methods for the nesting problem have used this algorithm and it is an open question if it would be an efficient approach. It would probably require that the arrangement of line segments used could somehow be updated dynamically.

2.4 Other techniques

For the sake of completeness, it should be mentioned that other approaches to nesting problems exist which do not utilize any of the geometric techniques described above. Heckmann and Lengauer [19] compute the area of intersections between pairs of polygons directly — after moving or swapping polygons and they even considered using raster models (a pixel representation) of the shapes instead for polygons, but experiments showed that betters solutions were found using the polygons. Rasterization has also been applied to the container, i.e., only a grid of possible placements are considered. In recent work, Burke et al. [5] use a discrete set of horizontal positions while the vertical positions are continuous.

Techniques which also work in three dimensions include Φ-functions [29] and the *point space metric* [9]. The Φ-functions are used to describe the state of a pair of shapes, i.e, whether they overlap, touch or are separated. They are also defined for more general shapes than polygons/polyhedra. The theory of Φ-functions is used to pack convex polyhedra by Stoyan et al. [30]. The point space metric is a measure of the compactness of unused space in a container. This is used to guide a sequential placement of shapes [10].

The best results obtained for nesting problems in the literature are currently based on the translation technique or on compaction/separation (and thus indirectly NFPs).

3 Overlap measures

The straightforward measure of an overlap is area (volume in 3D) and this has been used successfully by Egeblad et al. [13, 14][A,C] to solve nesting problems in both 2D and 3D. As

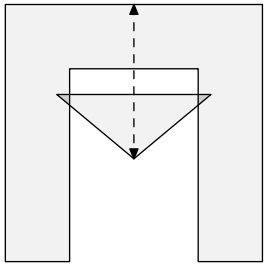


Figure 6: The area of the overlap of two polygons is often a good indication of how far one of the polygons must be translated in order to remove the overlap, but it does not always work. Here two polygons are placed such that the area of overlap between them is very small, but it does not indicate that it is close to a non-overlapping placement. The dashed line indicates the minimum distance required to translate the triangle to a position with no overlap (the penetration depth).

previously mentioned this was done by employing a succession of 1D (axis oriented) translations. For some shapes, however, this strategy can be problematic since the goal is to not only minimize overlap, but to eliminate overlap completely. Positions of a pair of shapes with a small area of overlap may not represent a good intermediate step in the search of a non-overlapping placement as illustrated in Figure 6.

In this case a better measure of the overlap is the so-called *penetration depth*, which is the minimum distance in any direction a given shape needs to be translated in order to eliminate the overlap with another shape (also illustrated in Figure 6). The principal disadvantage of using penetration depth is that it cannot be computed as efficiently as the area of overlap.

Part of the exercise in this section is to try to formalize the notion of an overlap measure. This is a subject which has received very little attention in the nesting literature; including the papers in this thesis.

In standard mathematical terms, a *measure* is a non-negative real function μ defined on a set $\mathcal{S}$ with two requirements. First, the empty set must have measure zero, i.e., $\mu(\emptyset) = 0$. Second, given any countable collection of pairwise disjoint sets $S_1, \ldots, S_n$ in $\mathcal{S}$, the measure of the union of the sets must be equal to the sum of the measures of each S_i, i.e., $\mu(\cup_{i=1}^{n} S_i) = \sum_{i=1}^{n} \mu(S_i)$.

The above definition of a measure includes the *trivial measure* which simply assigns the value 0 to any set. Clearly, the trivial measure is not useful when measuring the size of an overlap and this is one of the reasons that the following definition of an *overlap measure* is different than the standard definition of a measure. Furthermore, even though the second requirement above is true for most of the overlap measures described in this paper, it is not a necessary requirement to obtain a useful measure of an overlap.

Definition 1 (Overlap Measure)**.** *An* overlap measure *is a real-valued function μ which given any pair of polygons, P and Q, has the following properties:*

- $\mu(P, Q) = 0$ *if* $\text{int}(P \cap Q)$ *is empty and*
- $\mu(P, Q) > 0$ *if* $\text{int}(P \cap Q)$ *is non-empty,*

where $\text{int}(P \cap Q)$ *is the interior of the intersection of* $P \cap Q$*.*

Note that in order to obtain an intuitive measure for the size of an intersection of two polygons, one needs to consider the interior of the intersection. This is necessary to ensure that an overlap measure does not take on a positive value for polygons which are only touching. Also note that the overlap measure is based on two polygons instead of simply being a function that could be applied to the result of the intersection of the two polygons (more precisely, the closure of the interior of the intersection which is also a polygon). This distinction is important when considering penetration depth measures.

A range of overlap measures are discussed in the following subsections. Each overlap measure has its own strengths and weaknesses and typically there is a trade-off between how well an overlap measure describes intersections and how fast it can be computed. In general, we assume that the application of each overlap measure is the minimization of overlap used for solving the nesting problem as described by Egeblad et al. [14]$^{\mathrm{A}}$, i.e., the overlap measure is used to determine the best position (minimum overlap with respect to the overlap measure) for a given polygon when given a fixed translation direction. Without loss of generality, we continue to assume that this is a *horizontal* direction, i.e., only the x-coordinate of the position of the polygon is to be changed.

Some overlap measures have the property that they can be calculated by simply considering pairs of edges of the given polygons. Such a decomposition is useful for efficiently computing these overlap measures as briefly discussed in Section 2.3. This is very similar to the standard computation of the area of a polygon or the volume of a convex polytope (Lawrence [22]). A formal definition of such overlap measures can be given as follows.

Definition 2 (Decomposable Overlap Measure)**.** *A* decomposable overlap measure μ *is an overlap measure which given any pair of polygons, P and Q, with sets of edges F and G, for which there exists a real-valued function μ' such that*

$$\mu(P, Q) = \sum_{f \in F, g \in G} \mu'(f, g). \tag{1}$$

Most of the overlap measures discussed in this section are illustrated in Figure 7 and various special cases useful for illustrating strengths and weaknesses are shown in Figure 8. Based on these special cases, a comparison of the overlap measures is also presented in Section 3.5.

3.1 Indicator Measure

In some cases, the only desired information is whether two shapes overlap or, in the context of translation problems, when they overlap. For this purpose a very simple *indicator measure* is needed which is 1 when the shapes are overlapping and 0 when they are not overlapping. Any other overlap measure can be used to provide an indicator measure by mapping any positive value of the overlap measure to 1. We return to this subject in some of the following subsections with respect to both the use of an indicator measure and efficient algorithms to obtain it.

3.2 Penetration Depth Measures

Given two overlapping polygons, the *penetration depth* is the minimum translation distance (in any direction) required in order to separate them. An example is given in Figure 7a. In the context of nesting problems and the overlap minimization approach described in the

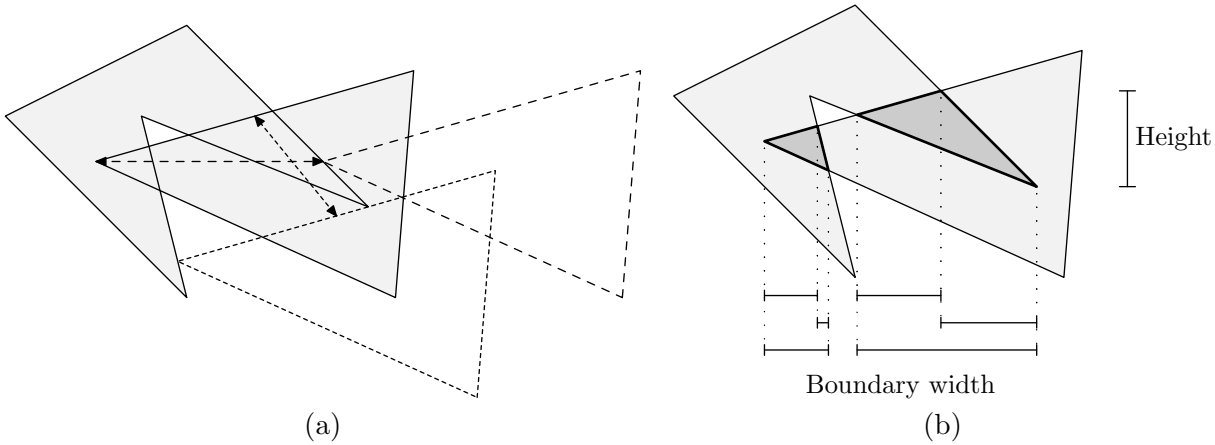


Figure 7: Examples of various overlap measures for a pair of polygons. (a) The length of the shortest double arrow is the penetration depth and the length of the longer horizontal double arrow is the more restricted horizontal (1D) penetration depth. (b) The size of the area with a dark grey color is the area measure of the overlap. The length of the bold lines is the boundary measure of the overlap. The height of the overlap is the height measure of the overlap. The sum of lengths of all vertical projections of boundary line segments is the boundary width measure of the overlap (the sum of the lengths of the horizontal intervals in the drawing).

introduction, the penetration depth is a very intuitive measure of how bad a given overlap is. It can even handle difficult overlap situations such as the one shown in Figure 8g. More formally, penetration depth can be stated as follows.

Definition 3 (Penetration Depth)**.** *Given two polygons P and Q in $\mathbb{R}^2$, the penetration depth is the smallest possible length of a vector $v \in \mathbb{R}^2$ such that $P^v \cap Q = \emptyset$ where P^v is P translated by v.*

Given two convex polygons with n and m edges, the penetration depth can be calculated in time $O(\log(n+m))$ [11]. In general, the main problem with the penetration depth measure is that it is not easily computed. The standard approach for computing the penetration depth of two polygons P and Q is to first compute their NFP in a preprocessing phase. The penetration depth is then equal to the shortest distance from the reference point of P to a boundary point of the NFP. For non-convex polygons the situation is more complicated as illustrated by some of the construction algorithms mentioned in Section 2.1, but the number of edges in the NFP is naturally an upper bound, i.e., $O(n^2m^2)$.

In this and the following subsections we describe a range of alternatives to the penetration depth measure. They can all be computed more efficiently than penetration depth and the algorithms described do not rely on any preprocessing. The disadvantage of these alternative overlap measures is that they can fail to describe some overlaps appropriately, especially when given highly non-convex shapes. The descriptions of the alternative overlap measures include a comparison with penetration depth. We say that an overlap measure *underestimates* an overlap whenever it has a relatively smaller value than the penetration depth of the overlap. Similarly, we say that it *overestimates* an overlap whenever it has a relatively larger value.

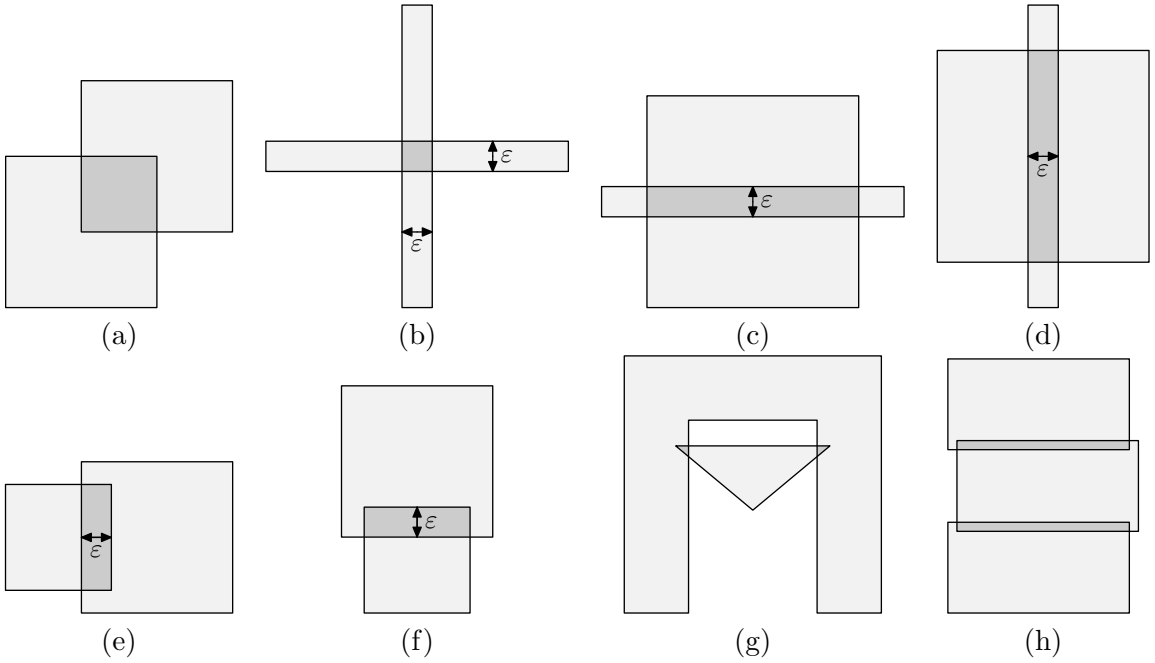


Figure 8: Examples of overlapping polygons. (a) A simple overlap handled well by most overlap measures. (b)-(f) If ε is small then various overlap measures might underestimate or overestimate the overlap. (g) This kind of overlap is only handled well by penetration depth measures. (h) The middle rectangle has a constant sum of penetration depths with the neighboring rectangles for any short translation in any direction.

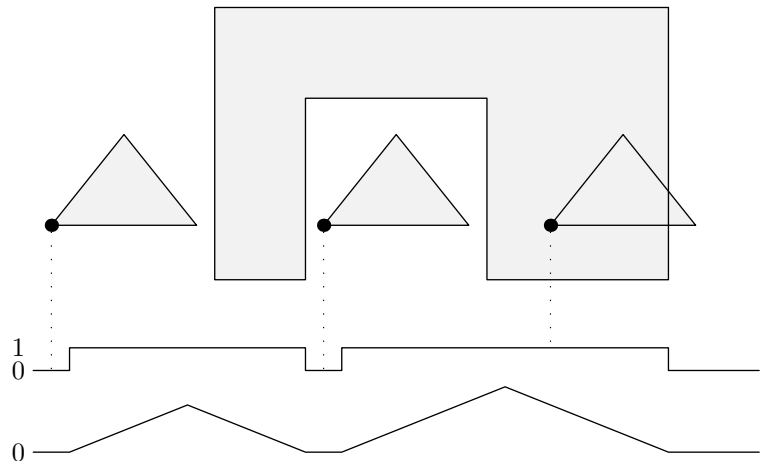


Figure 9: A triangle translated through a polygon. The reference point of the triangle is the bottom-left corner. Two functions are illustrated: The indicator function which is always 0 or 1 corresponding to whether the triangle overlaps with the polygon and the 1D penetration depth function which can easily be derived from the indicator function.

These definitions are of an intuitive nature and quite informal, but it is still useful when discussing the advantages and disadvantages of each overlap measure.

A closely related alternative to penetration depth is a simplification in which one only considers the penetration depth in a given direction (*1D penetration depth*). In Figure 7a this is illustrated with a horizontal direction. Using the NFP this can quite easily be calculated. This is the overlap measure used by Umetani et al. [32] to solve the nesting problem in 2D; they repeatedly perform axis-aligned translations with 1D penetration depth as the objective value. Further below, it is shown how this can be also be done without using NFPs.

An important feature of 1D penetration depth is that it never underestimates an overlap. The downside is that it can easily overestimate an overlap. For example, a horizontal penetration depth measure overestimates the overlap in Figures 8c and 8f.

Not all overlap measures are decomposable. In particular, penetration depth measures are not decomposable. No formal proof is given here, but consider the example in Figure 8g; the horizontal penetration depth of the triangle depends on edges that are not involved in the overlap itself. It seems highly unlikely that the penetration depth can be expressed as a sum of (signed) values based on pairs of edges.

The information needed to compute 1D penetration depth is a by-product of the computation of any other overlap measure. What is basically needed is best described by the indicator measure since a penetration depth measure is only concerned about the distance to a non-overlapping position. In Figure 9, the value of the indicator measure during a horizontal translation is illustrated. Given that this information is described by a piece-wise constant (0 or 1) function I, a piece-wise linear function, h, describing the horizontal penetration depth can easily be constructed (also shown in Figure 9).

$$h(x) = \min_{x' \in \mathbb{R}, I(x')=0} |x - x'|. \tag{2}$$

In practice, the penetration depth function can easily be computed by adding a second

phase to the standard translation algorithm. Assume that P is the polygon to be translated and that the other polygons are $Q_1, \ldots, Q_k$. Note that until now we have just used Q to denote the union of these polygons, but here we need to handle the static polygons more carefully. In the first phase, some indicator measure is used to generate the usual breakpoints for each Q_i. In the second phase, a new set of breakpoints is generated based on when overlap begins and ends with each of these polygons. When an overlap begins, the 1D penetration depth increases linearly. When an overlap ends, the horizontal penetration depth changes to 0 and in retrospect one knows that half way between the start and the end of the overlap, the function should have changed from linearly increasing to linearly decreasing.

Now all there is left to do, is to sort the breakpoints from the second phase (although the sorting can almost be avoided) and find the minimum of the sum of the penetration depth functions. This is done exactly as it is done for the area measure. Clearly, the running time is not worse than what was obtained for the decomposable overlap measures and the number of phase two breakpoints is less than or equal to the number of phase one breakpoints.

Since any overlap measure can be used as an indicator measure, it would be sensible to choose an overlap measure which is easy to compute. This is where some of the less promising overlap measures described in the following subsections suddenly become very interesting. In this context, it does not matter how bad a given overlap measure behaves in a comparison with other overlap measures with respect to over- and underestimation. All that matters is how easily it can be computed.

A viable alternative to the approach described here is to use the previously described NFPs to determine when overlaps begin and end when translating in a given direction [32]. Depending on NFPs can be a problem though, e.g., if one wants to handle free orientation of shapes or generalize the solution method to higher dimensions.

3.3 Area Measure

A natural measure of the overlap of two polygons is the area of the overlap as illustrated in Figure 7b. In contrast to 1D penetration depth, the *area measure* never overestimates an overlap, but it can easily underestimate it. This is illustrated in Figure 8g, but Figures 8b-d are also very problematic when ε is small. The main advantage is that the area measure can be computed very efficiently as already discussed in Section 2.3.

For the sake of completion, it should be mentioned that projections of the area measure are also overlap measures, e.g., the height of an overlap as illustrated in Figure 7b. It is an open question how these overlap measures can be calculated efficiently for non-convex polygons, but with respect to convex polygons these projections are essentially identical to the projections described in the following section for which an efficient algorithm do exist.

3.4 Boundary Measures

Instead of measuring the area of an overlap, one could measure the boundary of the overlap. This would simply be the sum of the lengths of the boundary edges of the overlap as illustrated in Figure 7b. Interesting generalizations to higher dimensions are also possible, e.g., in 3D one could measure the area of the faces of the overlap or measure the lengths of the edges of the faces of the overlap. One could even count the end-points of the edges of the overlap (in any dimension). This last option is a promising indicator measure although one should be careful with respect to precision issues with floating point computations.

We use the term *boundary measure* to describe the lengths of the boundary edges of an overlap. The boundary measure can both overestimate and underestimate an overlap. In Figures 8b and g, measuring the boundary of the overlap is a serious underestimation while it is a serious overestimation in Figures 8e and 8f. It could even be seen as a slight overestimation in Figures 8c and 8d.

The boundary measure is also a decomposable overlap measure and it is somewhat easier to understand the translation algorithm when using this measure. We do not provide all details here, but the boundary measure is part of the implementation described in Section 4. It is illustrated in Figure 10a-d for two simple triangles. Whether the length of a line segment is added or subtracted from the objective function depends on its orientation. In the figure, this is illustrated with bold (added) and dashed (subtracted) line styles. As required, the overlap function in the end becomes 0 because all lines are added and subtracted once. In the intermediate steps the signed sum of lengths is exactly the length of the boundary of the intersection. Preliminary computational experiments show that the boundary measure works as expected, but they do not indicate that it (in general) provides better solutions to standard benchmark nesting problems.

As with the area measure, the boundary measure has projective variants. This is illustrated in Figure 7b for two overlapping polygons and a vertical projection (a projection onto a horizontal line). In the following, it is referred to as the *boundary width measure*. Similarly the *boundary height measure* refers to a horizontal projection. It is important to note that, e.g., boundary width is not just the width of an overlap, but the accumulated width of all boundary segments in the overlap. Given a convex overlap area this is exactly two times the width of the overlap; for non-convex overlaps it may be more.

The boundary height measure is illustrated in Figure 10e-h in order to emphasize an important fact. For this overlap measure, fewer pairs of edges need to be considered when computing its value. The illustration does not explicitly include the projection of the boundary, but it does emphasize that fewer breakpoints are needed to compute it correctly. Breakpoints are only needed for pairs of edges (line segments) with opposite orientations. Adding the height of the bold lines and subtracting the height of the dashed lines corresponds to measuring the height of all the line segments of the boundary of intersection.

The advantages of a projected boundary measure are not immediately obvious, but it turns out to be a really good candidate if an indicator measure is needed. There are three reasons for this. First, it requires fewer breakpoints. Second, the functions are linear expressions. Third, if used as an indicator measure, the linear expressions can be avoided and replaced by a set of current overlap intervals. This can be implemented (and we have done this) to provide an indicator measure which is robust with respect to floating point calculations.

3.5 Comparison

A crude comparison of most of the overlap measures described is presented in Table 1. The comparison is based on the assumption that penetration depth is the ideal overlap measure. For some applications of overlap measures this might not be the case. The overlap examples in Figure 8 are used for the comparison. Note that these examples do not cover all possible overlap situations and they could make some of the measures look better or worse than they would be in practice. For example, the advantage of the penetration depth measures depend strongly on non-convex shapes or stick-like shapes like those in Figures 8c-e. Also remember that the definitions of over- and under-estimation are very informal.

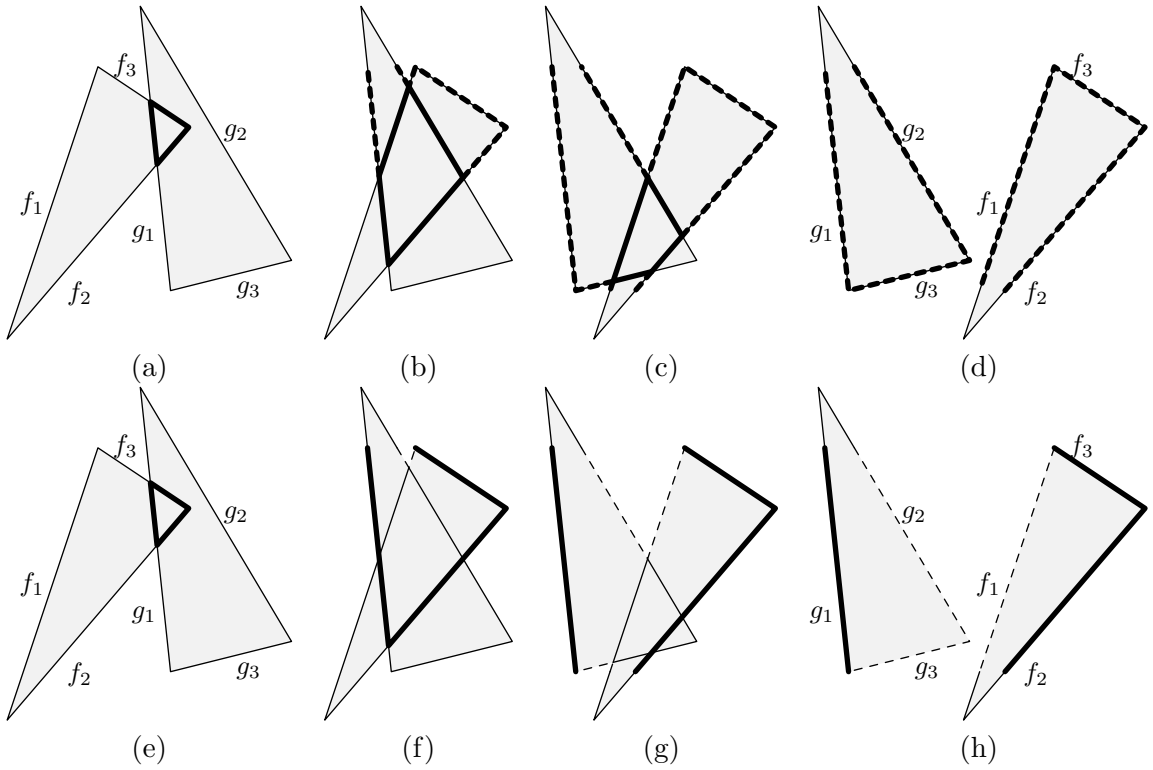


Figure 10: Two examples of the translation of a polygon with edges f_1, f_2 and f_3 through a polygon with edges g_1, g_2 and g_3. a-d) Boundary measure: The length of bold lines are added to the value of the overlap and the length of dashed lines are subtracted. e-h) Boundary height measure: The length of the projection of bold lines onto the y-axis are added to the value of the overlap and the length of the projection of dashed lines onto the y-axis are subtracted. Note that the projections are *not* shown, i.e., one must be careful and only consider the length of the projections of the bold (added) and the dashed (subtracted) line segments.

Overlap measure	a	b	c	d	e	f	g
Penetration depth	=	=	=	=	=	=	=
Horizontal depth	=	=	>	=	=	>>	=
Volume	=	<<	<<	<<	=	=	<<
Boundary	=	<<	<	<	>>	>>	<<
Boundary height	=	<<	<<	<	>>	=	<<

Table 1: A comparison of a range of different overlap measures where penetration depth is considered as being the ideal overlap measure. The letters a-g refer to the overlap examples shown in Figure 8. An equality symbol means that the overlap is (relatively) correctly measured. A less than symbol means that the overlap is underestimated and a greater than sign means that the overlap is overestimated. Double less/greater than symbols means that the overlap is severely under/over-estimated. Note that the area measure never overestimates an overlap and that the horizontal penetration depth measure never underestimates an overlap.

3.6 Combined Overlap Measures

Eisenbrand et al. [15] use a combination of volume and penetration depth in three dimensions to solve a problem with packing rectangular boxes in the trunk of a car. The basic solution method is relatively simple. Given a placement, a potential function is used to determine its validity. Trial moves are then made iteratively, either translating or rotating a box, and the potential of the new placements are calculated. Simulated annealing then controls which trial moves are accepted. The potential function includes both the volume and the penetration depth of all intersections in a given placement. This combination of overlap measures is motivated by the fact that some intersections are not handled well by volume and others are not handled well by penetration depth. The latter deserves an explanation since we have so far treated penetration depth as the ideal overlap measure. The problem for Eisenbrand et al. is that they only make small moves and in some cases the penetration depth is a constant for the local neighborhood which means that it can be difficult to get out of this local plateau. An example in 2D of this problem is given in Figure 8h. Three rectangles are placed next to each other with small overlaps. For any small local translation, the rectangle in the middle has a locally constant sum of penetration depths with the two neighboring rectangles. In this case, it would be handled better by an area or a boundary measure since they would not be locally constant under horizontal translations. Using larger neighboring rectangles one could construct an example for which all three overlap measures would be constant, but for Eisenbrand et al. this cannot occur since all boxes are identical.

The main problem with penetration depth is that it can be difficult to compute, especially when dealing with non-convex shapes. We have already stated that all other measures described here can be computed efficiently when solving the horizontal translation problem, but all of these alternatives to penetration depth clearly also have disadvantages as illustrated in Figure 8.

In general, one could combine k different overlap measures with assigned weights. Each weight should somehow depend on how well the corresponding overlap measure has performed during the solution process. A problem with this approach could be that some overlap measures are not of the same order of magnitude, e.g., the area measure has a quadratic growth while the 1D penetration depth has a linear growth. An alternative to combining overlap measures using weights is to dynamically shift between different overlap measures. This could be controlled by some meta-heuristic which attempts to use the overlap measure that works best for the given problem instance at a given time. Combining overlap measures as briefly sketched here is a very promising venue for future research.

4 Computational Experiments

The first paper in this thesis (Egeblad et al. [14]$^{\mathrm{A}}$) describes and reports results for an implementation which was done 5 years ago (a few improvements were done 3 years ago). The current implementation is based on the original work, but only few lines of code have survived (somewhere along the way the name changed from 2DNest to Nest2D). The most significant change is that the current implementation can be compiled such that it only uses rational numbers. This makes the solution method very slow, but it also makes it much easier to implement new features — and to debug any problems. Precision problems when using floating point numbers are more easily identified and handled more efficiently. The end result is that the implementation is now simpler and more robust — and naturally also more

	Current Nest2D							Old 2DNest	
	Initial	10	60	300	600	Worst	Best	600	Max.
Albano	69.68	85.47	86.59	87.34	87.50 (0.2)	87.11	87.90	86.96	87.44
Dagli	60.38	83.08	84.76	85.89	86.17 (0.5)	85.51	87.51	85.31	85.98
Dighe1	39.37	81.57	87.79	98.49	99.99 (0.0)	99.99	100.00	93.93	99.86
Dighe2	49.26	88.44	99.91	99.91	99.99 (0.0)	99.99	100.00	93.11	99.95
Fu	59.38	88.81	89.79	90.43	91.03 (0.7)	89.92	91.94	90.93	91.84
Jakobs1	54.44	84.51	87.80	88.86	89.05 (0.1)	88.92	89.09	88.90	89.07
Jakobs2	53.61	78.10	80.20	80.67	80.71 (0.7)	80.30	82.44	80.28	80.41
Mao	50.20	79.34	81.41	82.76	83.08 (0.6)	82.03	84.23	82.67	85.15
Marques	68.49	87.12	88.46	88.91	89.12 (0.4)	88.53	89.85	88.73	89.17
Shapes0	44.83	63.14	64.95	65.75	66.07 (0.5)	65.16	66.74	65.42	67.09
Shapes1	44.83	69.53	71.25	72.04	72.35 (0.9)	71.15	73.83	71.74	73.84
Shapes2	60.00	77.99	79.51	80.22	80.69 (0.6)	79.92	82.32	79.89	81.21
Shirts	71.05	83.54	85.37	86.17	86.61 (0.3)	85.83	87.35	85.73	86.33
Swim	44.16	64.83	69.63	71.38	72.00 (0.5)	71.16	73.04	70.27	71.53
Trousers	66.61	87.31	88.74	89.35	89.64 (0.2)	89.14	90.04	89.29	89.84
	55.75	80.19	83.08	84.55	84.93 (9.2)	84.31	85.75	83.54	85.25

Table 2: Utilization percentages after 0, 10, 60, 300 and 600 seconds for the current implementation of the solution method by Egeblad et al. [14]^{A}. All of these values are averages of 20 runs. Columns 7 and 8 state the worst and best result in these 20 runs. Columns 9 and 10 state the average and the best results as reported by Egeblad et al. [14]^{A} for the original implementation. The best average results and the best of the best results are underlined.

versatile since it supports repeated patterns, aligned translations, free orientation of shapes and various overlap measures as described in the papers in this thesis.

The (essentially) new implementation was used for the 3D experiments in the paper handling nesting problems in higher dimensions (Egeblad et al. [13]^{C}). None of the papers repeat the experiments in the first paper for the two-dimensional case, but it would be interesting to see how well the new implementation performs compared to the old implementation and other new results from the literature. This is the main purpose of the following experiment.

The problem instances have been selected from the literature and can be downloaded from the ESICUP homepage[1]. There are 15 problem instances and various degrees of freedom regarding orientation are allowed (0, 90°or 180°) as shown in the third column of Table 3. Illustrations of the problem instances can be found in the paper by Egeblad et al. [14]^{A}.

A comparison of the old and the new implementation can be seen in Table 2. The utilization of the initial solution (same algorithm as in the original implementation) and the utilization after 10, 60 and 600 seconds (10 minutes) are reported. All of these values are averages of 20 runs. The worst and the best of the results are reported for the 10 minute runs, and the rightmost two columns contain the average and the best results from Egeblad et al. [14]^{A}. The new experiments were conducted on a 2.16 GHz Intel Core Duo processor. The old experiments were conducted on a 3GHz Pentium 4.

The first thing to notice is that the average (of the averages) utilization after 300 seconds

[1] `http://www.fe.up.pt/esicup`

			Average results			Best results		
	Size	Deg.	Nest2D	ILSQN	SAHA	Nest2D	ILSQN	SAHA
Albano	24	180°	87.50	87.14	84.70	87.90	88.16	87.43
Dagli	30	180°	86.17	85.80	85.38	87.51	87.40	87.15
Dighe1	16		99.99	90.49	82.13	100.00	99.89	100.00
Dighe2	10		99.99	84.21	84.17	100.00	99.99	100.00
Fu	12	90°	91.03	87.57	87.17	91.94	90.67	90.96
Jakobs1	25	90°	89.05	84.78	75.79	89.09	86.89	78.89
Jakobs2	25	90°	80.71	80.50	74.66	82.44	82.51	77.28
Mao	20	90°	83.08	81.31	80.72	84.23	83.44	82.54
Marques	24	90°	89.12	86.81	86.88	89.85	89.03	88.14
Shapes0	43		66.07	66.49	63.20	66.74	68.44	66.50
Shapes1	43	180°	72.35	72.83	68.63	73.83	73.84	71.25
Shapes2	28	180°	80.69	81.72	81.41	82.32	84.25	83.60
Shirts	99	180°	86.61	88.12	85.67	87.35	88.78	86.79
Swim	48	180°	72.00	74.62	72.28	73.04	75.29	74.37
Trousers	64	180°	89.64	88.69	89.02	90.04	89.79	89.96
			84.93	82.74	80.12	85.75	85.89	84.32

Table 3: Average and best results are compared for three different solution methods. The highest utilization values are underlined. The number of shapes and the degree of rotation allowed are also reported. Processors are 2.16 GHz Intel Core Duo (Nest2D), 2.8GHz Intel Xeon (ILSQN), and 2.4GHz Pentium IV (SAHA).

(final row of the table) is about 1 percentage point better than the average for the old experiments. The gain from running another 300 seconds is just 0.38 percentage points, but then each of the averages of the problems instances are also better than for the old experiments. With respect to the best placements found, the new implementation is in general better, but there are 3 exceptions; Mao, Shapes0 and Shapes1. The average utilization values of the best placements is on average 0.5 percentage points better. Considering that the two implementations are basically equivalent, small differences in the implementations and better handling of precision issues have a remarkable effect.

We also compare the results with existing competing solution methods in Table 3. The best of these are currently ILSQN by Imamichi et al. [21] and Gomes and Oliveira [17]. Best results are obtained by ILSQN and Nest2D, but SAHA often obtains a second place and is therefore included in this comparison. Running times vary for SAHA (also see Egeblad et al. [14]^{A}) while the results by Imamichi et al. [21] are based on 10 runs of 600 or 1200 seconds depending on the size of the problem. Other recent results from the literature are reported by Burke et al. [5] and Umetani et al. [32], but they were not competitive with the above.

The basic solution method in Nest2D is relatively simple and it is mainly based on a simple algorithm for translating a polygon. The initial solution constructed is not very good, but this seems to only have a minor effect on the final results since few seconds are needed to improve the solutions considerably. The number of translations per second ranges from about 1000 to about 26000 and it would seem that these translations could be used more efficiently; if not to obtain better solutions then to obtain them faster. This could, e.g., involve an

efficient use of the directed translations (Nielsen et al. [27]^{D}), overlap measures such as 1D penetration depth or swapping shapes as done by Imamichi et al. [21]. The full potential of using translations to solve nesting problems has most likely not yet been achieved.

5 Conclusions and Future Directions

The two-dimensional nesting problem is a very difficult problem and in order to create a state-of-the-art solution method, a great amount of work has to be done. The translation algorithm — more or less the main subject of all the nesting papers in this thesis — is a relatively simple tool for solving nesting problems and it can be implemented in just a few hundred lines of code. Using an appropriate overlap measure such as boundary height, it also handles floating point precision issues extremely well. We show how it can also be adapted to handle repeated patterns [26]^{B} and free orientation of shapes [27]^{D}, and we also show how it can be generalized to higher dimensions [13]^{C}.

When comparing solution methods, it is always important to remember the effect of many parameters, e.g., efficiency of the code, tuning of meta-heuristic techniques and the use of strategies such as multi-level approximations of shapes and dynamic clustering. A comparison of geometric techniques is therefore very difficult and it is not possible to conclude that translations are better or worse than NFP bottom-left construction algorithms or compaction/separation strategies. NFPs provide a better measure of overlap (for nesting problems), but they are also primarily useful if they can be preprocessed. Furthermore, it is not clear whether NFPs can be efficiently generalized to three or more dimensions.

In this introduction to the nesting problem, we have introduced a simple definition of an overlap measure for geometric shapes, and in particular, polygons. We have shown that existing overlap measures such as penetration depth and area are members of a much larger family of overlap measures. We think of this family of overlap measures as a toolbox of techniques which can help solve various optimization problems. Our application for these overlap measures has been an existing solution method for the nesting problem which repeatedly translates shapes to positions with less overlap. Other optimization problems and solution techniques might also benefit from this toolbox. This could, e.g, be problems for which special kinds of maximum overlap are desired.

The various overlap measures do not only differ with regard to how overlap is measured. More importantly they differ with regard to ease of implementation (especially in higher dimensions) and running time for the computation. The translation algorithms presented can handle arbitrary polygons and cover a wide range of overlap measures including 1D penetration depth, decomposable overlap measures such as area and boundary and projections of these measures. Although we have not discussed it in detail, all of this can be generalized to higher dimensions. At least in two dimensions, more general shapes than polygons could probably also be handled efficiently by using boundary height (and thus also 1D penetration depth). After splitting the shapes into y-monotone curves the only serious geometric problem would be how to compute the distance between two such curves.

Our focus has primarily been on the translation algorithm itself and how variations of it can handle different problems. The remaining parts of the solution method, as implemented in Nest2D, has not changed considerably since the initial implementation. Two subjects which are especially interesting for future research are as follows.

- Dynamically clustering/pairing shapes in the solution process.

- Variations of the local search neighborhood, especially introducing swaps of shapes similar to the work of Imamichi et al. [21].

Another very interesting subject would be a generalization to three dimensions of the free orientation strategy used for two-dimensional polygons by Nielsen et al. [27]^{D}.

References

[1] M. Adamowicz and A. Albano. Nesting two-dimensional shapes in rectangular modules. *Computer Aided Design*, 1:27–33, 1976.

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

[3] J. A. Bennell and X. Song. A comprehensive and robust procedure for obtaining the nofit polygon using Minkowski sums. *Computers & Operations Research*, 35(1):267–281, 2008.

[4] 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.

[5] E. 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.

[6] 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(1):27–49, 2007.

[7] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational geometry: Algorithms and applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997. ISBN 3-540-61270-X.

[8] H. T. Dean, Y. Tu, and J. F. Raffensperger. An improved method for calculating the no-fit polygon. *Computers & Operations Research*, 33(6):1521–1539, 2006.

[9] 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.

[10] 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.

[11] D. Dobkin, J. Hershberger, D. Kirkpatrick, and S. Suri. Computing the intersection-depth of polyhedra. *Algorithmica*, 9:518–533, 1993.

[12] 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.

[13] J. Egeblad, B. K. Nielsen, and M. Brazil. Translational packing of arbitrary polytopes. Submitted.

[14] 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.

[15] 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.

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

[17] 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.

[18] L. J. Guibas, L. Ramshaw, and J. Stolfi. A kinetic framework for computational geometry. In *FOCS*, pages 100–111. IEEE, 1983.

[19] 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.

[20] T. Imamichi and H. Nagamochi. A multi-sphere scheme for 2D and 3D packing problems. In *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics. International Workshop, SLS 2007*, pages 207–211, 2007.

[21] T. Imamichi, M. Yagiura, and H. Nagamochi. An iterated local search algorithm based on nonlinear programming for the irregular strip packing problem. Technical Report 2007-009, Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, 2007.

[22] J. Lawrence. Polytope volume computation. *Mathematics of Computation*, 57(195):259–271, 1991.

[23] 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.

[24] H.-y. Liu and Y.-j. 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.

[25] D. M. Mount, R. Silverman, and A. Y. Wu. On the area of overlap of translated polygons. *Computer Vision and Image Understanding*, 64(1):53–61, 1996.

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

[27] B. K. Nielsen, M. Brazil, and M. Zachariasen. Using directed local translations to solve the nesting problem with free orientation of shapes. Submitted.

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

[29] 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.

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

[31] P. E. Sweeney and E. R. Paternoster. Cutting and packing problems: A categorized, application-orientated research bibliography. *Journal of the Operational Research Society*, 43(7):691–706, 1992.

[32] S. Umetani, M. Yagiura, T. Imamichi, S. Imahori, K. Nonobe, and T. Ibaraki. A guided local search algorithm based on a fast neighborhood search for the irregular strip packing problem. In *Proceedings of International Symposium on Scheduling 2006*, pages 126–131, 2006.

[33] X. Yan and P. Gu. A review of rapid prototyping technologies and systems. *Computer Aided Design*, 28(4):307–318, 1996.

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:

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

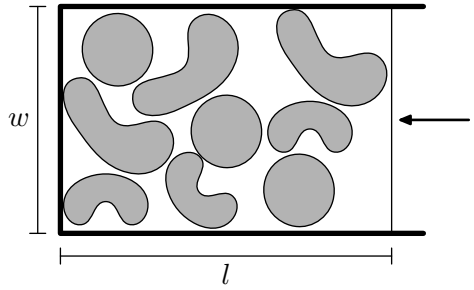


Figure 1: The Strip Nesting Problem. Place a number of stencils on a strip with width w such that no two stencils overlap and the length l of the strip is minimized.

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

- *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 [8–11, 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.

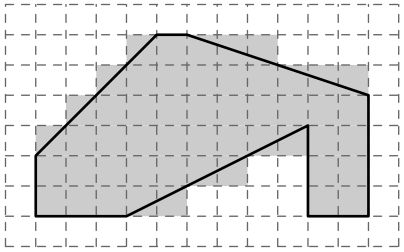


Figure 2: The raster model requires all stencils to be defined by a set of grid squares. The drawing above is an example of a polygon and its equivalent in a raster model.

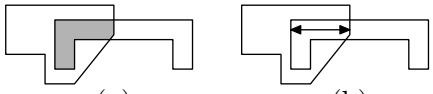


Figure 3: The degree of overlap can be measured in various ways. Here are two examples: (a) The precise area of the overlap. (b) The horizontal intersection depth.

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.

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 [5–7, 21].

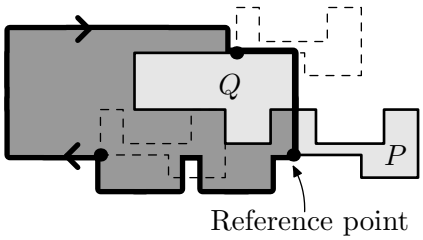


Figure 4: Example of the *No-Fit-Polygon* (thick border) of stencil P in relation to stencil Q. The reference point of P is not allowed inside the NFP if overlap is to be avoided.

No-Fit-Polygons (NFP)

Legal placement methods very often use the concept of the *No-Fit-Polygon* (NFP) [1–3, 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 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.
- **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 $N : \mathcal{P} \to 2^{\mathcal{P}}$ 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].

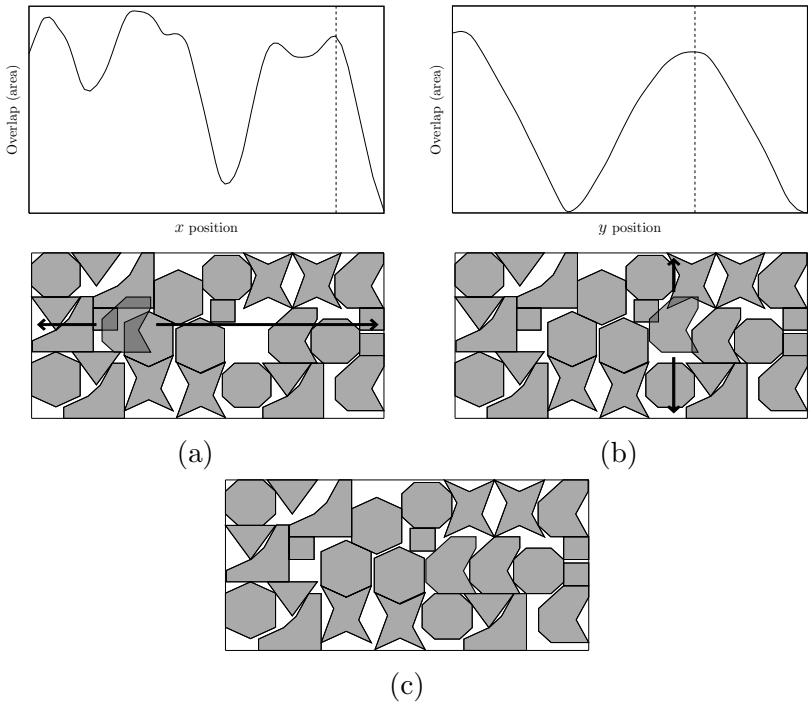


Figure 5: Example of a local search. (a) One polygon overlaps with several other polygons and is selected for optimization. In the top row we have drawn the amount of overlap as a function of the leftmost x-coordinate of the polygon. The positions beyond the dashed line are illegal since the polygon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

5.1 Features and penalties

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

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

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.

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.

Algorithm 1: Guided Local Search for Nesting Decision Problem

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

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 4 (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).

Definition 5 (Edge Count Functions)**.** *Given a set of edges E we define two edge count functions, $\overleftarrow{f_E}(p), \overrightarrow{f_E}(p) : \mathbb{R}^2 \to \mathbb{N}_0$,*

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

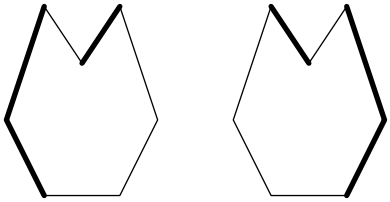


Figure 6: Positive and negative edges of a polygon according to Definition 7.

Definition 6 (Polygon)**.** *A polygon P is defined by a set of edges E. The edges must form one or more cycles and no pair of edges from E are allowed to intersect. The* interior *of the polygon is defined by the set*

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

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

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

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

Definition 7 (Sign of Edge)**.** *Given a polygon P defined by an edge set E we say an edge $e \in E$ is* positive *if*

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

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 4.

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 8 (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 9 (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.

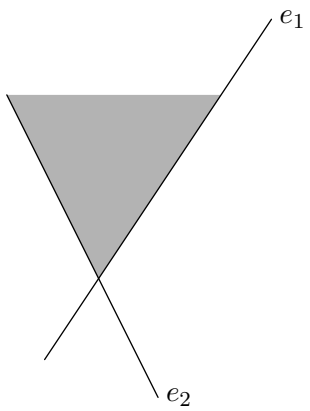


Figure 7: The edge region $R(e_1, e_2)$ of two edges e_1 and e_2 (see Definition 9).

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).

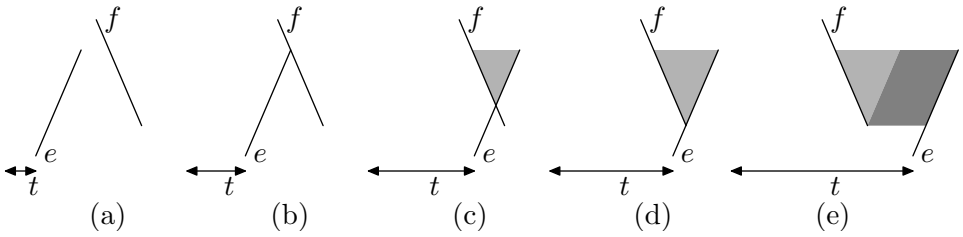


Figure 8: The edge region $R(e, f)$ of two edges as e is translated t units from left to right. (a) e is completely left of f. (b) e adjoins f. (c) e crosses f. (d) e and f are adjoined again. (e) e is to the right of f. Notice that $R(e, f)$ is either $\emptyset$, a triangle or a triangle combined with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

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

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

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

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

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

Determining the minimum overlap translation

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

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

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

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

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

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

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

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.

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 $\alpha(t)$

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.

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

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 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*

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.

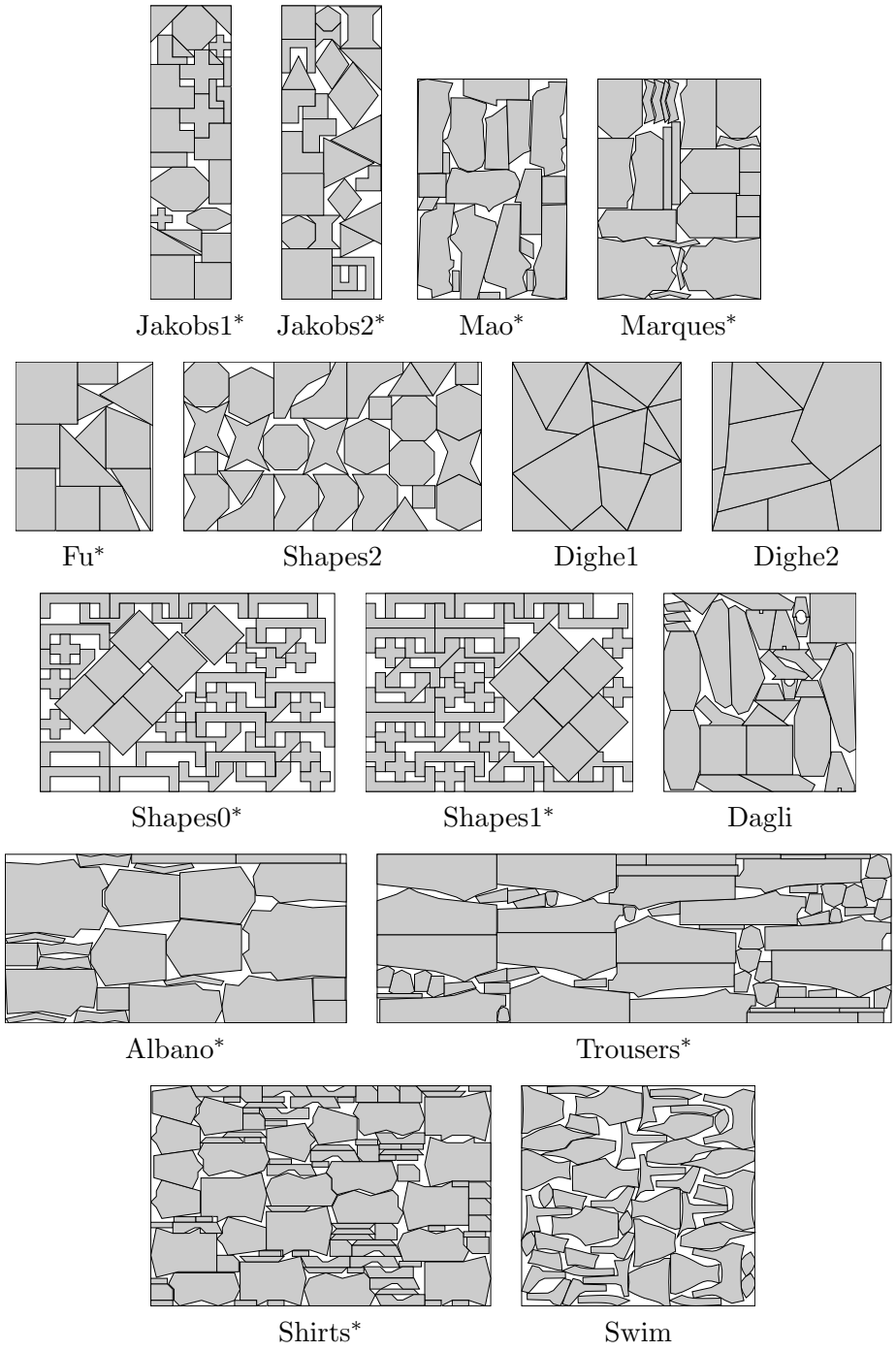


Figure 9: The best solutions found by 2DNest easily comparable with the ones shown in Gomes and Oliveira [20].

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

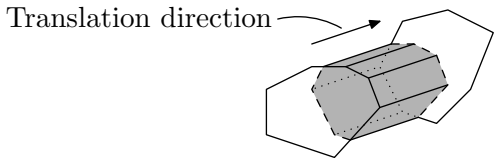


Figure 10: An illustration of the face region between two faces. The faces are not necessarily parallel, but the sides of the face region are parallel with the translation direction. The face region would be more complicated if the two faces were intersecting.

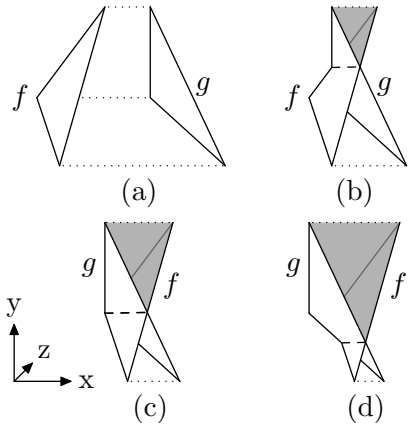


Figure 11: Translation of a triangle f through another triangle g along the x-axis, where the triangles have the same projection onto the yz-plane. The face region $\mathbf{R}(f, g)$ changes shape each time two corner points meet.

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.

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.

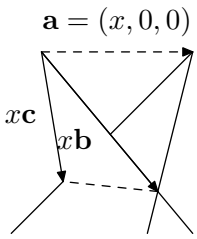


Figure 12: The volume of the above tetrahedron can be calculated from the three vectors **a**, **b** and **c**. In our case **b** and **c** are linearly dependent on x which is the length of **a** (and the translation distance since the tetrahedron started growing).

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.

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:

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

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 × 2.00 × 2.00
Part2	24	2.88	1.43 × 1.70 × 2.50
Part3	28	0.30	1.42 × 0.62 × 1.00
Part4	52	2.22	1.63 × 2.00 × 2.00
Part5	20	0.16	2.81 × 0.56 × 0.20
Part6	20	0.24	0.45 × 0.51 × 2.50
Stick2	12	0.18	2.00 × 0.30 × 0.30
Thin	48	1.25	1.00 × 3.00 × 3.50

Table 2: The Ikonen data set.

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

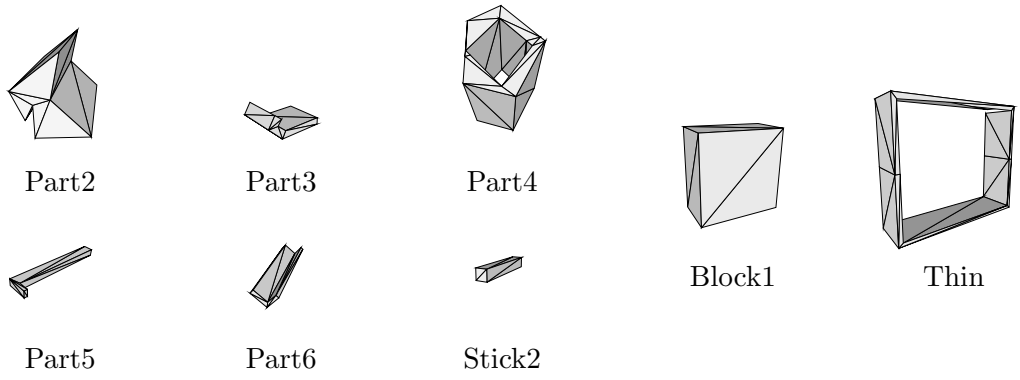


Figure 13: The Ikonen data set.

- Case 1
  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.

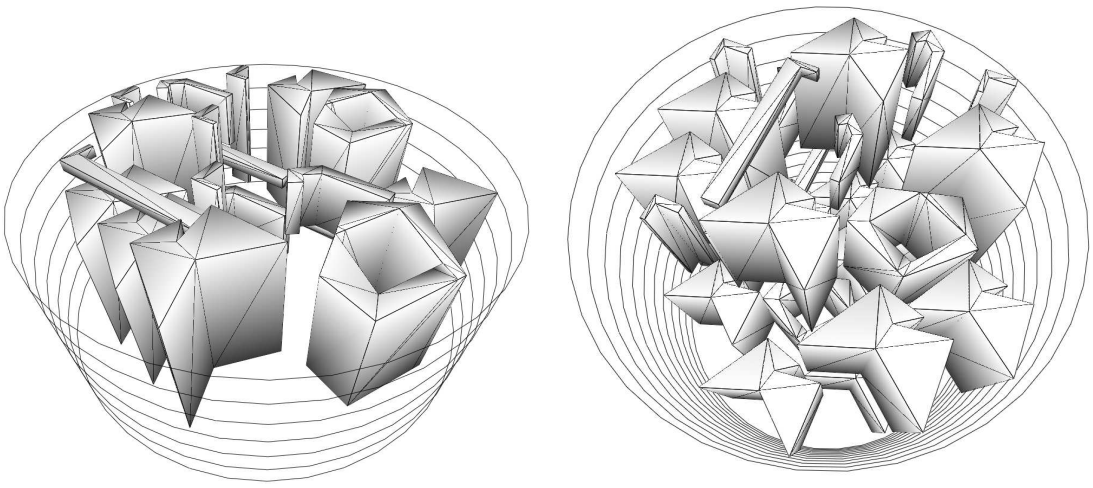
Our initial placement is a random placement which could be a problem since it would quite likely contain almost no overlap and then it would not say much about our algorithm — especially the GLS part. To make the two cases a bit harder we *doubled* the number of objects. Our tests were run on a 733MHz 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



Case 1 Case 2

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

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.

[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(1):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(1):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(3):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. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 2006. In Press.

[37] X. Yan and P. Gu. A review of rapid prototyping technologies and systems. *Computer Aided Design*, 28(4):307–318, 1996.

Published in *Theory of Computing 2007*
(Proceedings of the Thirteenth Computing: The Australasian Theory Symposium)

An efficient solution method for relaxed variants of the nesting problem

Benny K. Nielsen*

Abstract

Given a set of irregular shapes, the strip nesting problem is the problem of packing the shapes within a rectangular strip of material such that the strip length is minimized, or equivalently the utilization of material is maximized. If the packing found is to be repeated, e.g., on a roll of fabric or a coil of metal, then the separation between repeats is going to be a straight line. This constraint can be relaxed by only requiring that the packing produced can be repeated without overlap. Instead of minimizing strip length one minimizes the periodicity of these repeats.

We describe how to extend a previously published solution method [6]$^{\mathrm{A}}$ for the nesting problem such that it can also handle the relaxation above. Furthermore, we examine the potential of the relaxed variant of the strip packing problem by making computational experiments on a set of benchmark instances from the garment industry. These experiments show that considerable improvements in utilization can be obtained.

Keywords: Cutting, packing, irregular strip packing, lattice packing, nesting

1 Introduction

The nesting problem generally refers to the problem of placing a number of shapes within in the bounds of some material, typically rectangular, such that no pair of shapes overlap. Most often, it is also an objective to minimize the size of the material which is equivalent to maximizing the utilization of the material. This problem is also known as the *strip nesting problem* or the *irregular strip packing problem*. Both two- and three-dimensional applications of the nesting problem can be found within a large number of industries. For example, the two-dimensional problem arises when cutting parts of clothes from a roll of fabric. In this case one needs to maximize the utilization of the fabric or equivalently minimize the waste of fabric. This is done by finding the shortest strip of fabric necessary to cut all involved parts of clothes. An example of a solution to such a nesting problem is given in Figure 1. Other two-dimensional applications include sheet metal, glass and animal hide cutting. An example of an application of three-dimensional nesting can be found in the industry of *rapid prototyping*.

Knapsack or bin packing variants of the nesting problem can also be formulated, but in this paper as well as in the majority of related publications, the strip packing variant is the main focus. We are also going to focus on the two-dimensional problem although the problems and the solution method described can be generalized to three dimensions. In the typology of

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

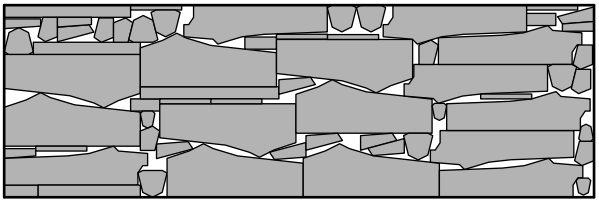


Figure 1: A nesting example of clothing parts for trousers.

Wäscher et al. [20] for cutting and packing problems, we are dealing with a two-dimensional irregular open dimension problem (ODP). The problem is $\mathcal{NP}$-hard even when the shapes and the material involved are rectangles.

In recent years, the strip nesting problem has received a considerable amount of interest. A multitude of solution methods which use various different geometric, heuristic and meta heuristic techniques have been described. Recent surveys are given in the introductions of the papers by Burke et al. [3], Gomes and Oliveira [7], and Egeblad et al. [6]$^{\mathrm{A}}$, and these papers also represent some of the most recent efforts to obtain good solutions for the nesting problem.

This paper concerns a simple variant of the nesting problem based on the following observation. In some cases the solution to a nesting problem is going to be repeated continuously on the material used. For example, consider a nesting solution for a subset of the shapes in Figure 1. If this solution is repeated continuously on a roll of fabric as illustrated in Figure 2a then it becomes clear that the vertical bounds of the rectangles are not really necessary constraints. Relaxing this constraint enlarges the search space and it might be possible to find solutions with better utilization. An example is given in Figure 2b. Relaxing both the vertical and horizontal bounds of the material (see Figure 3) makes the search space even larger. We label these relaxed problem variants *repeated nesting problems*. Note that we picked a subset of the shapes in Figure 1 to better illustrate the idea of repeated patterns, but it might also be a good idea in practice if the nesting problem contains duplicates.

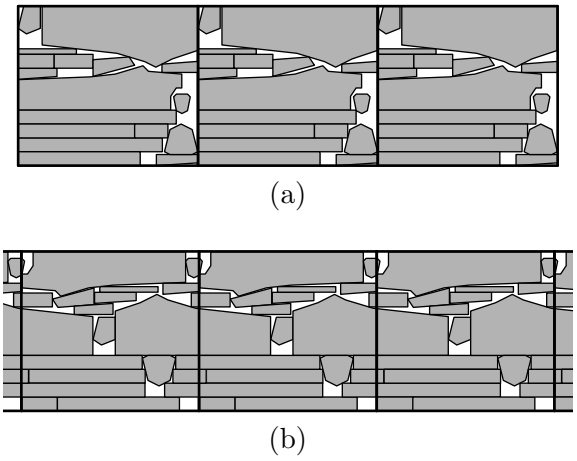
A formal description of repeated nesting problems is given in Section 2 and this is followed by Section 3 which reviews related existing literature. In Sections 4 and 5 we describe our approach to the problem which is an extension of the solution method described by Egeblad et al. [6]$^{\mathrm{A}}$. Computational experiments on benchmark problems from the literature are presented in Section 6. Finally, some concluding remarks are given in Section 7.

2 Problem descriptions

The decision variant of the nesting problem can be formulated as follows.

Nesting Decision Problem. *Given a set of shapes and a piece of material with fixed size, determine whether a non-overlapping placement of the shapes within the bounds of the material exists.*

Now, assuming that the material has a rectangular shape with fixed width and variable length then we can state the following problem.



(a)

(b)

Figure 2: (a) A solution to a strip nesting problem might be repeated continuously on a long strip, e.g., a roll of fabric. (b) A better solution could be obtained if the straight vertical bounds between repeats are relaxed. In this example utilization went up from 88.68% to 89.74% (based on 10 minute runs with the implementation described in Section 6).

Strip Nesting Problem. *Given a set of shapes and a strip of material with width w, find the minimum length l of the material for which the nesting decision problem has a positive solution.*

If the sum of areas of all shapes is denoted A then the utilization of material for a given solution is $A/(l \cdot w)$. Of course, one is not only interested in the minimum value of l (or the maximum utilization value), but also in a corresponding placement of the shapes.

Note that the use of the words width and length is based on the tradition in the (strip) nesting literature. In some cases it can be confusing, e.g., in the drawings, the visual width of a solution is its length and the visual height of a solution is its width.

The problem definitions above use very general terms. Numerous additional constraints can often be added depending on the specific application of the nesting problem. This includes the type of shapes allowed and to what extent rotation or flipping of shapes is allowed. Note that the decision problem is $\mathcal{NP}$-hard even for rectangular shapes (and material) with no rotation allowed. Thus in most cases one has to settle for heuristic solutions.

In this paper, the shapes are polygons with no self intersections. Holes in the polygons are allowed and although in theory rotation could also be allowed [6]$^{\mathrm{A}}$, the current implementation of the solution method presented cannot handle more than a few fixed rotation angles.

The problems dealt with in this paper can be viewed as relaxations of the strip nesting problem. The first relaxation is illustrated in Figure 2b. Here a nesting solution is repeated in one orientation assuming an infinite length of the strip of material. A formal definition of a more general decision problem is as follows.

Repeated Nesting Decision Problem. *Given a set of shapes, a width w and a length l, determine whether a non-overlapping placement of the shapes exists for which any translational copy with offset $(il, jw), i, j \in \mathbb{Z}, (i, j) \neq (0, 0)$, can be made without introducing overlap.*

The values w and l now describe the periods of an infinitely repeated nesting. It is easy to see that such a nesting also has a utilization of $A/(l \cdot w)$. We denote the problem of

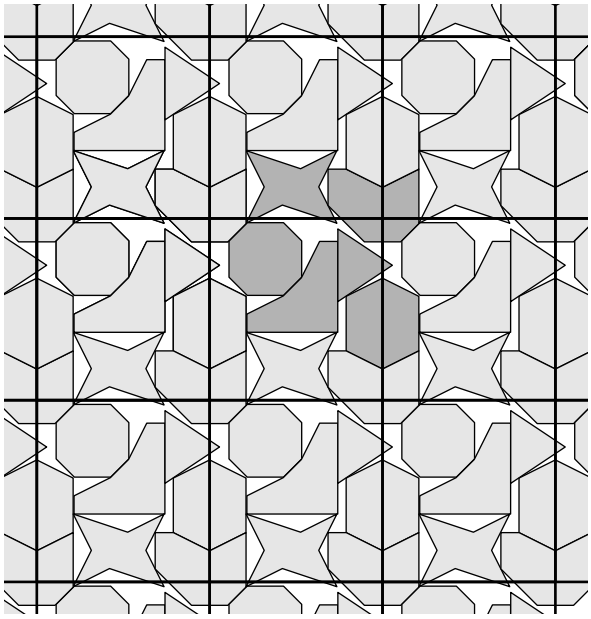


Figure 3: Six polygonal shapes are nested such that they can be efficiently repeated both horizontally and vertically.

minimizing $w \cdot l$, the *repeated (pattern) nesting problem*, and if w is a fixed value and repeats in the orientation of w is not allowed, we denote it the *repeated strip nesting problem* (as in Figure 2b).

An interesting alternative visual interpretation of the repeated strip nesting problem is as follows. Given a set of polygons and a cylinder with fixed height, find the minimum diameter for the cylinder such that a non-overlapping placement of the polygons on the cylinder wall exists.

3 Related work

To the best of our knowledge, the problems described above have not previously been described or solved in the existing literature. Therefore, the following is a survey of papers describing solution methods for problems involving some kind of repeated patterns. Note that this includes a more general problem, known as *the densest translational lattice packing of k polygons*, which is described at the end of this section.

Existing literature on nesting problems with repeated patterns has mainly been focused on *point lattice* packings with only one or very few polygons involved. A (point) lattice in two dimensions is the infinite set of points $au + bv, a, b \in \mathbb{Z}$ for two linearly independent vectors u and v. The parallelogram spanned by the vectors u and v is denoted the *fundamental parallelogram*. Now, the *densest translational lattice packing* of a given polygon is defined by a lattice in which a given polygon P can be repeated at every point of the lattice such that no overlap occurs and the area of the fundamental parallelogram, $|u \times v|$, is minimized. This corresponds to maximum utilization of an infinite material.

Given an n-sided *convex* polygon, Mount and Silverman [14] showed that this problem can

be solved to optimality by an algorithm with linear running time. Note that a solution to this problem is also a densest translational packing since Rogers [17] has shown that such a packing is always a lattice packing. Given a non-convex polygon, Stoyan and Patsuk [19] solved the problem using a mathematical model, but the result is an algorithm which runs in $O(n^6)$ time (reduced to $O(n^3)$ given a convex polygon). A mathematical model including the constraints of a fixed size rectangular material is presented by Stoyan and Pankratov [18], but it is a simplified mathematical model and the solutions found for this model are not guaranteed to be optimal. Other heuristic approaches for a non-convex polygon which include the constraints of a rectangular material are described by Cheng and Rao [4] and Gomes et al. [8].

A slightly more general problem is to find the *densest double-lattice packing*. A double-lattice packing is the union of two lattice packings where one lattice is used for the polygon P and the other one is used for a 180 degree rotation of P. Kuperberg and Kuperberg [10] showed that a densest double-lattice packing of a convex polygon can be constructed by finding a minimum area *extensive parallelogram* within P. Mount [13] then used this fact to describe an algorithm which can solve the problem in linear time. Kuperberg and Kuperberg [10] also proved that such a packing would have a density of at least $\sqrt{3}/2 \approx 86.6\%$. Double-lattice packings for a non-convex polygon are also handled by the mathematical model presented by Stoyan and Pankratov [18] (which includes a rectangular material), but again, optimal solutions are not guaranteed. Gomes et al. [8] handle double-lattice packings by first pairing P and a 180 degree copy, and then use their lattice packing heuristic on the merged polygon.

Much earlier work by Dori and Ben-Bassat [5] represents an interesting heuristic approach to the problem of packing a single convex polygon in the plane. They search for a small convex polygon which can both be used to *pave* the plane and to circumscribe P. Numerous possible pavers exist (corresponding to tilings of the plane) and they could include various rotations of P — unfortunately the authors restrict their search to a single type of paver which would make any solution found equivalent to a double-lattice packing. For this problem, the linear time algorithm by Mount [13] is guaranteed to find optimal solutions. An approach allowing several rotation angles of a non-convex polygon is described by Jain et al. [9]. A fixed number of copies are rotated and translated in a stepwise search of good solutions which can be repeated in one direction (packing on a strip). During the search, overlap is allowed and to avoid local minima the search is guided by the meta heuristic technique *simulated annealing*. Experiments are only done with 2 and 3 copies of the given polygon. The best solution is with 2 copies where one is a 180 degree variant of the other. This is equivalent to a one-dimensional double-lattice.

Few papers handle problems (with repeated patterns) which include several different polygons and some handle such problems by reducing them to problems only involving one polygon, e.g., Cheng and Rao [4]. The first problem is then to cluster the polygons and to describe this cluster by a single polygon. Afterwards any of the solution methods described above (handling a non-convex polygon) can be used. The hard part is to make a cluster which is both tightly packed and well-suited for repetition.

Milenkovic [12] has a more direct approach to the problem of handling several polygons in which a mathematical model is used to search for the densest translational lattice packing of k polygons. A solution to this problem can be described as the union of k lattice packings, one for each polygon, all using the same fundamental parallelogram. Note that unlike translational double-lattice packing, no rotation is involved. Milenkovic first reduces the problem by fixing the area of the fundamental parallelogram. A series of decision problems with decreasing area is then solved when trying to minimize the area. A decision problem can then be stated as

follows. Given a fixed area α, find a lattice defined by vectors u and v and a set of translation vectors for all polygons, such that $|u \times v| = \alpha$ and no polygons overlap when copied according to the lattice. Milenkovic has also implemented his algorithm and results are presented which handle up to 4 polygons.

The densest translational lattice packing of k polygons [12] is the problem that best resembles the problems examined in this paper. The repeated nesting problem as described in the previous section requires the two vectors u and v to be parallel to the x- and y-axes and the repeated strip nesting problem fixes the length of one of the vectors, i.e., the one corresponding to the strip width.

4 Solution method

Some of the best results obtained for the nesting problem have been reported by Egeblad et al. [6]$^{\mathrm{A}}$. In the following we give an outline of their approach and discuss how to extend it such that it can also handle repeated patterns.

First of all, the problem of minimizing the strip length is handled separately from the problem of nesting the shapes given a fixed strip length. An initial strip length is found by using a fast placement heuristic of the shapes. The length is then reduced such that the remaining area of the strip is reduced by some fixed percentage. Now the search for a nesting solution is initiated, and if one is found, the strip length is again reduced. If it takes too long to find a solution then a smaller reduction is attempted.

The strip nesting problem has now been reduced to a nesting decision problem. The shapes are either given as polygons or alternatively approximated by polygons and thus, geometrically, the problem is to nest (or pack) a set of polygons within the bounds of a rectangle.

Initially, all polygons are simply placed within the bounds of the rectangle. Most likely, several pairs of polygons overlap and thus the goal is to reduce this overlap iteratively until no overlap exists — corresponding to a solution to the nesting decision problem. The objective function value for this minimization problem is the total amount of overlap in the placement. A local search scheme is used with a neighborhood consisting of horizontal and vertical translations of each polygon. Now, the major strength of this approach is the existence of a very efficient algorithm for finding a minimum overlap horizontal or vertical translation of a given polygon.

The local search reaches a local minimum when no polygon can be translated vertically or horizontally with a decrease in the total amount of overlap. Whenever such a minimum is not zero (which is the known global minimum that we are looking for) the search needs to be diversified. The meta-heuristic *guided local search* is very well suited for this purpose. In short, the strategy is as follows. Given a local minimum placement of polygons, the "worst" overlap between a pair of polygons is punished (possibly more than one pair is punished) by adding a penalty to the objective function whenever these polygons overlap. This increases the value of the objective function at the local minimum. At some point it is no longer going to be a local minimum and the local search can continue. More details are described by Egeblad et al. [6]$^{\mathrm{A}}$.

To be able to handle repeated patterns we have to extend this solution method. It turns out that the only major change needed is a generalization of the translation algorithm. It needs to be able to take into account that the current placement wraps around at the ends.

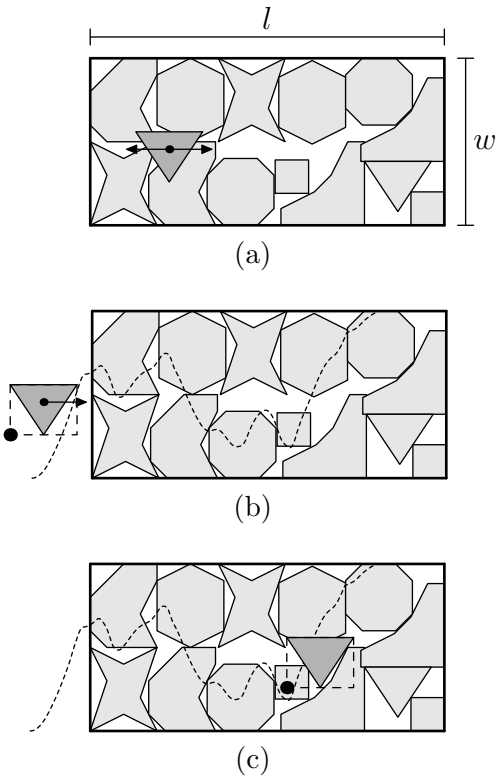


Figure 4: Translation of a polygon (triangle) in a bounded rectangle. (a) The triangle has been selected for translation and should be moved horizontally to minimize its overlap with the other polygons. (b) The triangle is moved outside the container. The dashed curve illustrates the area of overlap function for which we need to find the minimum. (c) Finally, the triangle is moved to the position corresponding to minimum overlap.

This is the subject of the following section.

5 Polygon translation with repeats

We need to extend the bounded translation algorithm given by Egeblad et al. [6]$^{\mathrm{A}}$ to handle repeated patterns. Without loss of generality we can assume that we are looking for a minimum overlap *horizontal* translation of a given polygon. In the following, we first describe the bounded translation algorithm (with a high-level view) and afterwards we show how it can easily be extended to handle repeated patterns. Note that no changes are needed concerning the low-level details of the algorithm.

In the following we use a very general notion of a *polygon*. Shortly, it can be described as a union of non-intersecting simple polygons for which holes are allowed.

Formally, the bounded (horizontal) translation algorithm solves the following problem.

Bounded Horizontal Translation Problem. *Given a polygon P with a fixed vertical offset y and a polygon Q with a fixed position, find a horizontal offset x for P within some given range $[x_1, x_2]$, such that the area of overlap between P and Q is minimized.*

An example is given in Figure 4a. The emphasized triangle is the polygon P and the union of the remaining polygons is Q. The goal is to find a horizontal position of P which minimizes its area of overlap with all of the other polygons. The lower left corner of a bounding box for P is the reference point when offsetting the polygon, thus the left-most position of P is $x = 0$. If w_P is the width of the polygon P then the range of values for x, that places P inside the rectangle, is $[x_1, x_2] = [0, l - w_P]$.

Intuitively, the translation algorithm first places P to the left of Q (see Figure 4b) and then iteratively moves it to the right while calculating a function which measures the overlap between P and Q. This is a piecewise quadratic function and it is drawn in Figure 4b where the x-coordinate is the position of the reference point of P and the y-coordinate is the amount of overlap. The latter value is scaled such that the maximum is at the top of the rectangle. In Figure 4c the triangle has been moved horizontally to its minimum overlap position.

If P contains n edges and Q contains m edges then this problem can be solved in $O(nm \log nm)$ time [6]$^{\text{A}}$. Note that most often $n \ll m$. If the number of edges in P is bounded by some constant then the running time is $O(m \log m)$.

The problems examined in this paper are slightly different and we need to solve a slight variation of the bounded translation problem.

Periodic Horizontal Translation Problem. *Given a polygon P with a fixed vertical offset y and an infinite set of copies of a polygon Q at offsets $(i \cdot l, y), i \in \mathbb{Z}$, find a horizontal offset x within the range $[0, l[$ such that the area of overlap between P and the copies of Q is minimized.*

The range given is just one period of the repeated polygon Q. Given a solution x^*, identical solutions are given at any offset $x^* + i \cdot l, i \in \mathbb{Z}$. Note that P is also going to be repeated at each $x^* + i \cdot l$ and thus it is important that it cannot overlap itself. A simple check can ensure this (proof not given): If there is no overlap between P at offset 0 and a copy at horizontal offset l then no pair of copies of P at offsets $i \cdot l$ and $j \cdot l, i \neq j$, can overlap.

Now, the key observation to be able to extend our bounded translation algorithm is that to correctly evaluate the overlap between P and the copies of Q we only need a finite subset of the copies. In most cases only one or two copies. Assume that the width of Q is w_Q and that its minimum positive offset is at o_Q (horizontally). Given that the width of P is w_P then the following copies of Q are sufficient (maybe not necessary) to guarantee correct overlap calculations in the range $[0, l[$.

$$(i \cdot l, y) \text{ for all } i < 0 : i \cdot l + o_Q + w_Q > 0 \tag{1}$$

$$\text{for all } i \geq 0 : l + w_P > i \cdot l + o_Q \tag{2}$$

Condition (1) includes any copies of Q which has an offset to the left of the range $[0, l[$ while still overlapping the range and condition (2) includes any copies of Q with a positive offset which P might overlap with an offset inside the range. Clearly, the copy of Q with an offset within the range $[0, l[$ is always included by condition (2).

In the above, only one fixed polygon was handled, but if it represents a set of polygons then they could be handled individually when checking the conditions. An example of the periodic translation problem is given in Figure 5a. Again the triangle is moved to the left of the fixed polygons, but it is also moved to the left of any copies of the fixed polygons which can affect overlap calculations while inside the interval $[0, l]$ (condition (1) above). The

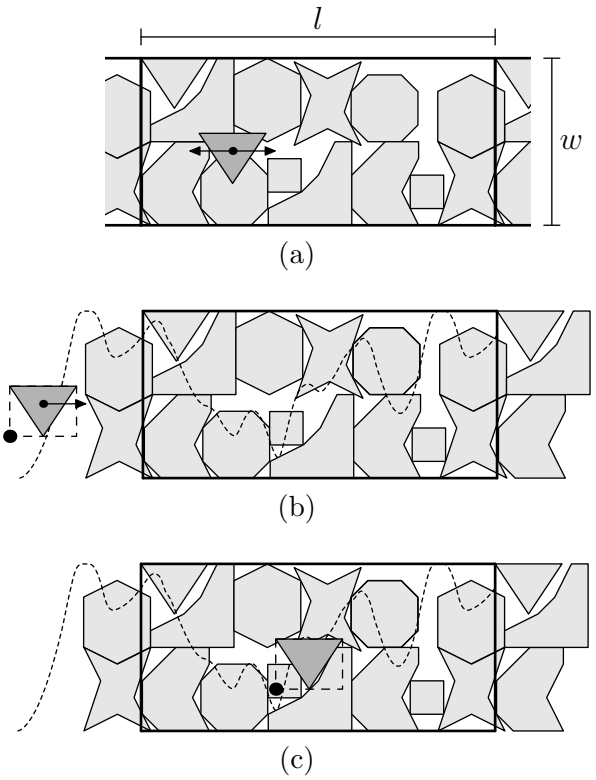


Figure 5: Illustration of the translation of a polygon within a horizontally repeated pattern. (a) The triangle has been selected for translation. (b) The triangle is moved to the left of the material *and* to the left of any polygons needed to get correct overlap calculations inside the material. Polygons are also included to the right of the container. (c) The triangle is moved to the position corresponding to minimum overlap

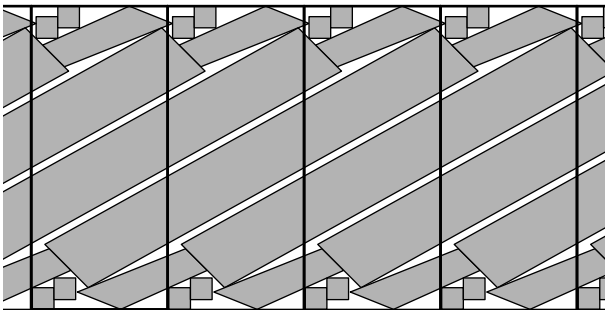


Figure 6: An example showing that a large number of copies of a given polygon can be necessary. Note the single large polygon which stretches four repeats.

overlap area function is also calculated for the full range which can require extra copies to the right of the maximum x-coordinate l (condition (2) above). Ensuring these copies is more or less all that is required to handle repeated patterns. In Figure 5b the triangle is moved to the left and the area of overlap function is calculated as before (and drawn in the figure). Finally, in Figure 5c, the polygon is moved to its new minimum overlap position.

The conditions for copies of Q may look overly complicated, but it is necessary since one could be dealing with horizontally very long (and tilted) polygons both regarding P and Q. An extreme example is given in Figure 6 in which a single polygon stretches four repeats.

A few issues remain to be discussed. 1) The description above does not include the penalties used by the guided local search. These are constants which are added to the quadratic overlap function whenever penalized pairs of polygons overlap. This requires some bookkeeping, but it does not affect the worst case running time and no special changes are needed to handle repeated patterns. 2) If both horizontal and vertical repeats are allowed then the copies of Q described are not sufficient since a horizontally translated polygon could overlap with several vertical copies of Q. This is simply handled by adding more copies of Q before translation using conditions similar to those already described. 3) We have not considered what to do with the ends of a roll (or the borders of a plate). It is assumed that the number of repetitions makes this additional waste negligible, but it is possible to solve a special nesting problem to minimize this additional waste. For example, one could treat the final placement as one big polygon (possibly simplifying it) and then solve the normal strip nesting problem using this polygon and all of the original polygons. The big polygon would only be able to move horizontally and the rest would be placed to its left and to its right side within the bounds of the rectangular material.

6 Computational experiments

The changes needed to handle repeated patterns have been implemented in C++ and incorporated in the nesting program 2DNest [6][A]. All experiments are done on a machine with a 3GHz Pentium 4 processor.

A set of problem instances has been selected from the literature and downloaded from the ESICUP homepage[1]. All of these instances are taken from the garment industry and thus

[1] `http://www.fe.up.pt/esicup`

Problem instance	Number of shapes	Number of unique shapes	Average number of vertices	Strip width	Width ratio	Origin
Albano	24	8	7.25	4900	0.56	Albano and Sappupo [1]
Dagli	30	10	6.30	60	1.18	Ratanapan and Dagli [16]
Mao	20	9	9.22	2550	1.73	Bounsaythip and Maouche [2]
Marques	24	8	7.37	104	1.50	Marques et al. [11]
Shirts	99	8	6.63	40	0.74	Oliveira et al. [15]
Trousers	48	17	5.06	79	0.36	Oliveira et al. [15]

Table 1: Problem instances used in the experiments. The width ratio is the ratio between the width of the problem instance and the length of a solution with 100% utilization.

Problem	No repeats		1D repeat			2D repeat			Waste reduction	
instance	Average	Max	Average	Impr	Max	Average	Impr	Max.	1D	2D
Albano	85.02 (0.6)	86.43	85.80 (0.4)	0.78	86.48	86.20 (0.4)	1.18	86.73	5.2	7.9
Dagli	82.05 (0.5)	82.88	83.04 (0.6)	0.99	84.13	83.06 (0.3)	1.01	83.56	5.5	5.6
Mao	78.91 (0.6)	79.54	80.11 (0.5)	1.20	81.02	80.73 (0.4)	1.82	81.54	5.7	8.6
Marques	86.90 (0.4)	87.51	87.05 (0.4)	0.15	87.50	87.38 (0.2)	0.48	87.50	1.1	3.7
Shirts	84.21 (0.3)	84.88	84.94 (0.1)	0.73	85.14	85.19 (0.2)	0.98	85.74	4.6	6.2
Trousers	85.21 (0.3)	85.84	85.60 (0.2)	0.39	86.02	85.98 (0.3)	0.77	86.26	2.6	5.2

Table 2: Results of experiments with three different problems. All values are based on 10 runs of 10 minutes with different seeds. The three problems are strip nesting with no repeats, strip nesting with horizontal repeats (1D) and strip nesting with both horizontal and vertical repeats (2D). Average utilization and maximum utilization are reported for all problems. The improvement in percentage is reported for the two latter problems in comparison with the first problem. The two rightmost columns are the waste reduction in percent (based on the same comparison).

the results presented should indicate the potential of repeated patterns in realistic settings. Characteristics of these instances can be seen in Table 1. The number of polygons vary between 20 and 99, but the number of unique polygons only vary between 8 and 17. The width ratio is the ratio between the width and the length of a solution with 100% utilization (which most likely does not exist). This value indicates how the rectangle of a solution is going to be shaped. Note that one could easily construct problem instances which would see considerable improvements when allowed to repeat patterns. The problem shown in Figure 6 is one example.

Although 2DNest can work with rotation angles of 90, 180 and 270 degrees, all experiments have been done without rotation. This also means that the results are not directly comparable with results from papers on the strip nesting problem. For this purpose, see the results of 2DNest reported by Egeblad et al. [6]^{A}. Here we focus on the improvements obtained by repeated patterns only.

The main results are presented in Table 2. Each instance has first been solved as a standard nesting problem. The average utilization of 10 runs with random seeds are given and these

are supplemented with the standard deviation and the maximum utilization found. The same instances have then been solved allowing either horizontal repeats only or allowing both horizontal and vertical repeats. Again, average utilization, standard deviation and maximum utilization are presented in the table. Furthermore, the improvement in percentage points is given. In all cases only the strip length is minimized, i.e., even when allowing both horizontal and vertical repeats the width is still fixed. The best solutions found for the problem with horizontal repeats are shown in Figure 7.

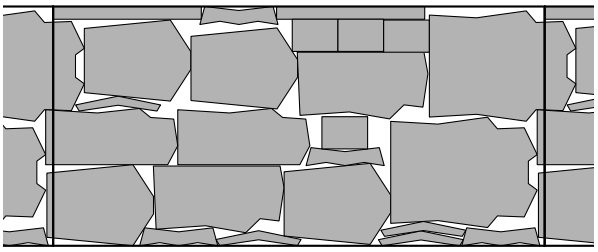
Focusing on the results for horizontal repeats, the improvements vary from only 0.15 up to 1.20 percentage points. This may not seem like a lot, but if one considers the amount of waste in the solutions then these numbers correspond to a 1.1% and 5.7% decrease in waste (second to last column in Table 2). Note also that the best improvements are obtained for the instances with the lowest utilization and vice versa. This indicates that the effect of repeated patterns is greater for "hard" instances. When repeating the pattern both horizontally and vertically the waste reduction is between 3.7% and 8.6%. It would be a reasonably conjecture that small problem instances are better candidates for improvements. This conjecture is supported by the large improvements for the small instances, Albano (24), Mao (30) and Dagli (20), but it is not supported by Marques (24) which is not really improved and Shirts (99) which is more or less improved just as much as the small instances. Nevertheless, given instances that are sufficiently large in the number of polygons involved, the advantage of repeated patterns should decrease.

Since we do not find the optimal solutions we cannot really know if the improvements are caused by the existence of better solutions in the larger search spaces, or are caused by a more efficient search for good solutions. It is worth noting though that the search for repeated patterns involves fewer steps than it does for non-repeated patterns. The worst case running time of the translation algorithm did not change when applied to repeated patterns, but it is slower by some constant factor. This is emphasized by the results presented in Table 3. The average number of translations performed per second is given for all three nesting problems. Small instances are punished more than large instances (most likely because relatively more copies of polygons are needed in each translation), but the results are quite similar. On average the number of translations performed decreases by 21% when making horizontal repeats only and by 33% when also making vertical repeats. This decrease in the number of translations is important to consider when analyzing the results in Table 2.

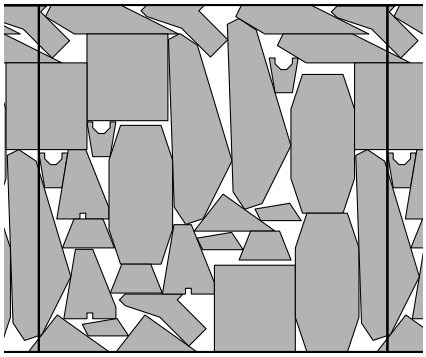
7 Conclusion

We have presented an efficient heuristic solution method which can construct very good solutions for strip nesting problems in which the solution is going to be repeated either horizontally or both horizontally and vertically. Results are given for fairly large instances and they strongly indicate that this can give a considerable reduction of waste for problem instances in the garment industry. The solution method is an extension of the work of Egeblad et al. [6]^{A} and the main difference is an altered algorithm for finding a minimum overlap horizontal or vertical position of a given polygon.

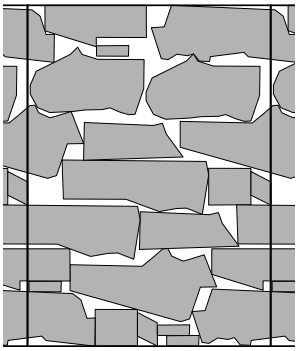
Numerous problems remain for future research including free rotation, border filling and a generalization to three dimensions. It should also be possible to alter the solution method such that it applies to the more general translational lattice packing problem for k polygons [12]. This would involve varying both width and length of the solution, but more importantly it



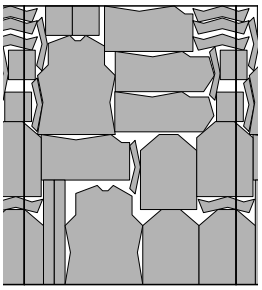
Albano (util.: 86.48%, length: 10066.06)



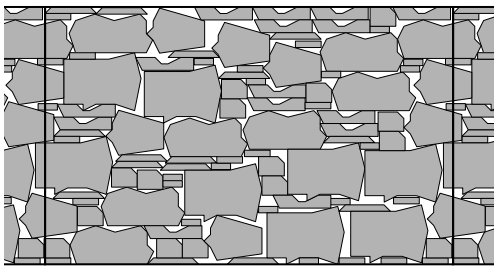
Dagli (util.: 84.13%, length: 60.28)



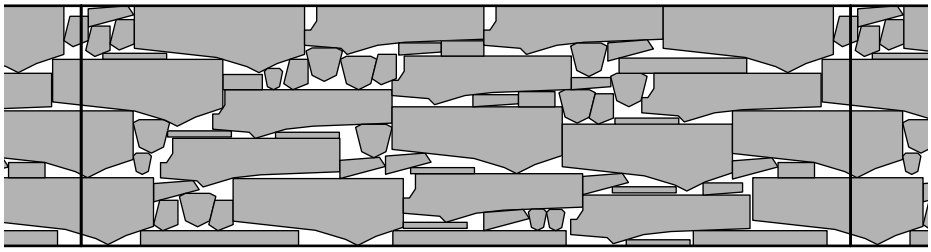
Mao (util.: 81.02%, length: 1819.20)



Marques (util.: 87.50%, length: 79.05)



Shirts (util.: 85.14%, length: 63.42)



Trousers (util.: 86.02%, length: 253.19)

Figure 7: Best results for the repeated strip nesting problem.

Problem instance	Translations per second			Decrease in percent	
	No	1D	2D	1D	2D
Albano	2847	2178	1891	23%	34%
Dagli	2808	2235	1856	20%	34%
Mao	1622	1218	982	25%	39%
Marques	2957	2213	1882	25%	36%
Shirts	1613	1385	1223	14%	24%
Trousers	2010	1658	1443	18%	28%
Average	2310	1815	1546	21%	33%

Table 3: The average number of translations performed within the running time of 10 minutes. Fewer translations are done when searching for solutions with repeated patterns. The two rightmost columns describe the decrease in percent.

would be necessary to repeat the solution in arbitrary directions. Another interesting idea is to extend the problem to other pavers than the parallelogram used in lattice packings. For example, it might be possible to produce solutions using a hexagonal paver.

Acknowledgments

The author would like to thank Jens Egeblad and Allan Odgaard for allowing the changes to 2DNest which were needed to produce the results presented in this paper.

References

[1] 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.

[2] C. Bounsaythip and S. Maouche. Irregular shape nesting and placing with evolutionary approach. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 4, pages 3425–3430, New York, 1997. IEEE.

[3] 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(1):27–49, 2007.

[4] S. K. Cheng and K. P. Rao. Large-scale nesting of irregular patterns using compact neighborhood algorithm. *Journal of Materials Processing Technology*, 103(1):135–140, 2000.

[5] D. Dori and M. Ben-Bassat. Efficient nesting of congruent convex figures. *Commun. ACM*, 27 (3):228–235, 1984.

[6] 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.

[7] 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.

[8] A. M. Gomes, M. T. Costa, and J. F. Oliveira. An iterated local search approach to the large-scale regular packing of irregular shapes in a limited sheet. In *The 6th Metaheuristics International Conference*, Vienna, Austria, August 2005.

[9] P. Jain, P. Fenyes, and R. Richter. Optimal blank nesting using simulated annealing. *Journal of Mechanical Design*, 114(1):160–165, 1992.

[10] G. Kuperberg and W. Kuperberg. Double-lattice packings of convex bodies in the plane. *Discrete and Computational Geometry*, 5(1):389–397, 1990.

[11] V. M. M. Marques, C. F. G. Bispo, and J. J. S. Sentieiro. A system for the compaction of two-dimensional irregular shapes based on simulated annealing. In *International Conference on Industrial Electronics, Control and Instrumentation, IECON '91*, pages 1911–1916, Kobe, Japan, 1991.

[12] V. J. Milenkovic. Densest translational lattice packing of non-convex polygons. *Computational Geometry*, 22:205–222, 2002.

[13] D. M. Mount. The densest double-lattice packing of a convex polygon. In J. E. Goodman, R. Pollack, and W. Steiger, editors, *Discrete and Computational Geometry: Papers from the DIMACS Special Year*, volume 6 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 245–262. Amer. Math. Soc., 1991.

[14] D. M. Mount and R. Silverman. Packing and covering the plane with translates of a convex polygon. *Journal of Algorithms*, 11(4):564–580, 1990.

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

[16] K. Ratanapan and C. H. Dagli. An object-based evolutionary algorithm for solving irregular nesting problems. In *Proc. Artificial Neural Networks in Engrg. (ANNIE'97)*, volume 7, pages 383–388, New York, 1997. ASME Press.

[17] C. Rogers. The closest packing of convex two-dimensional domains. *Acta Mathematica*, 86(1): 309–321, 1964.

[18] Y. G. Stoyan and A. V. Pankratov. Regular packing of congruent polygons on the rectangular sheet. *European Journal of Operational Research*, 113(3):653–675, 1999.

[19] Y. G. Stoyan and V. N. Patsuk. A method of optimal lattice packing of congruent oriented polygons in the plane. *European Journal of Operational Research*, 124(1):204–216, 2000.

[20] G. Wäscher, H. Haussner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 2006. In Press.

Submitted

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 polytopes with n and m facets 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. Container loading is generally 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*

*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.

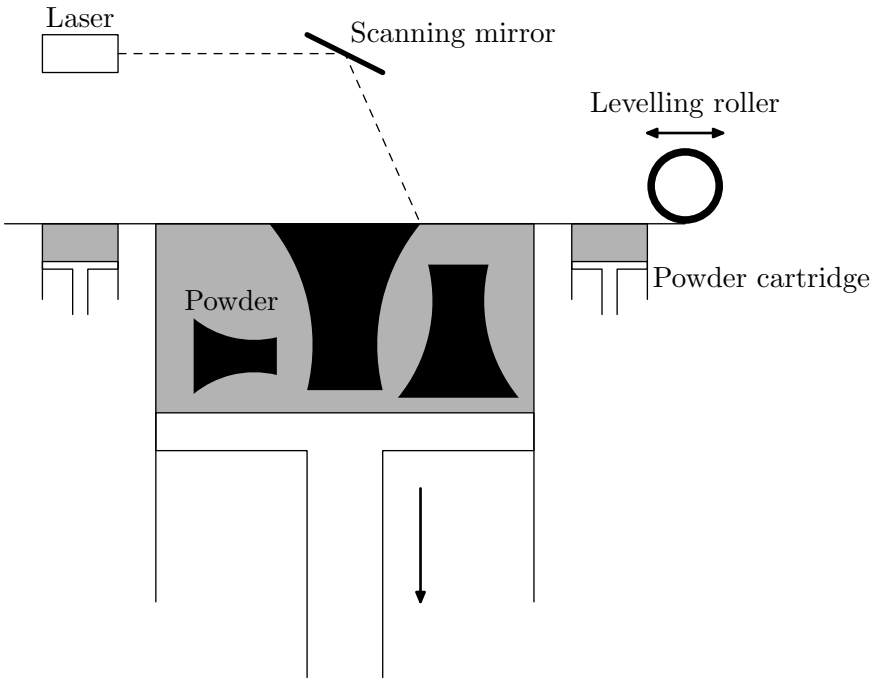


Figure 1: An illustration of a typical machine for rapid prototyping. The powder is added one layer at a time and the laser is used to sinter what should be solidified to produce the desired objects.

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 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 pack as many objects as possible into one run. A survey of rapid prototyping technologies is given by Yan and Gu [29].

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. [11]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 [15]. 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 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 will limit our description here 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. [28], 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 to the problem 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 [26] and a survey of the 2D packing literature concerning irregular shapes (nesting) is given by Dowsland and Dowsland [10]. Recent heuristic methods for orthogonal packing problems include the work of Lodi et al. [21] and Faroe et al. [14] for the bin-packing problem and Bortfeldt et al. [3] and Eley [13] 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. [5] 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 to determine intersections of shapes. A survey on *3D free form packing and cutting problems* is given by Osogami [24]. 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. [19] 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. [4] use the meta-heuristic *simulated annealing* and they also allow rotation. They can also handle various additional optimization objectives such as routing lengths. Intersection checks are done using *octree decompositions* of shapes. 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). At the top level 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. To check overlap of two shapes one can test the cubes of the associated octrees. 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. 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 [30, 31].

Dickinson and Knopf [8] 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 [9] 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. [18] 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. [18] 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. [12] 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. [25] 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. [25] 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. Although our solution method is heuristic, 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 instead. Note that 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. [11]$^{\mathrm{A}}$ 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, satisfying 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).

Following standard notation, we refer to a one-dimensional face of P as an *edge*, and a zero-dimensional face as a *vertex*. Note that our definition of a polytope differs from some

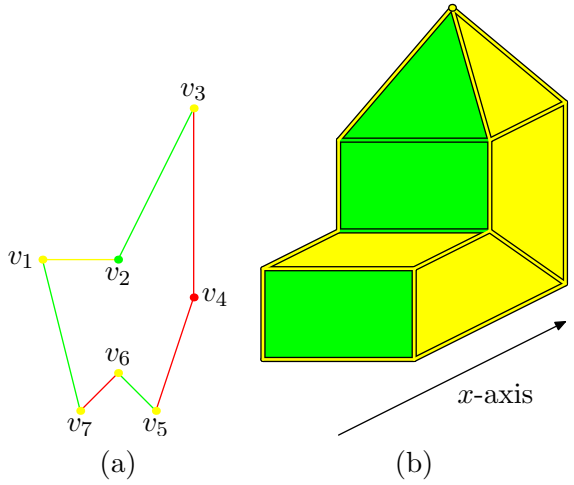


Figure 2: (a) A polygon with three positive (green) edges, (v_1, v_7), (v_2, v_3) and (v_5, v_6), three negative (red) edges, (v_3, v_4), (v_4, v_5) and (v_6, v_7), and one neutral (yellow) edge, (v_1, v_2). Also note that the end-points v_1, v_3, v_5, v_6 and v_7 are neutral, v_2 is positive and v_4 is negative. (b) A polyhedron for which only positive (green) and neutral (yellow) 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.

other definitions in the literature, in that we allow two adjacent facets to lie in the same hyperplane. As mentioned earlier, 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).

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., $\exists \varepsilon > 0 : \forall p' \in \mathbb{R}^d$, where $||p - p'|| < \varepsilon,\ p' \in P$.

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 10 (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*

- *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, given a set of facets F we can divide them into sets of positive facets, F^+, negative facets, F^- and neutral facets, F^0. 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 then it cannot belong to the interior of any other facet and thus it is simply assigned to this facet. 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 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 11 (Facet Count Functions)**.** *Given a set of facets F we define the* facet count functions *for all points $p \in \mathbb{R}^d$ as follows:*

$$\begin{aligned}
\overleftarrow{C}_{F^+}(p) &= |\{f \in F^+ \mid \exists t > 0 : p - te_1 \in f\}|,\\
\overleftarrow{C}_{F^-}(p) &= |\{f \in F^- \mid \exists t \geq 0 : p - te_1 \in f\}|,\\
\overrightarrow{C}_{F^+}(p) &= |\{f \in F^+ \mid \exists t \geq 0 : p + te_1 \in f\}|,\\
\overrightarrow{C}_{F^-}(p) &= |\{f \in F^- \mid \exists t > 0 : p + te_1 \in f\}|.
\end{aligned}$$

The facet count functions, $\overleftarrow{C}_{F^+}(p)$ and $\overleftarrow{C}_{F^-}(p)$, represent the number of times the ray from p with directional vector $-e_1$ intersects a facet from F^+ and F^-, respectively. Equivalently, $\overrightarrow{C}_{F^+}(p)$ and $\overrightarrow{C}_{F^-}(p)$ represent the number of times the ray from p with directional vector e_1 intersects a facet from F^+ and F^-, respectively.

The following lemma states some other important properties of polytopes and their positive/negative facets based on the facet count functions above.

Lemma 2. *Let P be a polytope with facet set F. Given a point p and interval $I \subseteq \mathbb{R}$, we say that the line segment $l_p(t) = p + t \cdot e_1$, $t \in I$,* intersects *a facet $f \in F$ if there exist $t_0 \in I$ such that $l_p(t_0) \in f$.*

Given a balanced assignment of the boundary points of P then all of the following statements hold.

1. *If $I = (-\infty, \infty)$ then, as t increases from $-\infty$, the facets intersected by $l_p(t)$ alternate between positive and negative.*
2. *If $p \notin \text{int}(P)$ then $\overrightarrow{C}_{F^+}(p) = \overrightarrow{C}_{F^-}(p)$, i.e, the ray from p in direction e_1 intersects an equal number of positive and negative facets. Similarly, $\overleftarrow{C}_{F^+}(p) = \overleftarrow{C}_{F^-}(p)$.*
3. *If $p \in \text{int}(P)$ then $\overrightarrow{C}_{F^-}(p) - \overrightarrow{C}_{F^+}(p) = 1$, i.e, the number of positive facets intersected by the ray from p in direction e_1 is one less than the number of negative facets. Similarly, $\overleftarrow{C}_{F^+}(p) - \overleftarrow{C}_{F^-}(p) = 1$.*

The proof is straightforward, and is omitted.

As a corollary, the facet count functions provide an easy way of determining whether or not a given point is in the interior of P. We do not need this to prove the main theorems of this section, but it illustrates the intuitive purpose of the facet count functions.

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.

Definition 12 (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 1. *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:*

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

where

$$w(p) = \mathbf{C}(F^+, G^-, p) + \mathbf{C}(F^-, G^+, p) - \mathbf{C}(F^+, G^+, p) - \mathbf{C}(F^-, G^-, p) \tag{1}$$

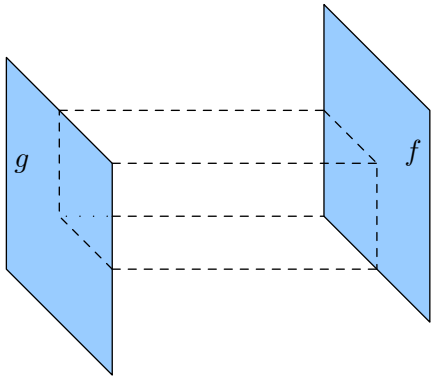


Figure 3: A simple example of the inter-facet region, $R(f,g)$, of two vertical faces f and g in $\mathbb{R}^3$. The dashed lines delimit the region. Note that the inter-facet region $R(g,f)$ is empty.

Proof. Using the facet count functions and Lemma 2, we can, in each special case, verify Equation 1 by counting. If $p \in \text{int}(P \cap Q)$ then the following calculations are needed:

$$\begin{aligned} w(p) &= \overleftarrow{C}_{F^+}(p) \cdot \overrightarrow{C}_{G^-}(p) + \overleftarrow{C}_{F^-}(p) \cdot \overrightarrow{C}_{G^+}(p) - \overleftarrow{C}_{F^+}(p) \cdot \overrightarrow{C}_{G^+}(p) - \overleftarrow{C}_{F^-}(p) \cdot \overrightarrow{C}_{G^-}(p) \\ &= \overleftarrow{C}_{F^+}(p) \cdot (\overrightarrow{C}_{G^-}(p) - \overrightarrow{C}_{G^+}(p)) - \overleftarrow{C}_{F^-}(p) \cdot (\overrightarrow{C}_{G^-}(p) - \overrightarrow{C}_{G^+}(p)) \\ &= \overleftarrow{C}_{F^+}(p) - \overleftarrow{C}_{F^-}(p) = 1. \end{aligned}$$

When $p \notin \text{int}(P \cap Q)$ the counting depend on whether $p \notin P$ and/or $p \notin Q$. The above and the rest of the proof is similar to the proof given for two dimensions by Egeblad et al. [11]$^{\text{A}}$. Replace edge with facet and polygon with polytope. □

In order to solve 1DdDTP we need to define a measure of the overlap between two polytopes.

Definition 13 (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.*

We would like an overlap measure which is computationally efficient and give a reasonable estimate of the degree of overlap of a polytope P with another fixed polytope, as P undergoes a translation in direction e_1. A general study of overlap measures in the context of translational packing algorithms will appear in a forthcoming paper.

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 2. *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 a decomposable overlap measure, and 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 1 and the proof given for the Intersection Area Theorem in Egeblad et al. [11]$^{\mathrm{A}}$, 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 2. 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 1 is 0 and therefore will have no impact on the resulting volume. However, this does not apply to decomposable overlap measures in general.

In order to simplify notation in the following sections, for a d-dimensional region R we will write $V(R)$ for $V^d(R)$; and for given facets f and g we will write $V(f, g)$ for $V^d(f, g)$.

4.2 Minimum area translations

In the following we describe an efficient algorithm for solving 1DdDTP with respect to volume measure, using Theorem 2. 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 2.

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 2, 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.

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 14 (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*

$$\mathrm{proj}_f(g) = \{p \in f | \exists t \in \mathbb{R} : \ p(t) \in g\} \tag{2}$$

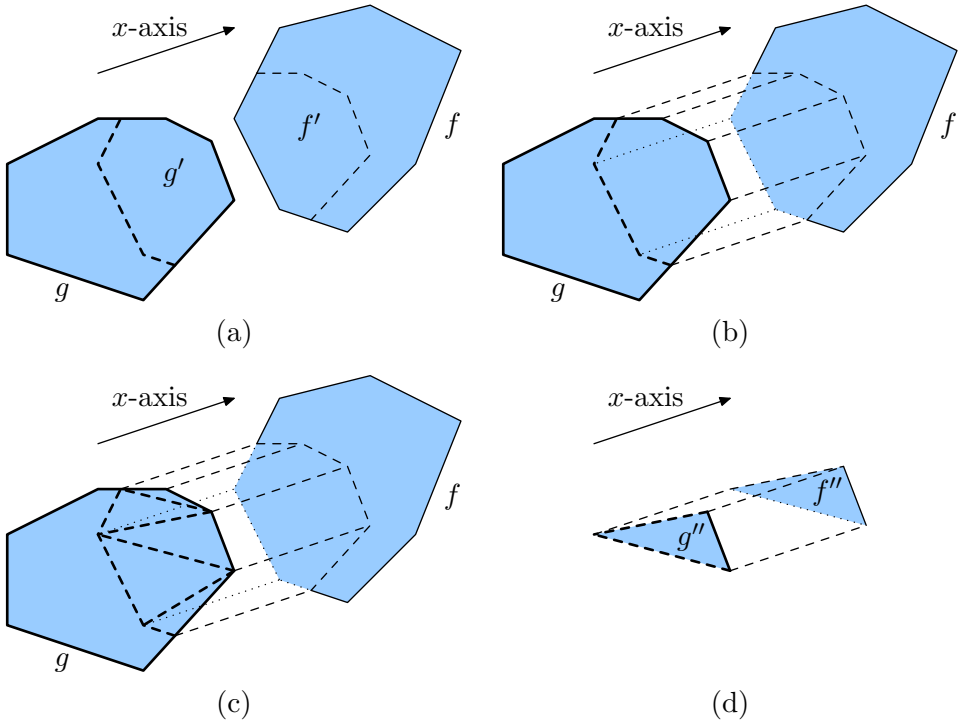


Figure 4: An example of the inter-facet region $R(f, g)$ between two faces, f and g, in three dimensions, where g is in front of f in the x-axis direction. In general the two facets are not necessarily in parallel planes and they can also intersect. (a) The dashed lines indicate the boundary of the projections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

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.

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. Furthermore, in d-dimensional space, we denote a lower dimensional simplex of dimension $d' < d$ as a *d'-simplex*.

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-trivial 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.

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 of g', p_i, 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:

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

This is illustrated in Figure 6.

Since $v_d = (1, 0, \ldots, 0)$, we can simplify the above expression to

$$V_0(f(t), g) = \frac{1}{d!} |\det([v_1, v_2, ..., v_d])t^d| = \frac{1}{d!} |\det([v_1', v_2', ..., v_{d-1}'])t^d|,$$

where v_i' is v_i without the first coordinate. This results in very simple expressions in low dimensions:

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

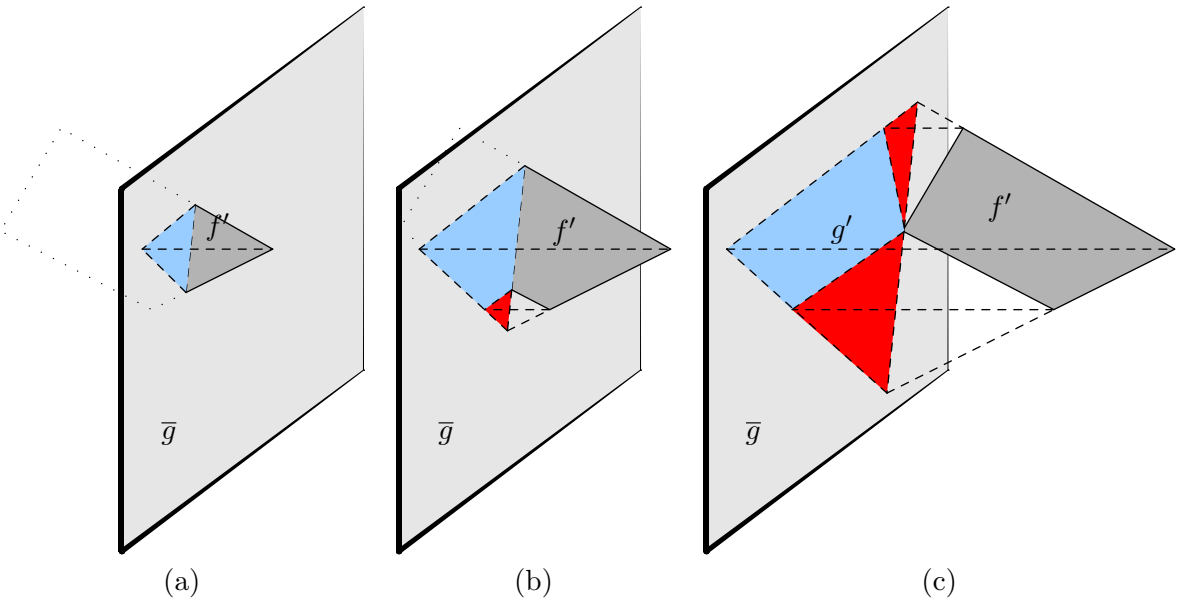


Figure 5: An illustration of the growing tetrahedron volumes needed when a face f' passes through a face g, or equivalently its plane $\overline{g}$, in order to calculate the volume of $R(f', g')$. (a) The initial growing tetrahedron after the first breakpoint. (b) After the second breakpoint, at growing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

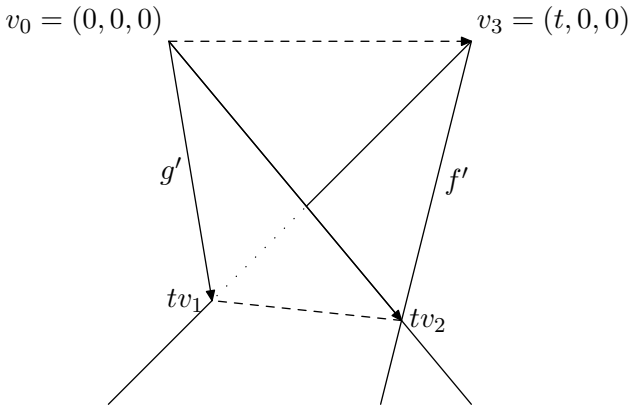


Figure 6: In 3D, it is necessary to calculate the volume function for a growing tetrahedron based on two points, v_1 and v_2, and an overlap distance t. The coordinates of the points change linearly with respect to a change of the distance t. In d dimensions $d-1$ neighboring points are used.

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 [20] 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 [20], we then have

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

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.

The breakpoints are sorted such that their distances are $t_1 \le t_2 \le \ldots \le t_n$. 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 $t \le t_1$, corresponding to P being completely outside the container (overlapping C). As each breakpoint t_i is reached, the algorithm adds an appropriate volume

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 $\nu(t_i')$

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 between succeeding breakpoints, $(t_i, t_{i+1}]$ 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 of breakpoint distances, (t_b^1, t_b^2). 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 $t \in (t_b^1, t_b^2)$. Therefore 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). Given 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 polytopes with n and m facets, the number of breakpoints generated is at most a constant times nm. Computing the volume function 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.

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 it is easy to see that 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.

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. [11]$^{\mathrm{A}}$. 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 [27]. 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.

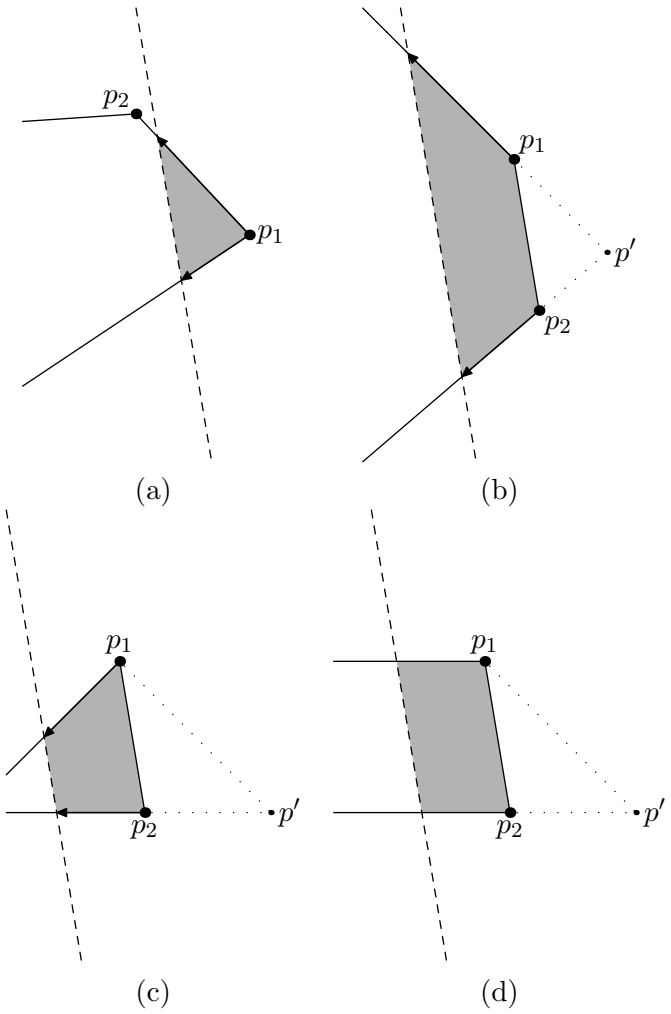


Figure 7: Various special cases can occur when the two first breakpoint distances are equal. Here it is illustrated in 2D using a dashed line to illustrate the intersection line of the two faces during the translation. (a) The standard case in which the first breakpoint distance is unique. (b) The two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

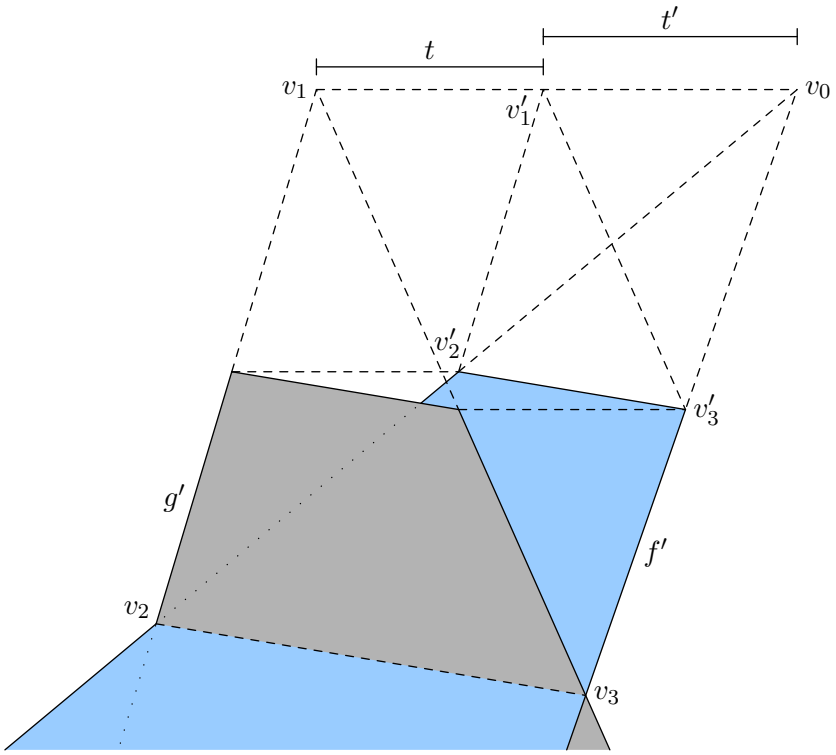


Figure 8: To calculate the volume between the faces f' and g', one can base the calculations on the extended faces illustrated with dashed lines. The volume is a growing tetrahedron (v_0, v_1, v_2, v_3) subtracted by a constant volume (the tetrahedron (v_0, v_1', v_2', v_3')) and subtracted by a linearly growing volume based on the triangle (v_1', v_2', v_3') and the translation distance t.

Note that if $P = Q_i$ is the polytope undergoing translation then the polytope Q (in the translation algorithm in the previous section) is actually the union of all other polytopes, i.e., $Q = \cup_{j=1,\ j\neq i}^{n} Q_j$. The container C in the previous section is assumed to be a simple rectangular parallelepiped with some initial height h. (The solution method can, however, be adapted to other containers.)

Let $V_{i\cap j}(\mathbf{P})$ be the volume of the pairwise overlap of Q_i and Q_j in a placement $\mathbf{P}$. The local search minimizes the objective function

$$f(\mathbf{P}) = \sum_{1\leq i<j\leq n} V_{i\cap j}(\mathbf{P}). \tag{4}$$

In other words, $f(\mathbf{P})$ is the total sum of volumes of pairwise overlaps in placement $\mathbf{P}$. If $f(\mathbf{P}) = 0$ then $\mathbf{P}$ is a solution to the current instance of dDDPP.

The local search uses the axis-aligned translation algorithm from the previous section to iteratively decrease the amount of overlap in the current placement. A local minimum is reached if no polytope can be translated in an axis-aligned direction to a position with less overlap. To escape local minima the objective function is augmented using the principles of GLS to give

$$g(\mathbf{P}) = f(\mathbf{P}) + \lambda \sum_{1\leq i<j\leq n} \phi_{i,j} I_{i,j}(\mathbf{P}), \tag{5}$$

where λ is a penalty constant used to fine-tune the heuristic, $\phi_{i,j}$ is a penalty term associated with Q_i and Q_j (which is described in more detail below) and $I_{i,j}(\mathbf{P}) \in \{0, 1\}$ is 1 if and only if the interiors of polytopes Q_i and Q_j overlap in placement $\mathbf{P}$. Due to the fact that the augmented terms are larger than zero if and only if $\mathbf{P}$ contains overlap, the augmented objective function, $g(\mathbf{P})$, retains the property that a value of 0 is reached if and only if there is no overlap in the placement $\mathbf{P}$. Therefore a placement $\mathbf{P}$, is a solution to the dDDPP if and only if $g(\mathbf{P}) = 0$.

Initially $\phi_{i,j} = 0$ for $1 \leq i < j \leq n$. Whenever the search reaches a local minimum with $g(\mathbf{P}) > 0$, the value of $\phi_{i,j}$ increases for the pair Q_i and Q_j with highest $\mu_{i,j}(\mathbf{P})$ where $\mu_{i,j}(\mathbf{P}) = \frac{V_{i\cap j}(\mathbf{P})}{1+\phi_{i,j}}$. We refer to this as *penalizing* the pair Q_i and Q_j. Intuitively, this change of the objective function $g(\mathbf{P})$ (if large enough), allows the local search to move Q_i and Q_j away from each other, even if it results in greater overlap. Ideally, this move causes the local search to reach a different part of the solution space. To avoid large discrepancy between the real and penalized solution space, the penalties are reset from time to time. To avoid searching the entire neighborhood in each iteration of GLS, we also apply *fast local search* [27].

The translation algorithm in the previous section needs to be able to handle the penalties introduced here. This subject was not fully covered by Egeblad et al. [11]$^{\mathrm{A}}$ although it is quite straightforward. First of all an augmented volume polynomial $\nu'(t)$ is defined by adding the penalties between the polytope $P = Q_i$ to be translated and all other polytopes. This is done by maintaining an array of volume functions $\nu_{Q_j}(t), Q_j \in \mathcal{S} \setminus Q_i$. Whenever the overlap between P and a given polytope Q_j changes from 0 or to 0, the penalty for the pair of polytopes P, Q_j is, respectively, added to or subtracted from the augmented volume function $\nu'(t)$. Note that this does not increase the asymptotic running time, since the volume polynomial of a breakpoint arising from a face of Q_j is only added to $\nu'(t)$ and $\nu_{Q_j}(t)$ and only $\nu_{Q_j}(t)$ needs to be checked for a change to or from 0.

With regard to the usefulness of the local search neighborhood in relation to the number of dimensions d, we note that a 1-dimensional translation becomes a less efficient move as d increases, since up to d axis-aligned translations may be required to move a polytope from one arbitrary point to another. However, it should also be noted that in general fewer polytopes would be involved in each translation. If the polytopes are placed compactly in a grid-like fashion with little overlap, then there are likely to be in the order of $\sqrt[d]{|(S)|}$ polytopes to be considered in each of the coordinate system axes directions.

5.2 Initial solution

The solution method described above can start with a parallelepiped of any height since the initial placement is allowed to contain overlaps. However, it makes more sense set the initial height to one for which a solution is known to exist.

In any dimension, a naive initial height can be based on the sum of heights of all polytopes, but in the following, we describe a more ambitious strategy. In short, we use a greedy bounding box based algorithm in which the polytopes are placed one by one inside the container in an order of decreasing bounding box volume. This algorithm is based on residual volumes and is related to the approach used by Eley [13] for the container loading problem in three dimensions. Although the algorithm could be generalized to higher dimensions, we are only going to describe its three-dimensional variant.

The algorithm maintains a set of empty box-spaces. Each box-space s consists of the volume $[\underline{x}_s, \overline{x}_s] \times [\underline{y}_s, \overline{y}_s] \times [\underline{z}_s, \overline{z}_s]$. Initially the entire container is the only empty space.

Whenever a new shape i with bounding box $B_i = [\underline{x}_i, \overline{x}_i] \times [\underline{y}_i, \overline{y}_i] \times [\underline{z}_i, \overline{z}_i]$ is to be placed inside the container, the list of empty spaces is searched. Let s' be the empty space with lexicographical least $\underline{z}'_s$, $\underline{y}'_s$ and $\underline{x}'_s$ (lower-left-back corner), which is large enough to contain B_i. Shape i is now positioned in s with offset $(x_i, y_i, z_i)^T$ such that B_i's lower-left-back corner is coincident with the lower-left-back corner of s; $(\underline{x}_i, \underline{y}_i, \underline{z}_i)^T + (x_i, y_i, z_i)^T = (\underline{x}'_s, \underline{y}'_s, \underline{z}'_s)^T$.

Next all residual spaces that overlap with the bounding box of the positioned shape i, $B'_i = [\underline{x}_i + x_i, \overline{x}_i + x_i] \times [\underline{y}_i + y_i, \overline{y}_i + y_i] \times [\underline{z}_i + z_i, \overline{z}_i + z_i]$, are split into six new box-spaces and removed from the list of empty box-spaces. For each overlapping space s we generate the following six new box-spaces:

$$[\underline{x}_s, \underline{x}_i + x_i] \times [\underline{y}_s, \overline{y}_s] \times [\underline{z}_s, \overline{z}_s], \quad [\underline{x}_s, \overline{x}_s] \times [\underline{y}_s, \underline{y}_i + y_i] \times [\underline{z}_s, \overline{z}_s], \quad [\underline{x}_s, \overline{x}_s] \times [\underline{y}_s, \overline{y}_s] \times [\underline{z}_s, \underline{z}_i + z_i]$$
$$[\overline{x}_i + x_i, \overline{x}_s] \times [\underline{y}_s, \overline{y}_s] \times [\underline{z}_s, \overline{z}_s], \quad [\underline{x}_s, \overline{x}_s] \times [\overline{y}_i + y_i, \overline{y}_s] \times [\underline{z}_s, \overline{z}_s], \quad [\underline{x}_s, \overline{x}_s] \times [\underline{y}_s, \overline{y}_s] \times [\overline{z}_i + z_i, \overline{z}_s],$$

representing the volumes left, below, behind, right, above and in-front of B_i, respectively. If any of the intervals are empty the new space is empty. Each of the new non-empty spaces are added to the list of empty spaces and may be used for the remaining bounding boxes. To reduce the number of empty spaces generated throughout this process, spaces which are contained within or are equal to other empty spaces are discarded whenever a new bounding box is placed.

The resulting placement is a non-overlapping placement and the maximum $\overline{z}$ value of any placed bounding box B'_i may be used as a basis for the initial strip-height. To diversify solutions to 3DSPP we place shapes randomly within a container with this strip-height. This is the only random element of the solution method.

6 Computational experiments

The solution method described in this paper was implemented for the three dimensional problem using the C++ programming language and the GNU C++ 4.0 compiler. We denote this implementation 3DNest. Although similar in functionality, this implementation is not identical to the one used by Egeblad et al. [11]^{A}. In particular, the new implementation can handle convex faces without triangulating them and it can handle the strip packing problem. Another noteworthy feature of the new implementation is that it is possible to do almost all calculations with rational numbers — the only exception is the computation of minimum values between breakpoints in the translation algorithm since this requires solving quadratic equations. This is primarily convenient for debugging purposes, and is currently not very useful in practice since it is much slower than using standard floating point precision.

Due to the limited precision of floating point calculations, the correctness of all solutions found are verified using Cgal [16], i.e., it is verified that no polyhedron is involved in any significant overlap with other polyhedra or the container. In the experiments presented in this section, the largest total volume of overlap allowed in a solution corresponds to 0.01% of the total volume of all polyhedrons for the given problem.

All experiments were performed on a system with a 2.16 GHz Intel Core Duo processor with 2 MB of level 2 cache and 1 GB of RAM.

6.1 Problem instances

The literature on the subject of three-dimensional packing contains only few useful problem instances with regard to a comparison of results. We have found two appropriate data sets for our experiments. The first one was introduced by Ikonen et al. [19] and the second one was introduced by Stoyan et al. [25]. The sets contain 8 and 7 polyhedra, respectively. Characteristics of these data sets are presented in Table 1. The Stoyan polyhedra are all convex and relatively simple with a maximum of 18 faces, while some of the Ikonen polyhedra are non-convex and feature up to 52 faces. Real world instances, e.g., from the rapid prototyping industry, could easily contain more than 100,000 faces, but in most cases it would also be possible to simplify these polyhedra considerably without making substantial changes to the basic shape, e.g., Cohen et al. [6] has an example of a model of a phone handset which is reduced from 165,936 to 412 triangles without changing its basic shape.

6.2 Puzzles

To further test the capabilities of our solution method, we devised and implemented a generator for random problem instances. The generator creates a problem instance by splitting a three-dimensional cube into smaller pieces. The pieces along with container dimensions matching the width and height of the cube constitute a problem instance for which the optimal utilization is known to be 100%.

A set of half-spaces $\mathcal{H}$ can be used to define a convex polyhedron as the set of points which is contained in all of the half-spaces. In practice, this polyhedron can be found by generating the set, I, of all intersection points of distinct planes p, q, r with $p, q, r \in \mathcal{H}$, and then generate the convex hull of the subset of points from I which are contained in all of the half-spaces. The convex hull of a set of points can be found by using the algorithm of Barber et al. [2].

					Ikonen		Stoyan		
Name	Faces	Volume	Bounding box	Type	1	2	1	2	3
Block1	12	4.00	1.00 × 2.00 × 2.00	Convex					
Part2	24	2.88	1.43 × 1.70 × 2.50	Non-convex	3	8			
Part3	28	0.30	1.42 × 0.62 × 1.00	Non-convex	2	2			
Part4	52	2.22	1.63 × 2.00 × 2.00	Non-convex	1	1			
Part5	20	0.16	2.81 × 0.56 × 0.20	Non-convex	2	2			
Part6	20	0.24	0.45 × 0.51 × 2.50	Non-convex	2	2			
Stick2	12	0.18	2.00 × 0.30 × 0.30	Convex					
Thin	48	1.25	1.00 × 3.00 × 3.50	Non-convex					
Convex1	14	176.00	5.00 × 6.00 × 8.00	Convex			1	1	2
Convex2	4	74.67	11.00 × 4.00 × 14.00	Convex			1	1	4
Convex3	10	120.00	3.00 × 4.00 × 12.00	Convex			1	1	6
Convex4	16	124.67	3.00 × 4.00 × 16.00	Convex			1	1	4
Convex5	18	133.33	4.00 × 8.00 × 10.00	Convex			1	3	4
Convex6	8	147.00	6.00 × 7.00 × 7.00	Convex			1	2	3
Convex7	16	192.50	6.00 × 10.00 × 9.00	Convex			1	3	2
				Number of polyhedra:	10	15	7	12	25

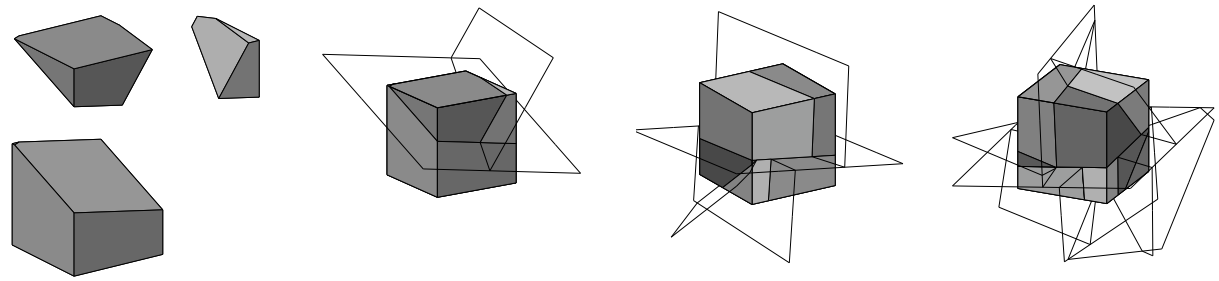
Table 1: Characteristics of the three-dimensional polyhedra from the literature used in the experiments.The rightmost 5 columns describe the sets of polyhedra used in the problems presented in the originating papers. A number in one of these columns is the number of copies of the polyhedra in the corresponding problem instance, e.g., 6 copies of the polyhedron named Convex3 is present in the problem instance Stoyan3.

Given a positive integer n, the construction of an n-piece puzzle commences as follows. Initially, a set of 6 half-spaces, H_0, is generated such that they correspond to a cube. Now let $\mathcal{P}_1 = \{H_0\}$ then we will iteratively construct a sequence of half-space sets $\mathcal{P}_i$. To do this, we select the smallest cardinality half-space set $H \in \mathcal{P}_i$ for each i and generate a random plane which can be used to split H into two new sets H' and H''. We then let $\mathcal{P}_{i+1} = (\mathcal{P}_i \setminus H) \cup H' \cup H''$, i.e., the set of half-space sets containing H' and H'' as well as all sets from $\mathcal{P}_i$ except $\mathcal{H}$. Since the cardinality $|\mathcal{P}_i| = i$, it follows that $|\mathcal{P}_n| = n$. If the random plane used to split each half-space set has been selected appropriately we may generate n non-empty convex polyhedrons from the half-space sets in $\mathcal{P}_n$. In Figure 9, three examples of various sizes are visualized including the cutting planes used to generate them.

It is important to emphasize that a solution method specifically designed with this type of instances in mind may be able to find better solutions more efficiently than our general method. However, since the optimal utilization for these instances is 100%, we may use them to evaluate the quality of the solutions produced by the heuristic. It is interesting to see if we can actually solve some of them even though the solution method is obviously not ideal for puzzle-solving.

6.3 Benchmarks

The two problems given by Ikonen et al. [19] (see Table 1) are decision problems with a cylindrical container and they have already been shown to be easily solved by Egeblad et al. [11]^{A}.



(a) (b) (c) (d)

Figure 9: Examples of three different puzzles and the cutting planes used to generate them. (a) Three convex polyhedra. (b) The corresponding cube and cutting planes. (c) A puzzle with 5 pieces. (d) A puzzle with 10 pieces.

The only previous results and thus also the best results for the Stoyan instances are reported by Stoyan et al. [25] and their results are repeated in the first two columns of Table 2.

	Stoyan		Bounding box		3DNest		
Problem	Height	Util. (%)	Height	Util. (%)	Height	Util. (%)	Improvement
Stoyan1	27.0	29.88	46	17.54	19.31	42.05 (3.4)	12.17
Stoyan2	30.92	27.21	34	24.75	19.83	42.45 (1.0)	15.24
Stoyan3	45.86	29.33	45	29.90	29.82	45.12 (0.8)	15.79

Table 2: The results from Stoyan et al. [25] are compared to the average results of 10 runs of 10 minutes with 3DNest. Results for the initial solution found by 3DNest is also reported. The second last column includes the standard deviation over the 10 runs. The last column emphasizes the difference between 3DNest and the approach by Stoyan et al.

In Table 2, we also report the results of the initial solution found using the algorithm described in Section 5.2 and the average results found by running 3DNest with 10 different random seeds and 10 minutes for each seed. Note that the initial solution found is actually slightly better than the solution found by Stoyan et al. [25] for the largest problem instance. The last column of Table 2 emphasizes the percentage of material saved (on average) when 3DNest is compared to the results from Stoyan et al. The utilization of 3DNest is on an average about 14 percentage points higher than that of Stoyan et al. and the average improvement over the utilization of Stoyan et al. is 50.2%. This demonstrates that 3DNest performs very well in comparison to existing methods.

In order to make some additional experiments, a new problem instance, Merged1, was created by combining the polyhedra from Stoyan and Ikonen. It contains one copy of each of the polyhedra in Table 1 and the Ikonen polyhedra have been scaled with a factor of 4 to better match the size of the Stoyan polyhedra. Larger versions of this problem instance, Merged$i$, are simply created by making i number of copies of each polyhedron. The dimensions of the

container for these instances are chosen such that a solution with 50% utilization is a cube. Furthermore, we have used the puzzle generator described above to generate 40 puzzles with 5, 10, 20 and 40 pieces. Rather than testing each puzzle with 10 different random seeds, 10 different puzzles are tested for each of the four cardinalities. The purpose of this is to illustrate the heuristic's capabilities independently of the input data. Each shape of these puzzles has an average of about 11-13 facets which is very similar to the Stoyan instances.

Results are presented in Table 3. The utilization of the initial solution (0 seconds) and the utilization after 10, 60, 300 and 600 seconds are reported. All values are averages over 10 runs with different seeds for 3DNest, except in the case of the puzzles where the seed is used to vary the problem instance. The best solution after 10 minutes and the standard deviation is also reported and so is the average number of translations done per second.

		Utilization after number of seconds					Max.	Std.	Translations
Problem	Size	0	10	60	300	600	util.	dev.	per second
Stoyan1	7	17.54	39.76	41.60	42.05	42.05	46.38	3.4	1468
Stoyan2	12	24.75	38.25	39.90	41.79	42.45	44.27	1.0	887
Stoyan3	25	29.90	39.19	42.49	44.58	45.12	46.67	0.8	756
Merged1	15	23.44	37.29	39.68	42.38	42.97	44.12	1.0	462
Merged2	30	23.62	30.23	39.77	42.80	42.92	42.99	0.1	295
Merged3	45	24.58	27.02	35.49	42.23	43.32	44.99	1.0	265
Merged4	60	24.80	26.09	31.61	40.06	41.99	42.81	0.5	233
Merged5	75	26.17	26.66	29.63	37.99	40.96	42.56	0.7	199
Puzzle5	5	28.85	98.30	98.89	98.89	99.22	100.00	2.2	-
Puzzle10	10	20.90	72.68	84.96	93.74	94.30	100.00	14.4	353
Puzzle20	20	15.77	42.27	50.05	72.20	82.54	95.16	12.2	205
Puzzle40	40	13.62	26.40	34.56	45.68	49.59	70.85	7.8	145

Table 3: Average results obtained by running 3DNest 10 times for at most 10 minutes in each run. Results include the utilization obtained with the initial solution, within 10 seconds and within 1, 5 and 10 minutes. The maximum utilization and standard deviation is also included for the results after 10 minutes. Finally, the average number of translations per second is presented except in the case of Puzzle5 for which an optimal solution was often found within a second. Note that the results for the Puzzle problems are on 10 different instances rather than with 10 different random seeds.

After 10 seconds the results are already better than those of Stoyan et al. for all of the Stoyan instances with an average utilization close to 40%. The heuristic continues to improve solutions but utilization is only improved by less than 1 percentage point after 300 seconds. Solutions for the Merged instances appear to be quite good with utilizations matching the smaller Stoyan instances even with as many as 60 and 75 shapes. Also here, solutions are generally only improved by one percentage point after the first 300 seconds with the exception of Merged5. The optimal utilization for Merged5 is probably higher than it is for Merged1 which is also indicated by the initial solutions, but the slow decline in the number of translations performed is also a strong indication that large problem instances are solved efficiently by 3DNest. Puzzles with 5 or 10 pieces are most often solved to optimality and even puzzles with 20 pieces are handled quite well within the time limit of 10 minutes. It is also clear though that puzzles with 40 pieces is more than our solution strategy can handle.

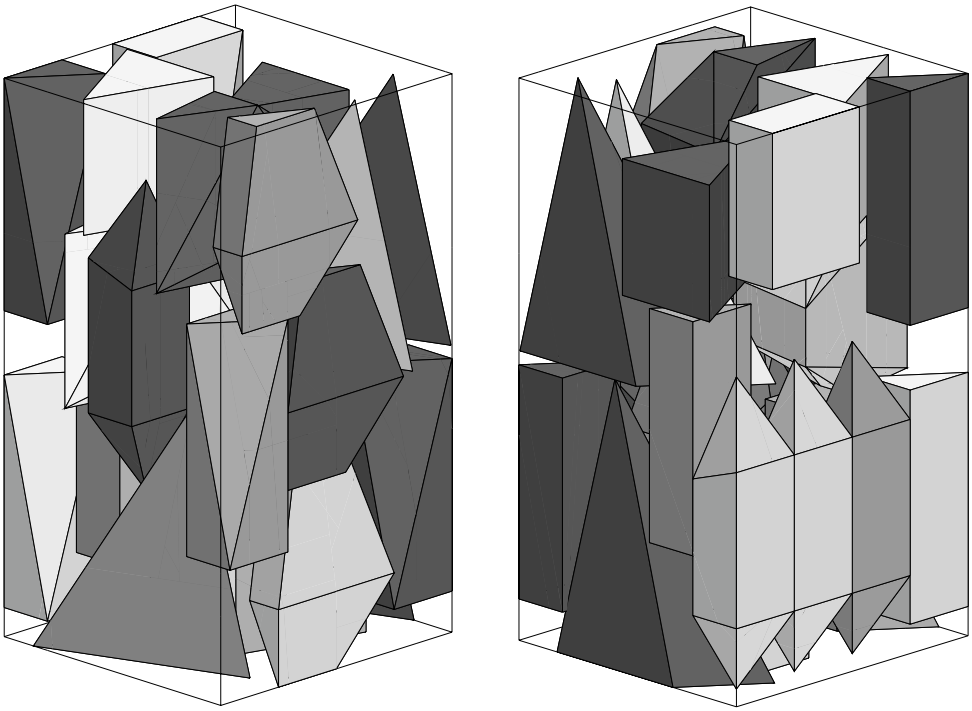


Figure 10: The best solution found for Stoyan3 without using rotation (from two different angles). The utilization is 46.67%.

The average utilization of these instances is only 50% after 10 minutes and the best found utilization is less than 71% which is far from the optimal 100%. The best solutions found for Stoyan3 and Merged5 are shown in Figure 10 and Figure 11.

A simple strategy for handling rotation has also been implemented in 3DNest. The local search neighborhood was expanded, so that in addition to trying translations in three directions, 24 different orientations (90° increments for each axis) are also tried. In each iteration the translation or orientation which results in least overlap is chosen. This was mainly done to get an indication of the improvement possible in the utilization when allowing rotation. The results are presented in Table 4 and better results are indeed obtained for the Stoyan instances while some of the large Merged instances are not handled very well, most likely because of the increased amount of computations needed and the increased size of the solution space.

A lower bound on the height of the Stoyan instances and Merged1 is 16 since these instances contain a shape (see Convex4 in Table 1) with height 16 which cannot be rotated and still be within the bounds of the container. In all runs on Stoyan1 and Merged1 as well as most runs on Stoyan2, 3DNest is able to find solutions matching this bound and therefore these solutions are optimal.

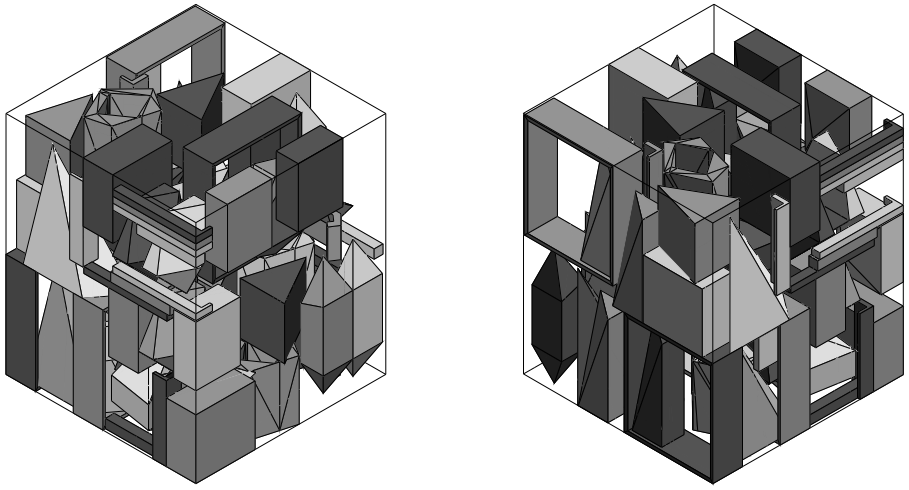


Figure 11: The best solution found for Merged5 without using rotation (from two different angles). The utilization is 42.12%.

7 Conclusion

In this paper we have presented a solution method for the multi-dimensional strip-packing problem. An earlier version of this method was previously tested by Egeblad et al. [11]^{A} and proved very successful for two dimensions. Three problem instances in three dimensions, by Stoyan et al. [25], were used to show that the presented solution method is able to reach far better results than those by Stoyan et al. [25]. The heuristic has also been tested on problems where the optimal value is known, and has proven able to find the optimal solution for instances with 10 items and close to optimal solutions for instances with 20 items. A simple rotation scheme shows that increased utilization may be achieved by allowing rotation, and optimal solutions are found for instances with 7 and even 15 items.

The translation algorithm presented in Section 4 is strongly connected to packing problems, but it is important to emphasize that the algorithm could also be used to maximize the volume of intersection of polytopes with an axis-aligned translation. Also, the restriction to axis-aligned translations is imposed only in order to keep the mathematical details as simple as possible. It is, of course, possible to alter the algorithm for translation in an arbitrary direction: a trivial approach would be to rotate the input data.

It is also important to note the simplicity of the translation algorithm which is able to work directly with the faces of the polyhedrons. Unlike many other methods, as presented in Section 3, we do not rely on additional approximating data-structures such as octrees, depth maps or voxels. Even though intersection volumes of non-convex polytopes are calculated, the intersections are never explicitly constructed. Non-convex polytopes are handled as easily as the convex ones and even holes are handled without any changes to the algorithm. Other problem variants might include non-rectangular containers [17], quality regions [17], repeated

Problem	Size	Average Height	Utilization Avg.	Min.	Max.	Translations per second
Stoyan1	7	16.00	50.42 (0.0)	50.42	50.43	1325.21
Stoyan2	12	16.13	52.17 (0.5)	51.15	52.58	682.27
Stoyan3	25	25.60	52.57 (1.2)	50.41	54.02	673.50
Merged1	15	16.00	46.87 (0.0)	46.87	46.87	440.18
Merged2	30	20.97	45.09 (1.3)	43.50	47.72	305.21
Merged3	45	26.50	40.83 (0.9)	39.75	42.95	299.31
Merged4	60	32.93	36.25 (1.9)	33.44	39.26	275.15
Merged5	75	40.27	31.89 (1.2)	29.31	33.52	242.22

Table 4: Results obtained when allowing rotation.Average utilization, standard deviation, minimum and maximum utilization are reported. The last column is the average number of translations per second.

patterns [23]$^{\mathrm{B}}$ and more. Although these references are for the 2D problem, the generalized constraints can be handled by our solution method in any dimension, essentially without affecting the running time.

If one does not calculate the minimum between each set of breakpoints in the translation algorithm, then only rational numbers are needed for the solution method described (given that the input problem only uses rational numbers). This property permits the use of the method if exact calculations are needed for some reason. In two dimensions, a minimum between breakpoints can also be found using rational numbers since one only needs to solve linear equations.

Mount et al. [22] described what is essentially a 2D translation algorithm in 2D space (solving 2D2DTP). Given polygons with n and m edges, the worst case running time is $O((mn)^2)$. Their approach is based on an *arrangement* of line segments. A 3D arrangement of polygons could be used for solving 3D3DTP, but as far as we know an algorithm for constructing a 3D arrangement of polygons does not currently exist. It is also an open question how to make an algorithm for 2D3DTP or more generally d'DdDTP for any $d \geq 3$ and $d' \in \{2, \ldots, d-1\}$. For the sake of completion, de Berg et al. [7] solve the maximization variant of 2D2DTP with two convex polygons in time $O((n+m)\log(n+m))$; Ahn et al. [1] consider a generalization of the same problem in which they allow rotation.

Free orientation of shapes is one of the most important directions for future research. Especially when considering that most applications of packing 3D shapes, e.g., rapid prototyping, do allow free orientation. Another important direction for future research is how to also handle some of the constraints which are typically part of more general layout problems, e.g., constraints concerning gravity or wire length [5].

References

[1] H.-K. Ahn, O. Cheong, C.-D. Park, C.-S. Shin, and A. Vigneron. Maximizing the overlap of two planar convex sets under rigid motions. *Computational Geometry*, 37(1):3–15, 2006.

[2] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, 1996.

[3] A. Bortfeldt, H. Gehring, and D. Mack. A parallel tabu search algorithm for solving the container loading problem. *Parallel Computing*, 29:641–662, 2002.

[4] 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.

[5] J. Cagan, K. Shimada, and S. Yin. A survey of computational approaches to three-dimensional layout problems. *Computer-Aided Design*, 34(8):597–611, 2002.

[6] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright. Simplification envelopes. *Computer Graphics*, 30(Annual Conference Series):119–128, 1996.

[7] M. de Berg, O. Cheong, O. Devillers, and M. van Kreveld. Computing the maximum overlap of two convex polygons under translations. *Theory of Computing Systems*, 31(5):613–628, 1998.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

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

[14] 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.

[15] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Information Processing Letters*, 12(3):133–137, 1981.

[16] P. Hachenberger and L. Kettner. 3D boolean operations on nef polyhedra. In C. E. Board, editor, *CGAL-3.2 User and Reference Manual*. 2006. URL `http://www.cgal.org/Manual/3.2/doc_html/cgal_manual/packages.html#Pkg:Nef3`.

[17] J. Heistermann and T. Lengauer. The nesting problem in the leather manufacturing industry. *Annals of Operations Research*, 57:147–173, 1995.

[18] S.-M. Hur, K.-H. Choi, S.-H. Lee, and P.-K. Chang. Determination of fabricating orientation and packing in sls process. *Journal of Materials Processing Technology*, 112(2-3):236–243, 2001.

[19] 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.

[20] J. Lawrence. Polytope volume computation. *Mathematics of Computation*, 57(195):259–271, 1991.

[21] 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.

[22] D. M. Mount, R. Silverman, and A. Y. Wu. On the area of overlap of translated polygons. *Computer Vision and Image Understanding*, 64(1):53–61, 1996.

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

[24] 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.

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

[26] P. E. Sweeney and E. R. Paternoster. Cutting and packing problems: A categorized, application-orientated research bibliography. *Journal of the Operational Research Society*, 43(7):691–706, 1992.

[27] 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.

[28] G. Wäscher, H. Haussner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 2006. In Press.

[29] X. Yan and P. Gu. A review of rapid prototyping technologies and systems. *Computer Aided Design*, 28(4):307–318, 1996.

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

[31] 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.

Submitted

Using directed local translations to solve the nesting problem with free orientation of shapes

Benny K. Nielsen* Marcus Brazil† Martin Zachariasen*

Abstract

The nesting problem is the problem of packing a set of shapes within the boundaries of some container. An existing efficient solution method for this problem is based on the following strategy. Construct a random initial placement of the shapes and then repeatedly move one of the shapes to a position where the total area of overlap between pairs of shapes is decreased. Previously, only axis-aligned translation directions have been used for this approach and only a fixed number of shape orientations could be handled. We demonstrate how to efficiently translate in arbitrary directions and how to use this to handle free orientation of shapes. Computational experiments show that this approach is an improvement when compared with existing results in the literature. Finally, it is shown how all of the techniques discussed can be generalized to three or even more dimensions.

Keywords: Translation algorithm, minimizing overlap, free orientation, packing, nesting

1 Introduction

Nesting is a term traditionally used to describe the process of designing a layout of shapes to be manufactured from some flat raw material, e.g., for cutting pieces of clothes from a roll of fabric. *The nesting problem* is the problem of doing this as efficiently as possible, i.e., wasting as little of the material as possible. The problem comes in numerous variations depending on the specific application. In particular, the material can impose some constraints on the freedom of orientation of the parts to be cut, e.g., parts to be cut from a roll of fabric are often not allowed to be rotated with the exception of a 180° rotation. Such constraints are less likely when dealing with materials such as glass or metal. In the existing literature, problems with no rotation allowed or just a few rotation angles have received much more attention than the nesting problem with free orientation of shapes. This may partly be because of the application of the problem in the textile industry, but it is also because of the greater intrinsic difficulty of solving the problem with free orientation. In this paper, we present a new solution method for solving nesting problems with free orientation of shapes. Although most of the paper is oriented towards the two-dimensional problem, the techniques described also work for generalizations to higher dimensions. Therefore, in the typology of Wäscher et al. [21], the problem handled in this paper is the d-dimensional irregular open dimension problem (ODP) with free orientation of shapes.

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

†Department of Electrical and Electronic Engineering, The University of Melbourne, Victoria 3010, Australia. E-mail: brazil@unimelb.edu.au.

We assume that the shapes to be nested are polygons; the container is usually assumed to be a rectangle, though it could also be an arbitrary polygon. The shapes may be non-convex and contain holes. The simplest version of the problem is a straightforward decision problem which can be described as follows:

Nesting Decision Problem. *Given a set of polygons $\mathcal{P}$ and a polygonal container C, determine whether the polygons can be placed (allowing free orientation) such that no two polygons overlap — and all polygons are within the bounds of the container.*

Depending on the application, the related optimization problem to be solved could, e.g., be a knapsack problem, a bin packing problem or a strip packing problem. Most of the existing literature is focused on the strip packing problem and this is also the case in this paper. For this problem, it is assumed that the width w of the material is fixed and the problem is then to minimize the length of the material (the strip). The quality of a solution is most often given in terms of the *utilization* of the material. If the sum of the areas of all polygons in $\mathcal{P}$ is denoted A and the container has area A_C, then the utilization is A/A_C. Hence, the utilization at a given length l is then $A/(w \cdot l)$.

It is important to note that most of this paper is actually only concerned with the decision problem described above. The only exception is that the computational experiments are concerned with the strip packing problem. Existing work on this or similar problems is described in Section 2.

The solution method described in this paper is based on an existing solution method first described by Egeblad et al. [8]$^{\mathrm{A}}$. Their solution method is based on the repeated use of a translation algorithm which can find the minimum area overlap for a given polygon under translation in an axis-aligned direction (horizontally or vertically). We first describe how to generalize this translation algorithm such that it can be used to translate polygons in directions which are not axis-aligned (Section 3). This might seem trivial at first, but we show how to do it efficiently with regard to both speed and precision. The use of arbitrary translation directions encourages a new movement strategy in which short local moves are done based on the location of neighboring polygons (Section 4). Using this technique, it is straightforward to handle free orientation of shapes as well (Section 5), but we do not know of any existing solution methods which handle it in a similar manner. It is especially important to note that no fixed set of rotation angles is used. Instead the rotation angles used depend on the problem instance itself, e.g., if rectangles are packed in a rectangular container then our scheme would never attempt a rotation angle which is not a multiple of 90°.

As mentioned above, we use these techniques to extend the solution method described by Egeblad et al. [8]$^{\mathrm{A}}$. In Section 6 we present some computational experiments which mainly show how much our techniques improve utilization when compared to solutions found without free rotation. A comparison with results from the literature is also presented and this generally favors the approach described in this paper.

All of the techniques described can be generalized to higher dimensions. A discussion of the issues involved in such a generalization is given in Section 7. Concluding remarks and future directions are given in Section 8.

2 Related Work

Nesting problems and a wide range of solution methods have been discussed extensively in the existing literature. A survey is given by Dowsland and Dowsland [6] for 2D problems.

Recent work is described in the introductions of the papers by Egeblad et al. [8]^{A}, Burke et al. [1], and Gomes and Oliveira [10]. Note that most of the existing literature is focused on the nesting problem with only few or no rotation angles allowed. In this section, we only describe solution methods which (to some extent) can handle free orientation of the shapes to be nested. Interestingly, the literature about 3D nesting/packing problems with irregular shapes is most often focused on free orientation. This is most likely due to the fact that problems with restrictions on freedom of orientation are less common in three dimensions. A survey of 3D layout problems (a broader context) was given by Cagan et al. [2].

Solution methods which use iterative improvements of some initial placement are relatively easy to extend to the handling of free orientation. Typically, these methods have some neighborhood of possible short translations of individual shapes; this neighborhood can simply be extended with some set of rotation angles or some small change of the current orientation. This approach is, e.g., exemplified by Jain et al. [14] for the problem of nesting blanks on a coil of metal. More recent work in 3D includes Hur et al. [13] and Eisenbrand et al. [9].

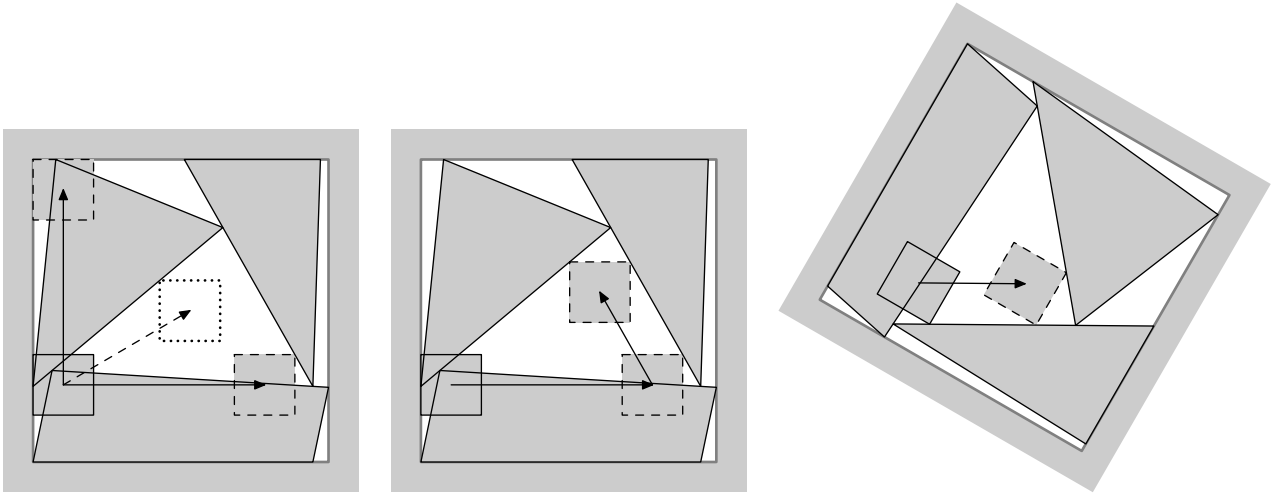
An alternative and popular solution approach to nesting problems is a serial placement algorithm in which the shapes are placed one by one. This is, e.g., the approach used by Heistermann and Lengauer [12] and Crispin et al. [4] for a problem in the leather manufacturing industry where the container is an (irregular) animal hide. Crispin et al. [4] simply use a neighborhood with 20 possible angles while Heistermann and Lengauer [12] use compaction techniques as, e.g., described by Li and Milenkovic [15]. They determine the orientation of the next shape to be placed on the basis of the orientation of any neighboring shapes in the current placement. This often means that neighboring edges are aligned. A serial placement algorithm by Dickinson and Knopf [5] uses simulated annealing to place each shape in a position which leaves the remaining free space as compact as possible. Finally, Liu and He [16] use a *lowest-gravity-center principle* to place shapes. They find the orientation of each shape by recursively dividing a range of possible angles into a number of fixed angles. The angle which results in the lowest gravity center is used as the center of the next range which simply extends to the neighboring fixed angles. This is the only paper in which results have been found which can be compared to our new solution method (see Section 6).

The work of Milenkovic [18] is in a class of its own. Using mathematical programming he shows how to obtain optimal solutions for problems involving free orientation of shapes. Although the method is only practical for at most 2 or 3 shapes, it is an important step in a research field which is mostly dominated by heuristic and meta-heuristic techniques. Milenkovic [17] also shows how to do *rotational compaction*. Given a placement (without overlap), mathematical programming is used to translate and rotate the shapes to a local minimum. Experimental results show that rotational compaction improves the utilization when compared to only doing translational compaction.

3 Directional Translations

The most important part of the solution method for the nesting problem by Egeblad et al. [8]^{A} is an efficient translation algorithm which is used to repeatedly move polygons horizontally or vertically to positions which minimize the area of overlap (see Figure 1a). Formally, the horizontal translation algorithm solves the following problem.

Horizontal Translation Problem in 2D. *Given a fixed polygonal container C, a polygon P with fixed position with respect to its y coordinate, and a polygon Q with fixed position, find*



(a) (b) (c)

Figure 1: Intermediate placements in the search for a non-overlapping solution. (a) In the solution method by Egeblad et al. [8]^{A}, the box can only be moved horizontally or vertically. In this it would have been better with a different direction as indicated by the dashed arrow. (b) Another example of a non-axis-aligned translation direction. In this case the direction is based on an edge of a neighboring triangle. (c) Any translation direction can be handled by first rotating the problem and then making a horizontal translation.

a horizontal offset x for P such that the area of overlap between P and Q is minimized (and P is within the bounds of the container C).

The translation algorithm by Egeblad et al. [8]^{A} works even for very general definitions of a polygon. Self-intersections are not allowed (and make no sense in this context), but multiple disjoint components and holes are allowed. Naturally, the algorithm is very easily changed to provide an answer for a vertical direction instead of a horizontal direction.

In this paper, we examine how we can most easily handle translations in any non-axis-aligned direction as illustrated in Figure 1b. Although the problem definition above is only for a horizontal direction, a solution to the problem would also be applicable for any other direction. Naively, it would simply be a matter of rotating all polygons (including the container) such that the desired direction becomes horizontal. The resulting position can then be rotated back to provide a solution to the original problem. It turns out that one can do much better than resorting to using a large number of trigonometric calculations. Mathematically — and with respect to standard use of theoretical running time — these trigonometric calculations are not a problem. In practice, they are slow and they can cause problems with the precision of floating point calculations.

Let us state the more general problem of directional translations without reducing it to the special case of horizontal translation.

1-Dimensional Translation Problem in 2D (1D2DTP). *Given a fixed polygonal container C, a polygon P, a direction vector v, and a polygon Q with fixed position, find a position for P in the direction of v or $-v$, such that the area of overlap between P and Q is minimized (and P is within the bounds of the container C).*

It would be possible to adapt the translation algorithm by Egeblad et al. [8]^{A} to work in arbitrary directions by simply changing all calculations such that they no longer assume an axis-aligned direction. This would make most calculations more complicated and in particular, the distances computed would require square root calculations. Again, theoretical running time would not suffer and mathematically it would work correctly, but in practice it would be slower and it could increase problems with precision.

One of the important properties of the horizontal translation algorithm is that it can be implemented using rational numbers only [7]^{C}, under the assumption that all vertices of the problem have rational coordinates. It is not immediately clear that the same property can be obtained for arbitrary directions since either trigonometric calculations are needed to transform the problem or square roots are needed for distance calculations. In this section, we show how the horizontal translation algorithm can be generalized to arbitrary directions with just a few simple changes — both theoretically and in terms of lines of code. At the same time we preserve the property of being able to do it exactly using only rational numbers, for any vector v with rational gradient.

First, we briefly review the horizontal translation algorithm. Full details and a proof of correctness can be found in the original paper by Egeblad et al. [8]^{A}. The horizontal translation algorithm is based on a theorem which states that the area of the overlap of two polygons, P and Q, can be computed as a sum of (signed) areas *horizontally* between pairs of edges, one from P and one from Q. Not surprisingly, this theorem also works if an arbitrary direction is used instead of a horizontal one. It is not surprising since one could simply rotate the problem to make it equivalent to the horizontal case. Nevertheless, we are going to state the theorem explicitly for arbitrary directions since we do not want to rotate the problem. For this purpose we first need a definition of the area between two edges in a given direction.

Definition 15 (Edge Region)**.** *Given an edge f, we say that a point $p \in f$ if and only if p is a point on the edge f. Given two edges f and g and a translation direction represented by a vector v, we define the* edge region $R_v(f, g)$ *as follows.*

$$R_v(f, g) = \{p \in \mathbb{R} \mid p + s_f v \in f, p - s_g v \in g, s_f, s_g \in \mathbb{R}_+\}.$$

In other words, the edge region $R_v(f, g)$ of two edges, f and g, is the set of points that are in between f and g, but in such a way that g is in the direction of $-v$ and f is in the direction of v from each such point. If we informally think of v as pointing to the right, then note that for two edges, f and g, where g is completely to the right of f the edge region $R_v(f, g)$ is empty. Also note that $R_v(f, g) = R_v(g, f)$ if and only if both edge regions are empty.

It is useful to divide the edges of a polygon into three different classes.

Definition 16 (Signs of Edges)**.** *Given a polygon P with edge set F and a translation direction represented by a vector v, we say that an edge $f \in F$ is*

- positive*, if the interior of P is in the direction of v relative to f,*
- negative*, if the interior of P is in the direction of $-v$ relative to f,*
- *and* neutral*, if it is neither positive nor negative.*

The set of all positive edges in F is denoted F^+ and the set of all negative edges is denoted F^-.

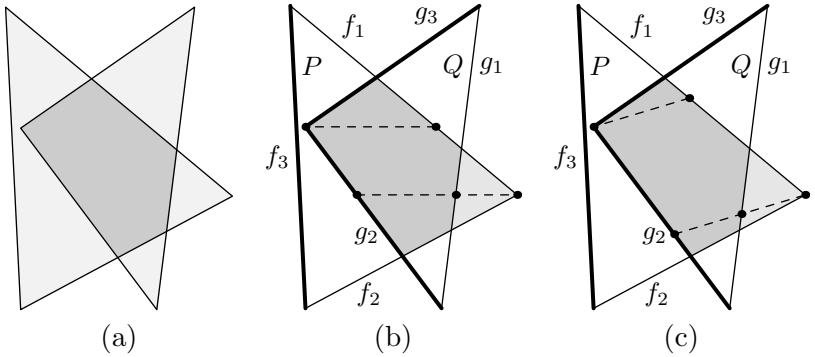


Figure 2: (a) Two overlapping triangles. (b) The area of the overlap can be computed based on areas horizontally between pairs of edges. More precisely, based on Equation 1, the area is $A(R(f_1, g_3)) + A(R(f_1, g_2)) + A(R(f_2, g_2)) - A(R(f_1, g_1)) - A(R(f_2, g_1))$. Bold edges are positive and non-bold edges are negative. (c) This works in other directions as well. In this case, it is the same edge pairs as before, but this could be different for other directions.

We now state the main theorem needed for generalizing the translation algorithm to arbitrary directions.

Theorem 3. *Given two polygons P and Q with edge sets F and G, respectively, and a translation direction represented by a vector v, let $R_0 = P \cap Q$. For any region R, let $A(R)$ denote the area of R. Then:*

$$A(R_0) = \sum_{\substack{f \in F^+ \\ g \in G^-}} A(R_v(f,g)) + \sum_{\substack{f \in F^- \\ g \in G^+}} A(R_v(f,g)) - \sum_{\substack{f \in F^+ \\ g \in G^+}} A(R_v(f,g)) - \sum_{\substack{f \in F^- \\ g \in G^-}} A(R_v(f,g)). \quad (1)$$

The theorem stated by Egeblad et al. [8]$^{\mathrm{A}}$ is identical to the theorem above except that a horizontal vector v is assumed. The proof there generalizes easily to an arbitrary direction vector v, and there is no need to repeat it here. An illustration of a simple overlap between two triangles and the edge regions which can be used to compute its area, can be seen in Figure 2. Note that the theorem applies to non-convex polygons as well as convex ones.

Given a horizontal positive direction, the situation for two given edges f and g from polygons P and Q respectively, is illustrated in Figure 3. Assume that initially f is to the left of g. The two edges can only contribute to the overlap of P and Q after translation if their vertical extents overlap (Figure 3a). The parts of the edges outside this vertical overlap can be ignored as illustrated in Figure 3b. When P is translated to the right a sufficiently large distance, f passes through g. Before they intersect (with f to the left of g), the edge region between them is empty. While the two edges intersect, the area of the edge region is the area of a triangle (see Figure 3c) and when f is entirely to the right of g, the area of the edge region is a simple linearly growing area based on a constant height (Figure 3d). The computations needed to derive the area of the growing triangle are trivial, but also constitute a special case of the computations needed for handling arbitrary translation directions described later in this section.

Now the translation algorithm by Egeblad et al. [8]$^{\mathrm{A}}$ can be sketched as follows. For each pair of edges, f and g (with g fixed and f moving), two *breakpoints* are generated,

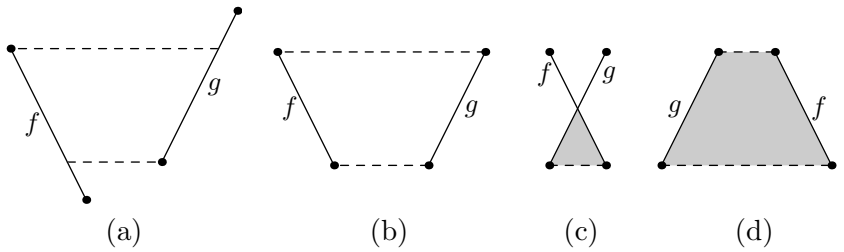


Figure 3: Examples of the edge region $R_v(f, g)$ for a horizontal vector v and various horizontal positions of f. (a) Some parts of f and g are irrelevant for their edge region and they can be ignored. (b) When f is completely to the left of g, the edge region is empty. (c) When the edges intersect, the edge region is triangular. (d) When f is to the right of g the edge region is a quadrilateral with two opposite parallel sides.

corresponding to the horizontal position of a reference point of P (the polygon containing f) at the beginning and the end of the intersection of f and g, since these events mark the non-differentiable changes in the area-function. Each breakpoint is provided with a quadratic function describing the new area function to the right of that point (in terms of the distance from the initial position), and a distance value describing how far f has been moved. When breakpoints have been generated for all pairs of edges, one of which belongs to the moving polygon P and the other to one of the fixed polygons, the breakpoints are sorted according to distance, and are visited one by one while searching for a minimum of the function. For a more detailed description we refer to the original description [8]$^{\mathrm{A}}$.

We now turn to the problem of area computations for arbitrary directions, other than the vertical direction. This more general problem is illustrated in Figure 4a. The dashed lines indicate the translation direction and they also show the projection of all four end-points onto the y-axis. Using these projections it is easy to determine whether the two edges, f and g, can contribute to the overlap of P and Q when P is translated. It is also easy to see that finding the relevant parts of the edges can be done using very simple calculations. The result is illustrated in Figure 4b. Now, it would be natural to use the distances between the end-points of the edges to measure when the two edges begin and complete their intersection, as in the horizontal case. A solution to the translation problem would then be given as the distance traveled by P in order to obtain the minimum area of overlap. As previously noted this would require square root calculations.

Square root calculations can be avoided if one does not measure the distances, but instead only measure the distance traveled in the horizontal direction. The solution to the problem is then the x-coordinate of the optimal position. Given a pair of edges, f and g, we want to compute the area of their edge region in terms of the horizontal distance traveled by f. As illustrated in Figure 4b, the important distances are the distance traveled when the intersection of f and g starts (t_1) and the distance traveled when the intersection of f and g stops (t_2). For any value $t \leq t_1$, the area of the edge region is zero.

Assuming $t_1 \neq t_2$ then the main problem is to obtain a function describing the area of the region $R_v(f, g)$ illustrated in Figure 4c during translation ($t \in [t_1, t_2]$), i.e., the area of a growing triangle. Also assume that $t_1 = 0$. Now, it is convenient to describe two sides of the triangle using two parameterized vectors, $\mathbf{f}(t) = (\mathbf{f}_x(t), \mathbf{f}_y(t))$ and $\mathbf{g}(t) = (\mathbf{f}_x(t), \mathbf{f}_y(t))$, rooted at the intersection of f' and g', as illustrated in Figure 4c. For $t = t_2$, these vectors exactly

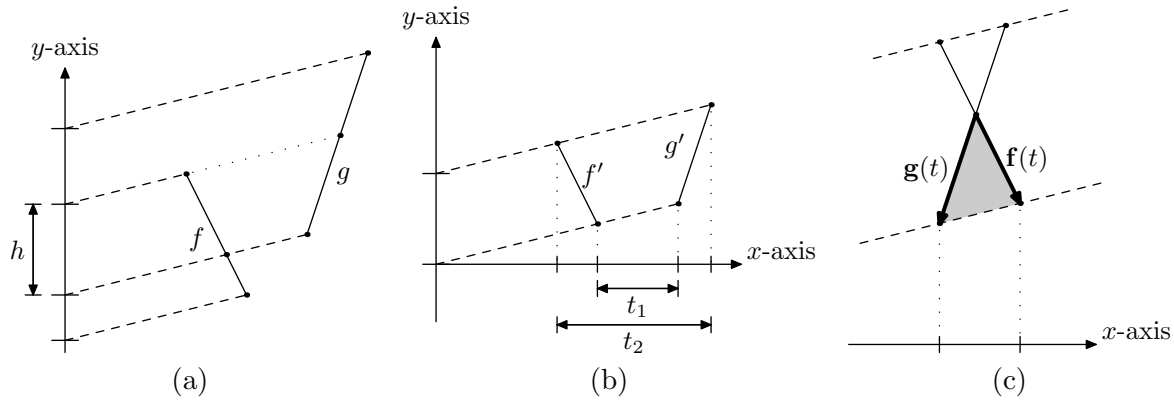


Figure 4: (a) To determine whether two edges, f and g, intersect when one of them is translated in a given direction, one can project them using the same (or opposite) direction onto the y-axis. (b) The edges are now shortened such that only the relevant parts are considered. These edges are denoted f' and g'. The values t_1 and t_2 are the distances needed for translation of f' with respect to the x-axis before the edges start and stop intersecting, respectively. (c) To compute the area of the region between f' and g', it is useful to use vectors rooted at the intersection. These vectors are denoted $\mathbf{f}(t)$ and $\mathbf{g}(t)$ where t is the distance traveled by f.

overlap f' and g'. A function, $\alpha_0(t)$, to describe the area of the triangle delimited by these vectors can now be stated as follows.

$$\alpha_0(t) = |\mathbf{f}_x(t)\mathbf{g}_y(t) - \mathbf{f}_y(t)\mathbf{g}_x(t)|/2.$$

The lengths the vectors change linearly with respect to the horizontal distance traveled. Since we have assumed that $t_1 = 0$, we can rewrite the expression to obtain a polynomial.

$$\alpha_0(t) = \frac{1}{2}|t\mathbf{f}_x(1)t\mathbf{g}_y(1) - t\mathbf{f}_y(1)t\mathbf{g}_x(1)| = \frac{1}{2}|\mathbf{f}_x(1)\mathbf{g}_y(1) - \mathbf{f}_y(1)\mathbf{g}_x(1)|t^2$$

Now, if we let $A = \alpha_0(1)$ then the general function $\alpha(t)$ for any value of t_1 easily follows:

$$\alpha(t) = \alpha_0(t - t_1) = A \cdot (t - t_1)^2 = A \cdot (t^2 - 2t_1t + t_1^2) = At^2 - 2At_1t + t_1^2.$$

In the horizontal case we have $\mathbf{f}_y(1) = \mathbf{g}_y(1) = h_1$ where h_1 is the height of the triangle when $|\mathbf{f}_x(1) - \mathbf{g}_x(1)| = 1$. This means that $A = h_1/2$ which is exactly what one would expect.

The remaining small problem is for $t > t_2$. In this case the area of the edge region is the value of $\alpha(t_2)$ plus a simple linearly growing area (the area of a parallelogram based on the value $t - t_2$ and the height h as illustrated in Figure 4a). This is also the only case to handle if $t_1 = t_2$; in this case $\alpha(t_2)$ is defined as zero.

The end result is that in order to handle translations in arbitrary directions, one only needs to do the projections slightly different than in the horizontal case. This methodology then includes horizontal translation as a special case. However, note that it is also necessary to compute the y-coordinate corresponding to the x-coordinate found as a result of the translation algorithm.

This algorithmic method essentially equates a translation in any arbitrary direction (except vertical) with a horizontal translation. Depending on the direction, it could be better instead to use a vertical translation with respect to floating point precision of the calculations. A natural choice — and our choice for the implementation used in Section 6 — would be to choose the closest of these two directions in terms of angle.

The functions described are for the problem of minimizing the area of an overlap. Other overlap measures such as those described by Nielsen and Brazil [20] can also be adapted to work nicely with arbitrary directions. The same is true for higher dimensional translation problems as described by Egeblad et al. [7]^{C}. We return to the subject of higher dimensions in Section 7.

4 Movement Strategy

4.1 Local search neighborhood

So far, we have shown that directional translations can be easily implemented with a small constant overhead in running time. Now we address the question of whether they are really useful. Clearly some problems, e.g., with appropriately shaped containers or polygons, would benefit from translations in other directions than those which are axis-aligned, but the directional translations also open up some new movement strategies. For this discussion we first need a few standard definitions.

The *search space* is all possible placements of the polygons within the boundaries of the container; the majority of these placements contain overlapping polygons. A set of *moves* define which placements can be reached from a given placement. These placements constitute the *neighborhood* of a given placement. A local search iteratively perform moves in the neighborhood of the current placement as long as the value of the objective function can be decreased; otherwise a local minimum has been found. The solution method by Egeblad et al. [8]^{A} for the nesting problem uses a local search neighborhood containing horizontal and vertical translations of polygons. These translations are repeated until all overlap is gone or until no polygon can be moved horizontally or vertically to a position with less overlap. The local minima are then handled by a meta-heuristic method, *guided local search*, which penalizes features of the placement such that it is no longer a locally optimal placement.

In this section we focus on alternatives to repeated horizontal/vertical translations. This is a subject that has received only cursory attention in previous papers, such as Egeblad et al. [8]^{A} and Egeblad et al. [7]^{C}. The following is a discussion of some of the alternatives for a local search neighborhood when translations in arbitrary directions are available. In the case of Egeblad et al. [8]^{A}, the neighborhood only contained horizontal translation moves and vertical translation moves (for every polygon). Two such neighborhoods are illustrated for a single polygon (a square) with the double arrows in Figures 5a and 5b.

The directional translations introduced in the previous section comprise a useful alternative to the horizontal and vertical moves. The straightforward alternative is to extend the neighborhood with a fixed set of extra directions, e.g, diagonal translations. Another alternative is to employ random directions. This makes the neighborhood infinitely large, but can be handled by deciding that if an improvement cannot be obtained for some fixed number of random directions then it is assumed that the polygon cannot be translated to a position with less overlap in any direction.

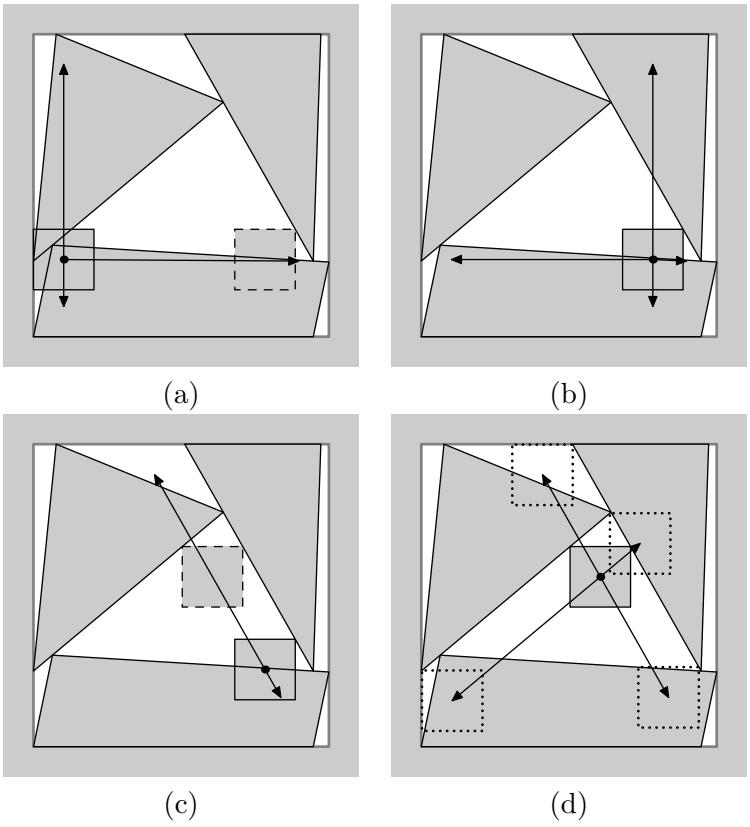


Figure 5: Examples of local search neighborhoods for a single shape (the square). (a) A neighborhood of horizontal and vertical translations. The dashed square indicates the best possible position of the square in this neighborhood. (b) The subsequent neighborhood after translating the square. (c) Alternatively, the neighborhood can contain the aligned translations available corresponding to contact points. Only one contact point and thus one translation direction is available in this case. (d) Two directions are available in the following placement of the square. Note that the possible translations are limited in such a way that at least one of the contact points is preserved. The dotted squares are the extreme positions in the neighborhood.

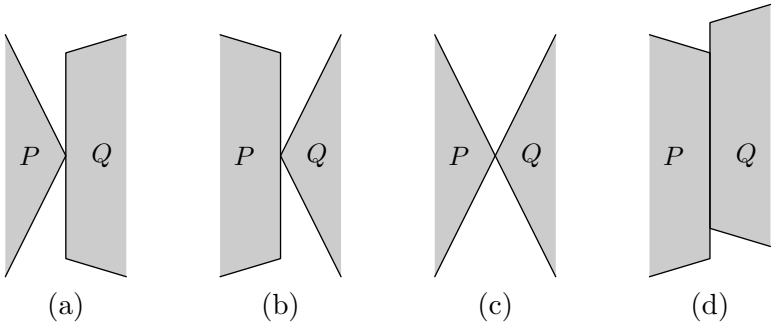


Figure 6: Two non-overlapping polygons P and Q can be in contact with each other in four different ways. (a-b) A vertex of on polygon is on the interior of an edge on the other polygon. (c) Vertex-to-vertex contact point. (d) Similar to the first case except with two contact points, which are the endpoints of the boundary edge segment shared by P and Q.

4.2 Aligned translations

Here we describe a promising alternative strategy based on translations in arbitrary directions. Given two polygons P and Q (one of which may be the container), we say that the point p is a *contact point* for P and Q if p lies on the boundaries of both polygons and p is a vertex of either P or Q. It is easy to see that given any placement without overlap, it can always be compacted such that each polygon has a least one contact point with another polygon. This suggests that contact points are an important feature of non-overlapping placements. There are four different types of contact points come in four variations as illustrated in Figure 6. In all four cases, we can represent the contact point by a vertex from one of the polygons and an edge from the other polygon (there may be more than one candidate for the choice of vertex or edge, as in Figure 6c).

Given a vertex and an edge which constitute a contact point, we can find a directed finite-length translation of one of the polytopes such that this vertex stays on the edge over the course of the translation. In Figure 5b, the upper right corner of the small box is a contact point as it is also on an edge of the triangle in the upper right corner of the placement. This suggests a translation of the box aligned with the edge of the triangle as illustrated by the double arrow in Figure 5c. If the box is translated to the position indicated by the box with dashed edges then the box now has two contact points that should be considered before the next move. This is illustrated by the neighborhood in Figure 5d. We refer to such local vertex-edge moves as *aligned translations*.

Aligned translations are inherently local (short) moves as opposed to the horizontal and vertical translations of Egeblad et al. [8]$^{\mathrm{A}}$ which are global (long) moves in the sense that they can move a an arbitrarily large distance within the container. In the small example used in Figure 5, this difference is not obvious, but for larger problem instances, the difference is important. The example in Figure 5 also suggests a strategy which involves both global and local moves: When a polygon has been selected as a candidate for translation then first use a global move to find its optimal placement in a horizontal or vertical direction and then use a series of local translations to optimize its placement locally (aligned with neighboring polygons). This is the approach chosen for the implementation used for the experiments in Section 6. When multiple contact points (or associated vertex/edge pairs) are available, the

corresponding aligned translations are attempted in a random order and the first one which improves the placement (if any) is performed.

4.3 Implementation details

Ultimately, the most important aspect of the movement strategy is what works best in practice, which may depend strongly on the problem instance. In the remaining part of this section, we focus on how the aligned translations can be implemented efficiently. We have already seen that the translation algorithm itself can be implemented efficiently, but we have not yet discussed how to find the available contact points and their corresponding vertices and edges. Given a polygon P, the naive approach is to simply make a series of tests for each of its vertices and edges in relation to the edges and vertices, respectively, of all other polygons. This could work quite well, but given the combination of global and local moves sketched above, there is a much better approach to this problem.

As previously described, each translation involves the generation of a set of breakpoints corresponding to the changes in the quadratic function describing the area of overlap between P and the other polygons. If the minimum of this function is found at a breakpoint then it must correspond to a contact point, since breakpoints are only generated for translation distances that bring a vertex and an edge in contact. On the other hand, if a minimum is found between breakpoints then no such contact point can exist. Now, if the breakpoints are reported (cached) as part of the result of a translation then a following aligned translation can be based on the information provided by these breakpoints. If the minimum was found between breakpoints then we already know that no contact points are available, but if the minimum is at a breakpoint then all breakpoints with the same distance value represent all the contact points. The origin of each breakpoint (vertex and edge) must be stored temporarily, but this does not affect the asymptotic running time. In other words, we get the information required for aligned translations at no extra overhead. The only condition is that we must begin with a non-aligned translation to obtain the first set of candidate contact points.

Some problem instances may only involve very short edges, especially if they are approximations of curved shapes. This can be handled with one or both of the following approaches. Firstly, it would in general be an advantage to make the approximations cruder since the running time already depends heavily on the number of edges involved. Secondly, any aligned translations based on very short edges could be extended to some minimum distance — after all it is not required that the shapes are in contact after the translation and it can be thought of as moving in a direction that is tangent to the shape. This could also be thought of as an emulation of approximating the shape using fewer edges. The implementation used for the experiments presented in Section 6 does not extend the translations.

5 Free Orientation

In general, it can be quite difficult to create a solution method which can handle free orientation in nesting problems, but here we describe a very simple way to incorporate it into the solution method described so far. Consider the aligned translations in the previous section and observe the types of contact points illustrated in Figure 6. In each of the first three cases (6a-c) it is possible to rotate P to create a situation similar to the last case (6d), essentially by aligning edges of neighboring polygons. This is basically all that is needed to handle free

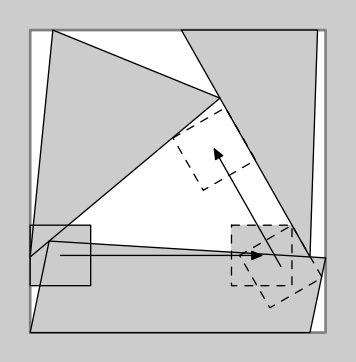
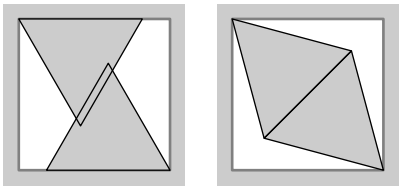


Figure 7: Aligned translations can easily be used to handle free orientation of shapes as well. After an initial axis-aligned translation of the small box, it has obtained a contact point with a neighboring triangle. Based on this contact point, the box is rotated such that it is aligned with the corresponding edges of the triangle before it is translated to its final position.

orientation. In Figure 7 we illustrate how this could affect the example from Figure 5 (involving translations of a small box). The initial horizontal translation works as usual, but before the following aligned translation, the box is rotated around the contact point such that it is aligned with the triangle in the upper right corner. Whenever a polygon is to be rotated as described here, there is a choice to be made, since it can be rotated in two opposite directions. The most obvious strategy (and our choice) is simply to choose the smallest angle since this seems most likely to succeed.

The first interesting observation to be made is that (at least theoretically) this approach to free orientation works in a relatively small search space when compared to the full search space of all possible orientations (and positions) of the polygons. This is due to the fact that the available orientations are based on the angles of edges of all polygons in the initial placement — including combinations of these angles as explained further below. The extreme case is to pack equilateral triangles inside a larger equilateral triangle as container. If the initial orientations of all triangles are identical then the orientations of the triangles are limited to multiples of 60°. Similarly, packing rectangles inside a larger rectangle limits all orientations to multiples of 90°. This is both a useful property (in terms of restricting the size of the search space) and a limitation of this approach. Consider, for example, the situation with initial placement as shown in Figure 8a. No matter what happens during the solution process when using rotation combined with aligned translations, one can never find a non-overlapping placement such as the one shown in Figure 8b. In general, given a problem instance with edges with orientations $\{\alpha_1, \ldots, \alpha_n\}$ then through an appropriate finite number of steps, any edge can obtain any orientation that can be constructed as a scalar combination of the input orientations, i.e., any orientation $\alpha = s_1\alpha_1 + \ldots + s_n\alpha_n \mod 2\pi$ for $s_i \in \mathbb{Z}$. Clearly, even a small number of orientations in the input problem can allow a very large (possibly infinite) number of possible orientations during the solution process.

Note that it is also possible to "lose" possible orientations during the solution process. For example, assume we are given a problem with rectangular shapes and a rectangular container, but with initial orientations such that the rectangles are not aligned with the container. These orientations can then disappear if all rectangles are aligned with the container (directly or



(a) (b)

Figure 8: (a) Two identical equilateral triangles placed inside a square shaped container. (b) This non-overlapping placement cannot be found if only doing rotations in conjunction with aligned translations.

indirectly) during the solution process. When all rectangles are aligned with the boundaries of the container then there is no way of recovering the original orientations. In this case and many other cases, however, this is most likely an advantage. The described approach to free orientation can be interpreted as a dynamic set of discrete orientations which can both grow and shrink during the solution process.

Early experiments with this solution method show that it is important that the edges of the container polygon are included when considering contact points for rotation. This helps keep the polygons aligned with the boundary instead of drifting towards less promising orientations. Another early observation has been that rectangular (or close to rectangular) shapes are best kept close to an orientation aligned with the edges of the container given that the container is also a rectangle which is the case for most benchmark problem instances. These are examples of small changes to the solution method which have a great impact on the results produced, but which are also closely related to the problem instances at hand.

In the previous sections, it has been noted that both the original axis-aligned translation algorithm and the directional translation algorithm can work using rational numbers only. In general this is not true when also using rotation. However, given the right set of initial conditions, it is still possible to use rational numbers only. We have already assumed that the vertices of the polygons have rational vertex coordinates. In order to make rotation work with rational numbers, the lengths of the edges of the polygons must also be rational numbers.

Any polygon can be arbitrarily closely approximated by a polygon with rational coordinates for its vertices and rational edge lengths, by the construction of *rational Heronian triangles*. A rational Heronian triangle is a triangle whose edge lengths and area are both rational. This implies that the triangle can be positioned so that its vertices have rational coordinates. By finding solutions to a suitable Diophantine equation, any triangle can be approximated arbitrarily closely by a rational Heronian triangle. This is demonstrated by Murasaki [19] for an equilateral triangle, and the method can be easily generalized. See also Campbell and Goins [3] for more details.

This approach can easily be extended to polygons, by triangulating the polygon, and successively replacing the component triangles by rational Heronian triangles. Again this allows one to construct a polygon whose vertices have rational coordinates and whose edge lengths are rational, that closely approximates the original polygon in the problem.

Starting with such polygons means that one can run the algorithm without having to use floating point approximations of real numbers, and hence with no floating point errors. This is particularly valuable for testing and debugging the program.

6 Computational Experiments

The two-dimensional aligned translation algorithm and the corresponding rotation technique described in this paper has been incorporated with the existing nesting program, Nest2D, described by Egeblad et al. [7, 8]. Initial experiments with this implementation are presented in this section, but they are not to be considered conclusive as to determining the full potential of the techniques described. The main purpose is to show that this approach is competitive with existing solution methods although results from the literature is very limited with respect to free orientation. All experiments are with the strip packing variant of the nesting problem, i.e., the goal is to minimize the length of a rectangular strip. This is done by repeatedly solving the decision problem, each time making the strip a bit shorter [8]$^{\mathrm{A}}$.

All experiments were conducted on a system with a 2.16 GHz Intel Core Duo processor with 2 MB of level 2 cache and 1 GB of RAM. The problem instances can be found on the ESICUP homepage[1]. All solutions found are verified using Cgal [11], that is, it is verified that no polygon is involved in overlap with other polygons or the container. Due to the limited precision of floating point calculations, very small overlaps can occur, but Nest2D has proven to be very robust. For the experiments done for the results presented in this section, the largest total area of overlap in a solution corresponds to 1×10^{-13} of the total area of all polygons for the given problem.

Using a large number of problem instances, we have conducted a series of experiments with and without free orientation. Experiments without free orientation have been conducted both with and without shape aligned translations. Preliminary experiments have shown that the initial orientations of the shapes in the problem instances used are most often very good orientations for finding compact solutions — especially for the problem instances with shapes that are close to being rectangular. Therefore, experiments have also been conducted where all shapes are initially randomly oriented. Each combination of problem instance and parameters has been run 10 times and the average utilization and the standard deviation is reported in Table 1.

The results vary between problem instances, but the variations are quite small. For fixed orientations of shapes, there is (on average) no difference between using axis aligned translations and also using shape aligned translations. The results when using random initial orientations of the shapes show a considerable decrease in utilization. On average it is 7.6% percentage points worse. Again, there is no difference between using axis aligned translations and also using shape aligned translations. Both have an average utilization of 70.2%.

In Table 2 results with and without free orientation allowed are compared. Interestingly, some results with free orientation of shapes are worse than when using fixed orientations (without initial random orientations). Again, this is most likely because some of these problem instances are best suited for a limited set of rotation angles and allowing anything else is a waste of time. On average, the results obtained with free orientation is 2.1% percentage points better than for fixed orientation. When comparing the results for initial random orientations with and without free orientation in the solution method, the difference is almost 10 percentage points on average. This could be a good indication of what results can be obtained for problem instances which really benefit from free orientation.

The last two columns of the table describes the best results from the literature where 90° or 180° rotations are allowed for most of the problem instances [1, 8]. These results are on

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

				Random initial orientation	
	Size	Axis	Shape	Axis	Shape
Albano	14	85.4 (0.3)	85.1 (0.2)	74.3 (0.8)	74.2 (1.0)
Dagli	30	82.6 (0.4)	83.0 (0.3)	77.1 (0.5)	77.2 (0.6)
Jakobs1	25	83.3 (0.9)	82.9 (1.0)	72.6 (1.2)	72.5 (1.5)
Jakobs2	25	76.5 (0.5)	76.5 (0.7)	67.9 (0.6)	68.2 (0.6)
Mao	20	79.4 (0.1)	79.4 (0.2)	72.0 (1.2)	72.2 (1.9)
Marques	24	87.4 (0.1)	87.2 (0.4)	75.4 (0.6)	75.1 (0.7)
Poly5b	75	72.3 (0.3)	72.7 (0.6)	72.3 (0.7)	72.4 (0.5)
Profiles6	65	73.4 (0.9)	73.2 (0.8)	57.7 (0.6)	57.5 (0.7)
Profiles10	91	66.7 (0.4)	67.0 (0.7)	63.4 (0.6)	64.0 (0.6)
Shapes0	43	66.0 (0.5)	65.9 (0.5)	60.5 (0.8)	60.7 (0.4)
Shirts	99	84.5 (0.3)	84.2 (0.4)	77.5 (0.5)	77.1 (0.6)
Swim	48	69.0 (0.5)	69.0 (0.7)	65.7 (0.8)	65.2 (0.5)
Trousers	64	85.3 (0.4)	85.3 (0.3)	75.4 (1.5)	75.8 (1.6)
		77.8 (7.4)	77.8 (7.3)	70.2 (6.3)	70.2 (6.3)

Table 1: Experimental results from Nest2D showing the utilization percentages (average of 10 runs) obtained for a range of different problem instances from the existing literature. The number of shapes are reported for each problem instance in the second column. Column 3 and 5 are the results when using axis-aligned translations only. Column 4 and 6 are the results when also using shape-aligned translations. Column 6 and 7 are the results when the shapes are initially randomly oriented. The final row is an average over all problem instances.

		Random initial orientation		Best from literature	
Orientation:	Fixed	Fixed	Free		
Albano	85.1 (0.2)	74.2 (1.0)	85.1 (0.3)	86.9 (0.3)	180°
Dagli	83.0 (0.3)	77.2 (0.6)	85.0 (0.5)	85.3 (1.1)	180°
Jakobs1	82.9 (1.0)	72.5 (1.5)	85.0 (0.8)	88.9 (0.4)	90°
Jakobs2	76.5 (0.7)	68.2 (0.6)	79.5 (0.6)	80.2 (0.2)	90°
Mao	79.4 (0.2)	72.2 (1.9)	81.0 (0.8)	82.6 (0.9)	90°
Marques	87.2 (0.4)	75.1 (0.7)	86.3 (0.6)	88.7 (0.3)	90°
Poly5b	72.7 (0.6)	72.4 (0.5)	77.2 (0.5)	75.8 (-)	90°
Profiles6	73.2 (0.8)	57.5 (0.7)	74.8 (1.1)	75.6 (-)	0°
Profiles10	67.0 (0.7)	64.0 (0.6)	69.4 (0.5)	66.2 (-)	0°
Shapes0/1	65.9 (0.5)	60.7 (0.4)	73.8 (0.8)	71.7 (0.8)	180°
Shirts	84.2 (0.4)	77.1 (0.6)	84.3 (0.7)	85.7 (0.4)	180°
Swim	69.0 (0.7)	65.2 (0.5)	69.1 (0.8)	72.2 (1.0)	180°
Trousers	85.3 (0.3)	75.8 (1.6)	87.8 (0.5)	89.2 (0.3)	180°
	77.8 (7.3)	70.2 (6.3)	79.9 (6.2)	80.7	

Table 2: Experimental results from NEST2D showing the utilization percentages (average of 10 runs) obtained for a range of different problem instances from the existing literature. Standard deviations in parenthesis. Parameters have been varied in order to show the difference in results with/without free orientation and with/without a random initial orientation of shapes. The final two columns are the best results found in the literature where 90° or 180° rotations are allowed (no rotation is allowed for two of the problem instances). The final row is an average over all problem instances.

	Liu and He [16]			Nest2D		
Problem	Length	Utilization (%)	Seconds	Length	Utilization (%)	Seconds
Poly5b	60.12	76.3	37.6	59.40	77.2	600
Shapes0	56.9	70.1	4082.0	54.02	73.8	600
Swim	6320.7	69.9	156.5	6398.9	69.1	600

Table 3: Results by Liu and He [16] compared with results produced by NEST2D (averages of 10 runs). Both solution methods allow free orientation of shapes. When comparing running times, it should be taken into consideration that Liu and He conducted their experiments on a 1.5 GHz Pentium IV processor.

average better than those with free orientation, but there are a few exceptions. This is a strong indication that even though the solution method presented in this paper can handle free orientation of shapes, it might be better for some problem instances to use a restricted set of rotation angles. It would of course be better if the solution method could be improved in such a way that it performed at least as well as a solution method specialized for a small number of rotation angles.

The existing literature only contains few computational results with respect to free orientation of shapes, but a small comparison is possible with the results reported by Liu and He [16] for their solution method. They report both the lengths of their solutions and the utilization of material. Unfortunately, the utilization values do not match with the definition of utilization used in this paper and it is unclear what their method is. Therefore, we have computed utilization values based on the lengths reported by Liu and He. Their results are compared with the average results produced by NEST2D of 10 runs of 10 minutes each (see Table 3). Liu and He handle the Swim problem instance slightly better with a difference of 0.7 percentage points while the opposite is the case for the Poly5b problem instance which is 0.9 percentage points better with NEST2D. The largest difference is NEST2D being 3.7 percentage points better on the Shapes0 problem instance. Although the comparison with Liu and He is very limited, it does indicate that NEST2D works at least as well as other solution methods in the existing literature.

7 Generalization to higher dimensions

The implementation used in the previous section currently only applies to problems in two dimensions. However, theoretically the techniques described can be generalized to three and more dimensions. The following is a short discussion of some of the details of such a generalization, but it is still an open question as to how well this would work in practice.

Egeblad et al. [8][A] generalizes the axis-aligned translation algorithm to three dimensions, and a more detailed description (and proof) of this generalization is given in Egeblad et al. [7][C]. The latter paper also generalizes the result to arbitrary higher dimensions. It is therefore interesting to discuss the applicability of the techniques described in this paper to problems in three or more dimensions.

The d-dimensional generalization of a polygon is a *polytope*. Often in the literature it is assumed that a polytope is a convex region of d-dimensional space, defined, e.g., by a finite set of halfspaces, but here we allow a broader definition. We define a polytope as any shape

that can be described as the finite union of a set of convex polytopes. The boundary of a polytope is composed of convex faces of varying dimensions ($< d$); the $(d-1)$-dimensional faces are called *facets*, the 1-dimensional faces *edges*, and the 0-dimensional faces *vertices*. The details are given in Egeblad et al. [7]C.

A generalization of the directional translations is straightforward. As in the two-dimensional case, basically only three changes are needed which can be briefly sketched as follows:

1. Given a direction, one must find an axis-aligned translation direction with which to equate this direction (via projection). A good choice, with respect to precision, is the axis-aligned direction which is closest to the given direction.

2. When determining the overlap of two $(d-1)$-dimensional facets in the translation direction, the projections needed are no longer axis-aligned. Fortunately, as in the two dimensional case, projections in arbitrary directions are straightforward to compute.

3. The result of the translation algorithm only provides a single coordinate (on the corresponding chosen translation axis) of the optimal position for the polytope. Using the previous position of the polytope and the translation direction, the other coordinates are easily derived.

The difficult question is that of how to choose good translation directions, or more precisely: what is a good local search neighborhood for the movement strategy. To answer this question, we return to the concept of contact points. A contact point generalizes nicely to higher dimensions. We say that the point p is a *contact point* for P and Q if p lies on the boundaries of both polygons and p lies on a face of dimension $\leq d-2$ of either P or Q. More types of contact points exist in higher dimensions, but they can all be represented by a pair of faces: the facet f containing p for one polytope, and the face f' with smallest dimension (no greater than $d-2$) containing all contact points in a small neighborhood around p for the other polytope. In relation to aligned translations, this limits the translation direction to directions that keep all contact points belonging to the face f' on the facet f. If only one contact point exists (in other words, f' is a vertex) then any one of these directions could be chosen, but a natural choice might be one that keeps the direction as axis-aligned as possible. If, on the other hand, there are exactly two discrete contact points available and the facets corresponding to these contact points are not in the same hyperplane, then, $d \geq 3$, it is possible to move the shape in a direction which respects both contact points. In three dimensions this direction is unique, but in higher dimensions there is some choice, and we could again choose the direction that is closest to being axis-aligned. If there are infinitely many contact points (for example, an edge lying across a 2-dimensional facet in 3-dimensional space) then again we can choose a translation direction closest to being axis aligned that respects all contact points.

To summarize, a good local search neighborhood in d-dimensional space would contain all translation directions obtained from contact pairs (f, f'), where a contact pair consists of a facet from one polygon and a lower dimensional face from the other corresponding to one or more contact points.

The final question to answer is how to handle rotation in arbitrary dimensions. In this case, we limit the discussion to the three-dimensional case. First, assume that a single isolated contact point p is available corresponding to a vertex f' of one polytope and lying on a facet f of the other. As usual we denote these polytopes P and Q, where P is to be translated, but

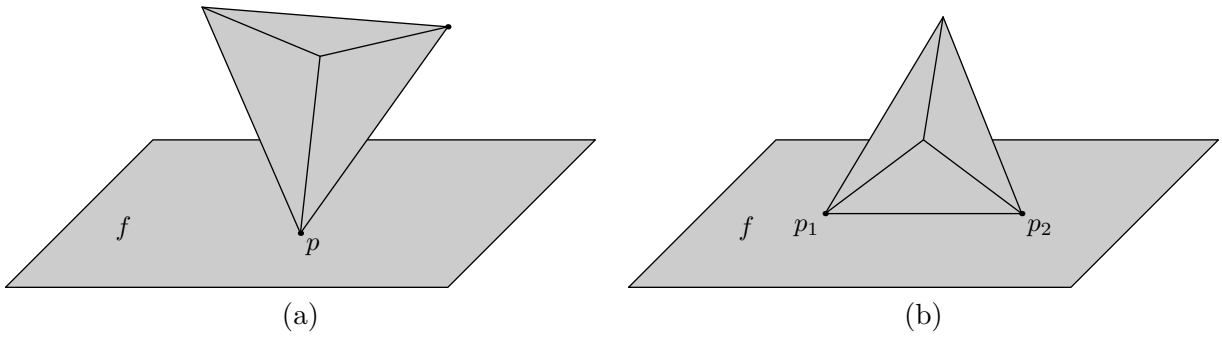


Figure 9: (a) A tetrahedron with a single contact point given by a point p on the tetrahedron and a face f from some other polytope. There are numerous ways to rotate the tetrahedron such that the contact point is fixed and one of three faces of the tetrahedron is aligned with f. (b) A continuum of contact points (corresponding to the edge (p_1, p_2) which is in contact with facet f) limits the freedom of the rotation such that only two options are available if the edge (p_1, p_2) is fixed.

we do not know which polytopes contain of f' and f respectively. The vertex f' is a corner point of a set of faces of its corresponding polytope. Each of these faces is a candidate for a rotation of P such that the face is aligned with f (see Figure 9a). A natural choice would be the candidate which requires the smallest rotation angle. If, on the other hand, a continuum of contact points are available for the same facet f them we can assume that f' is an edge bounding exactly two faces of the polytope. Rotations that keep this edge fixed on the face f are obvious candidates and again it seems most natural to try the one which requires the smallest rotation angle to align the face with f (see Figure 9b). A third possibility is that there are two or more isolated contact points on f that do not belong to the same edge of the other polytope, but even in this case, this pair of contact points can often still help to identify a useful axis of rotation.

8 Conclusion

We have presented a new approach to solving the nesting problem with free orientation of arbitrary polygonal shapes. This solution method is based on a translation algorithm which can find the minimum area of overlap position in a given direction. Such translations are done repeatedly until the total overlap has been reduced to zero and thus a non-overlapping solutions has been found. The direction and the orientation used for each translation is based on contact points with neighboring shapes. This makes the solution method biased towards parts of the search space with a large number of contact points and a large number of aligned edges of neighboring polygons. This is at least a well suited approach for solving jigsaw puzzles, but it can also be a limitation since other parts of the search space are ignored.

Computational experiments have shown that this solution method works very well in practice. Without free orientation, the aligned translations work almost as well as the alternative of using simple axis-aligned translations. With free orientation, the utilization of many problem instances is improved although it is not always the case. This is mainly due to the fact that some of the problem instances in the literature have irregular polygonal shapes which

are really close to being rectangular. The implementation of aligned translations (and free orientation) has not been optimized to the same degree as the basic axis-aligned translation method. It is therefore not unlikely that the full potential of this approach has not been fully illustrated by the computational experiments.

Each of the techniques described in this paper could most likely also be used for other solution methods for the nesting problem and even help solve completely different problems.

References

[1] E. 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.

[2] J. Cagan, K. Shimada, and S. Yin. A survey of computational approaches to three-dimensional layout problems. *Computer-Aided Design*, 34(8):597–611, 2002.

[3] G. Campbell and E. H. Goins. Heron triangles, diophantine problems and elliptic curves. URL `http://www.swarthmore.edu/NatSci/gcampbe1/papers/heron-Campbell-Goins.pdf`. Preprint.

[4] A. Crispin, P. Clay, G. Taylor, T. Bayes, and D. Reedman. Genetic algorithm coding methods for leather nesting. *Applied Intelligence*, 23(1):9–20, 2005.

[5] 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.

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

[7] J. Egeblad, B. K. Nielsen, and M. Brazil. Translational packing of arbitrary polytopes. Submitted.

[8] 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.

[9] 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.

[10] 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.

[11] P. Hachenberger and L. Kettner. 3D boolean operations on nef polyhedra. In C. E. Board, editor, *CGAL-3.2 User and Reference Manual*. 2006. URL `http://www.cgal.org/Manual/3.2/doc_html/cgal_manual/packages.html#Pkg:Nef3`.

[12] J. Heistermann and T. Lengauer. The nesting problem in the leather manufacturing industry. *Annals of Operations Research*, 57:147–173, 1995.

[13] S.-M. Hur, K.-H. Choi, S.-H. Lee, and P.-K. Chang. Determination of fabricating orientation and packing in sls process. *Journal of Materials Processing Technology*, 112(2-3):236–243, 2001.

[14] P. Jain, P. Fenyes, and R. Richter. Optimal blank nesting using simulated annealing. *Journal of Mechanical Design*, 114(1):160–165, 1992.

[15] 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.

[16] H.-y. Liu and Y.-j. 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.

[17] V. J. Milenkovic. Rotational polygon overlap minimization and compaction. *Computational Geometry*, 10:305–318, 1998.

[18] V. J. Milenkovic. Rotational polygon containment and minimum enclosure using only robust 2D constructions. *Computational Geometry*, 13:3–19, 1999.

[19] T. Murasaki. On the heronian triple (n+1, n, n-1). *Sci. Rep. Fac. Educ., Gunma Univ.*, 52:9–15, 2004.

[20] B. K. Nielsen and M. Brazil. A toolbox of geometric overlap measures for solving nesting problems. In Preparation.

[21] G. Wäscher, H. Haussner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 2006. In Press.

Steiner Tree Problems

1 Introduction

In the introduction to nesting problems it was stated that cutting and packing problems are classic subjects in computer science. More precisely, the first papers were written on the subject in the 1950's [15]. *Steiner tree* problems are classic subjects in mathematics, but this is in a completely different time frame.

The first stated Steiner tree problem was the Euclidean Steiner tree problem for an input of 3 points. It was stated by Fermat (1601-1665) and is also known as the *Fermat problem*. In short, this problem asks for a point in the plane that minimizes the sum of the distances to three given points. This problem can, in a natural way, be generalized to finding a point that minimizes the sum of distances to n given points. Although the mathematician Steiner (1796-1863) was interested in this problem, it is not the problem that is generally known as the Steiner tree problem. The Steiner tree problem is actually due to Jarník and Kössler, who in 1934 stated the following problem in a Czechoslovakian journal: For a given set of points, find the shortest network in the plane that connects them. This was later referred to as the Steiner tree problem by mistake and it has since become the standard name [7].

Steiner tree problems come in numerous variants and a good introduction to solution methods (both exact and heuristic) is given by Hwang et al. [7]. The practical applications of Euclidean Steiner trees are quite obvious. A network with, e.g., pipelines, electricity wires or telephone cables, is going to be cheaper if it is shorter. Here, we first give a general introduction to the subject of Steiner trees which is then followed by a discussion of the problem variants relevant for the four Steiner tree related papers in this thesis. These problem variants are based on two unrelated applications; *VLSI design* (Section 2) and construction of *phylogenetic trees* (Section 3). Finally, a short comparison and discussion of these applications is given in Section 4.

Steiner trees are closely related to the (also classic) subject of spanning trees; in both cases the standard problem is to find a tree with minimum length. A minimum length tree is known as a *Steiner minimum tree* (SMT) and a *minimum spanning tree* (MST) for the respective variants. The abbreviations are slightly confusing, but they are the de-facto standard in the existing literature. These problems come in two basic variations; graph problems and metric problems.

Consider a graph G with vertices V and undirected edges E where the edges have positive weights. Also assume that the graph is connected, i.e., there is a path between any pair of vertices. A minimum spanning tree problem can now be defined as the problem of finding a set of edges in G with minimum sum of weights such that the edges connect all vertices in the graph. Obviously such a set of edges is a tree. This is a polynomially solvable problem. Now, consider a subset of vertices, $P \subset V$, and consider the problem of finding a tree with minimum sum of weights connecting the vertices in P (denoted *terminals*). The vertices in such a tree which are not terminals are denoted Steiner vertices. This is the Steiner tree problem for graphs and this small change in problem definition makes it an $\mathcal{NP}$-hard problem.

The classic metric problem definitions are for the Euclidean plane. Given a set of points, again denoted *terminals*, in the Euclidean plane, a minimum spanning tree is a shortest tree spanning all of the terminals where each edge in the tree connects pairs of terminals. This is illustrated in Figure 1a. If additional points, denoted *Steiner points*, are permitted in the

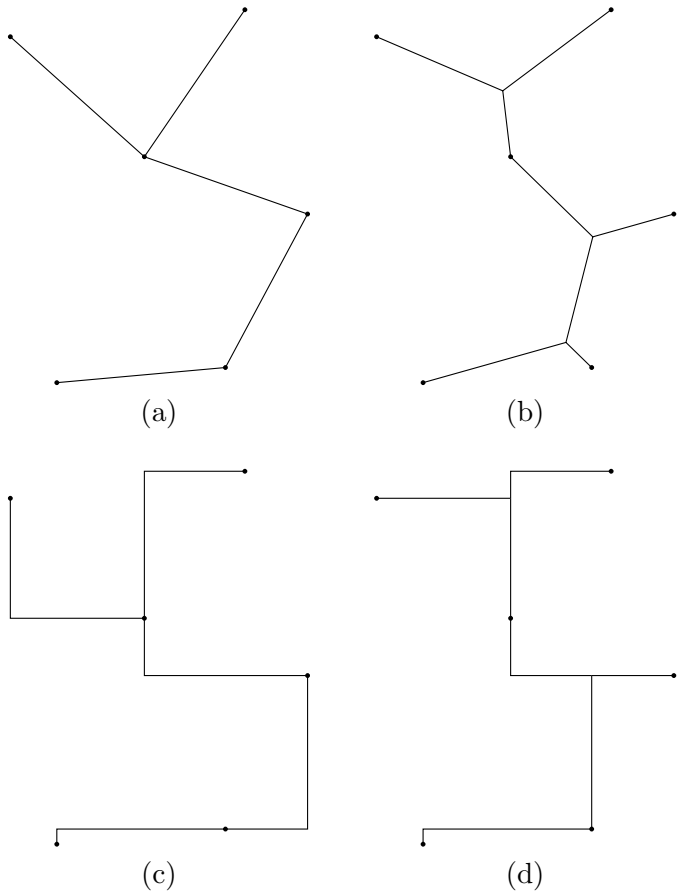


Figure 1: Various tree construction solutions for 6 points in the plane. (a) Euclidean minimum spanning tree (EMST). (b) Euclidean minimum Steiner tree (ESMT). (c) Rectilinear minimum spanning tree (RMST). (d) Rectilinear minimum Steiner tree (RSMT).

construction of the tree then the problem, again, becomes $\mathcal{NP}$-hard. Such as tree is known as the Euclidean Steiner minimum tree which is most often abbreviated SMT (Figure 1b). Here we need to distinguish it from similar problems and thus we denote it ESMT.

2 VLSI

The rectilinear minimum Steiner tree is a problem variation for the rectilinear metric (Manhattan distance). Such a tree is always at least as long as the Euclidean variant. A rectilinear MST and a rectilinear SMT are illustrated in Figure 1c-d. They problems are abbreviated RMST and RSMT, respectively. Applications and a further generalization is part of Section 2.

Note that the metric MST problems can also be formulated as graph problems by first constructing a complete graph with edge weights corresponding to the distances between pairs of terminals. Something similar is possible for the RSMT problem using a so-called Hanan-grid which is a grid obtained by having horizontal and vertical lines through all vertices. The intersections of these lines are then a sufficient set of potential Steiner points for an optimal solution. The ESMT problem is special in the sense that it is not possible to (easily) predict

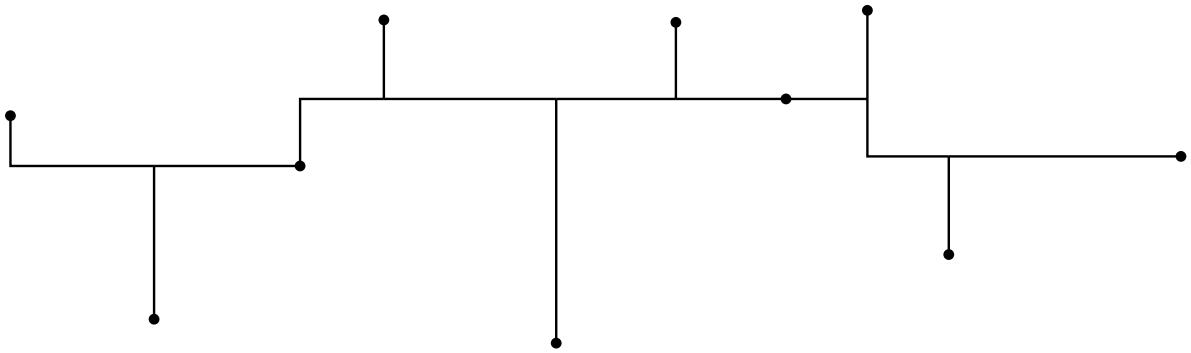


Figure 2: A rectilinear Steiner minimum tree which contains 3 FSTs. Each FST is a tree with only Steiner points as interior nodes.

a set of potential locations of Steiner points in an optimal solution.

Interestingly, the last paper in this thesis (Brazil et al. [1]$^{\mathrm{H}}$) is an attempt to solve what is essentially a Steiner graph problem by using a heuristic method for a Euclidean Steiner tree problem.

Even though both ESMT and RSMT are $\mathcal{NP}$-hard problems they can often be solved exactly even for very large problem instances — including some problem instances with more than 10000 terminals [16]. The current state-of-the-art solution method is based on a two-phase model. In the *generation phase* a small set $\mathcal{F}$ of so-called *full Steiner trees* (FSTs) are generated while ensuring that some union of a subset of $\mathcal{F}$ is an ESMT/RSMT for the entire set of terminals. Finding this subset of FSTs is called the *concatenation phase*. While the concatenation phase is essentially the same in the Euclidean and the rectilinear Steiner tree problems, the generation phases are quite different. Figure 2 is an example of an RSMT with three FSTs.

The standard application of rectilinear Steiner trees is in the industry of VLSI (very-large-scale integration) design. The term VLSI was originally used in the 1970s to describe integrated circuits created by combining thousands of transistor-based circuits on a single chip; e.g., a microprocessor. Today chips are produced with more than a billion transistors. In relation to Steiner trees, the interesting part is the wiring on the chip. Traditionally, routing has been restricted to horizontal and vertical directions, but in recent years there has been an increasing interest in using more directions, in particular, diagonal directions. Diagonal routing can reduce wire length with an average of 20% and thus make chips considerably faster[1]. At the same time the space taken by the wiring is reduced. Each direction is in its own layer on a chip which explains why a large number of directions is not feasible.

The above motivates a generalization of the Steiner tree problem. Suppose that we are given a set P of n points in the plane. We are interested in finding a λ-Steiner minimum tree (λ-SMT) for P which is an SMT with the restriction that the edges are permitted to have only a limited number $\lambda \geq 2$ of equally-spaced orientations. We assume that the permissible orientations are defined by straight lines making angles $i\omega$, $i = 0, 1, \ldots, \lambda-1$, with the positive x-axis, where $\omega = \pi/\lambda$. Examples of the edge orientations available for various values of λ are illustrated in Figure 3.

[1]X Initiative Home Page: `http://www.xinitiative.com`

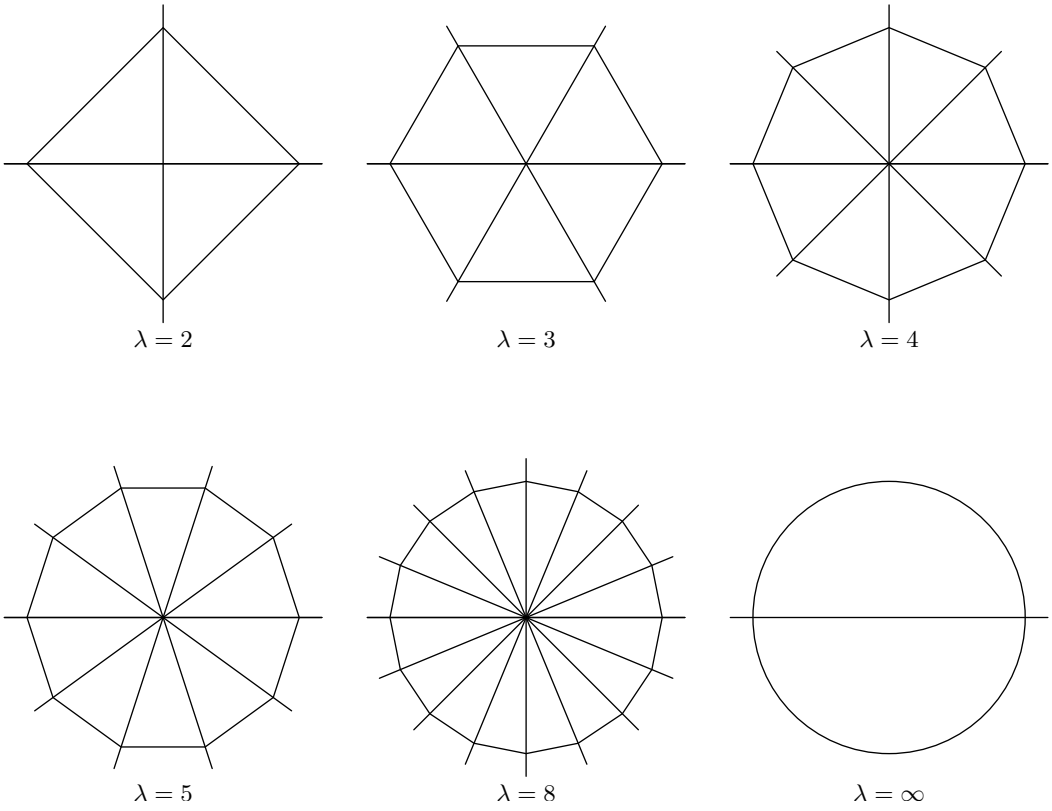
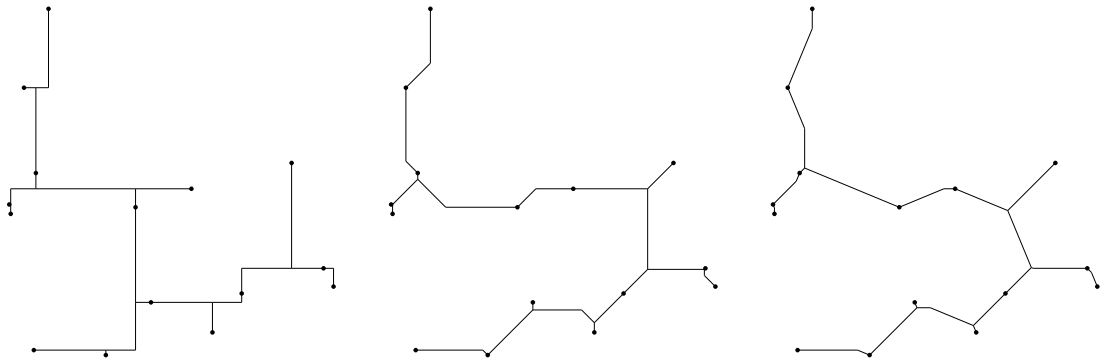


Figure 3: The unit disks corresponding to various values of λ. Here, ∞ is used to symbolize the Euclidean metric which can be interpreted as an infinite number of available orientations.

The rectilinear Steiner tree problem is the problem of finding a 2-SMT, i.e., $\lambda = 2$. We also say that we want to solve SMT problems in uniform orientation metrics. Examples of λ-SMTs for three different values of lambda for the same set of points are shown in Figure 4.



λ=2	λ=4	λ=8
6 FSTs. Length: 28094.00	7 FSTs. Length: 25724.63	9 FSTs. Length: 24952.03

Figure 4: λ-SMTs with varying λ for the same set of 15 points. Note that the topologies are *not* identical.

Brazil et al. [2][F] presented an important result on the location of Steiner points for Steiner minimum trees in uniform orientation metrics for $\lambda = 3m$ for an integer $m \geq 1$; they show that the Steiner points can be placed in the exact same positions as in the Euclidean case. Using this result one easily obtains a linear time algorithm for constructing a λ-SMT when given a full topology by applying the corresponding algorithm for the Euclidean case [6] and then replace the edges with the shortest connections possible in the uniform orientation metric. This result was later extended to arbitrary values of λ by Brazil et al. [3]. When not given the topology, the problem can be solved with the same two-phase strategy as for the rectilinear and Euclidean cases. Again, the concatenation phase is the same while there are major differences in the construction phase. Details and experimental results are reported by Nielsen et al. [9].

Increasing the number of orientations is one way to obtain shorter wire lengths in VLSI design. Another interesting technique is presented in the paper by Nielsen et al. [8][E] which is based on the observation that the length of a λ-SMT changes if the set of terminals is rotated; or equivalently if the set of allowed orientations is rotated. A small example is shown in Figure 5. The main problem is to find the rotation angle resulting in the shortest λ-SMT. They show that a finite set of candidate angles can be found and computational experiments are done showing what the gain is for RMST and RSMT problems.

3 Phylogenetic Trees

A completely different application of Steiner trees can be found in computational biology. Given a set of (existing) species, an important problem is to determine how the species are related. The relationships are often described by a tree in which the species are located in the leaves. Interior nodes represent hypothesized ancestors and in some cases the branch

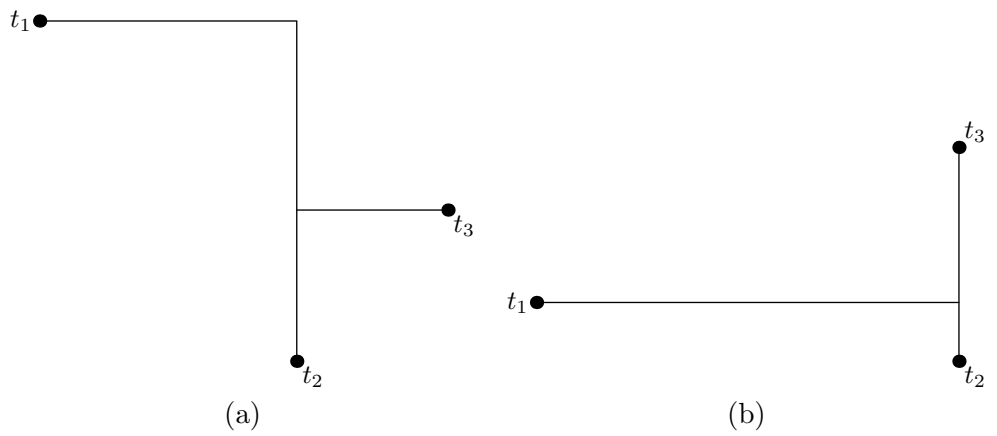


Figure 5: Three terminals in the rectilinear metric. (a) Non-optimal orientation for the length of the RSMT. (b) Orientation minimizing the length of the RSMT.

lengths correspond to some measure of evolutionary distance. Such a tree is known as a phylogenetic tree. In a sense, a shortest tree can be assumed to be a very likely tree to explain the relationships among a set of species. Further below, we show how this can be stated as a Steiner tree problem for a graph.

The whole idea of species being related to each other through evolution was first described by Charles Darwin in the 19th century [4]. As mentioned such relationships can conveniently be represented by trees as also suggested by Darwin (at the end of this introduction we discuss some limitations of this approach).

Historically, various kinds of input has been used in order to construct phylogenetic trees, e.g., physical characteristics such as the number of legs or the ability to fly. Today, most kinds of analyses of the interrelationship of species are based on their DNA (or RNA). For the purposes here, we are simply going to assume that a species is represented by a sequence of the letters (nucleotides) `A`, `T`, `C` and `G`.

Solution methods for obtaining a phylogenetic tree based on a set of sequences come in many variations. Books on the subject include the work of Felsenstein [5] and Semple and Steel [13]. In this introduction, we are only going to discuss the approaches relevant for the subsequent papers.

Some of the most simple solution methods are the so-called distance-based approaches. A prerequisite of this approach is a matrix of distance values. This can be obtained from DNA sequences by computing an *edit distance* between pairs of sequences, i.e., it is measured how many steps (or mutations) which are most likely to have occurred in the evolution from one sequence to another sequence. Given two equal length sequences, the simplest way to calculate a distance is to calculate the *Hamming distance*; counting the number of differences. More sophisticated evolutionary models are needed in order to more closely emulate the mutations occurring in nature (and to handle sequences of differing length).

One of the classic and very simple methods for deriving an unrooted phylogenetic tree from a distance matrix is the *neighbor joining* (NJ) algorithm [11]. A description can be found in Brazil et al. [1]$^{\text{H}}$. In short, the algorithm repeatedly merges two species which are characterized by being close to each other while being far away from the other species. Each pair is, in subsequent iterations, represented as a species itself by computing averages

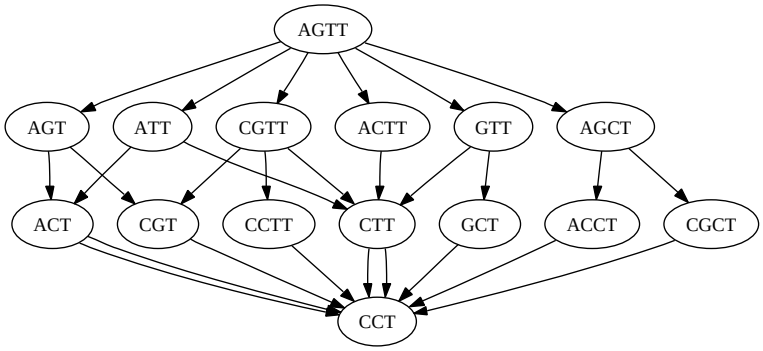


Figure 6: Two DNA sequences, `AGTT` and `CTT`, and the possible three step mutation sequences possible to explain their relationship.

of distances to the other species.

Brazil et al. [1][H] suggest an approach for constructing phylogenetic trees which transforms a distance matrix into a problem which can be solved using Euclidean Steiner trees. This is done by using *multi-dimensional scaling* to place the points in d-dimensional Euclidean space such that the distances in the distance matrix match the Euclidean distances as closely as possible. After this transformation a heuristic is used to approximate the d-dimensional ESMT and the resulting topology is then used to construct a phylogenetic tree.

Multi-dimensional scaling is an example of how to use a metric Steiner tree problem for the construction of a phylogenetic tree. Alternatively, the DNA sequences can be handled more directly. The distances discussed above are an attempt to measure the number of mutations which has occurred when one DNA sequence has changed into another DNA sequence. For simplicity, assume that only three kinds of mutations can occur. A nucleotide can change, e.g., from `A` to `G`, a nucleotide can be deleted or a nucleotide can be inserted. Given two simple sequences, `AGTT` and `CTT`, numerous explanations of the series of mutations necessary to change one in to the other exist. Examples of all possible variations with three mutations are shown in Figure 6. In general, it is reasonable to assume that a short path is more likely to be closer to the correct explanation than a long path (*maximum parsimony*). Although that does not mean that the shortest path is necessarily the correct explanation.

Now, consider a graph in which the nodes are all possible DNA sequences and the edges are all possible single event mutations. Weights on the edges can be used to make some mutations more or less likely than other mutations. This is an infinite graph (an upper limit on the length of the DNA sequences makes it finite). An example of a small neighborhood of a short DNA sequence is shown in Figure 7. Given DNA sequences from a set of species, each of them correspond to a node in the graph. The conjecture then is that a shortest Steiner tree in the graph — connecting all the species — is also a good phylogenetic tree.

In practice, the DNA graph is too big to construct explicitly for realistic problem instances. An alternative heuristic approach is described by Nielsen et al. [10][G]. Note that the heuristic was originally suggested by Schwikowski and Vingron [12] and it is known as the deferred path heuristic (DPH). It is similar to the distance based methods with respect to the sequence of merges, but instead of calculating a new set of distances after merging a pair of species, a *sequence graph* representing a large set of possible ancestor sequences is constructed. When all sequences have been merged, one can select a sequence in the final sequence graph and then go

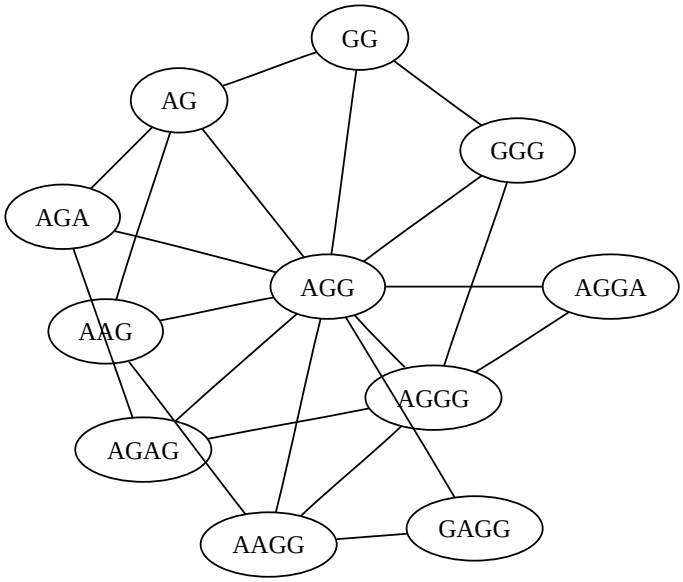


Figure 7: A sequence `AGG` and all its neighbors if only simple mutations are allowed (single change, deletion or insertion of a nucleotide). The alphabet has been limited to `A` and `G` for simplicity.

back through the intermediate sequence graphs in order to construct an alignment of all the initial sequences, i.e, insert gaps to make all sequences of equal length. This is a (heuristic) solution to the *generalized tree alignment problem*. If the phylogenetic tree is already known then one can also use this solution method for just constructing a *tree alignment* based on the phylogenetic tree. These problems are also illustrated in Figure 8.

Nielsen et al. [10]^{G} show how to obtain good solutions for generalized tree alignment problems, but they do not compare the resulting trees with other solution methods for constructing phylogenetic trees. After publication of their work, some additional work on the heuristic allowed the following small experiments to be conducted. First, a simple comparison with neighbor joining (NJ) could be done as follows:

1. Artificially generate a phylogenetic tree and sequences for the leaves in the tree [14].
2. Compute pairwise distances and apply NJ to reconstruct the phylogenetic tree.
3. Based on the sequences, reconstruct the phylogenetic tree using DPH.
4. Compare the reconstructed trees (by NJ and DPH) with the original (correct) tree.

The experiment above was done for 10, 20 and 40 sequences with relatedness values of 10 and 50 [14]. The latter is a value describing how similar the sequences are. The results can be seen in Table 1. All results are averages of 10 runs. The basic version of DPH simply makes a greedy choice when selecting the pair to be merged, i.e., the choice is based on the shortest distance between a pair of sequence graphs. It is also possible to integrate the strategy of NJ. Results for this approach is also included in Table 1. The main conclusion is that the phylogenetic trees produced by DPH are not very good when the simple greedy choice is used while the integration of NJ almost works as well as using NJ on pairwise distances. The

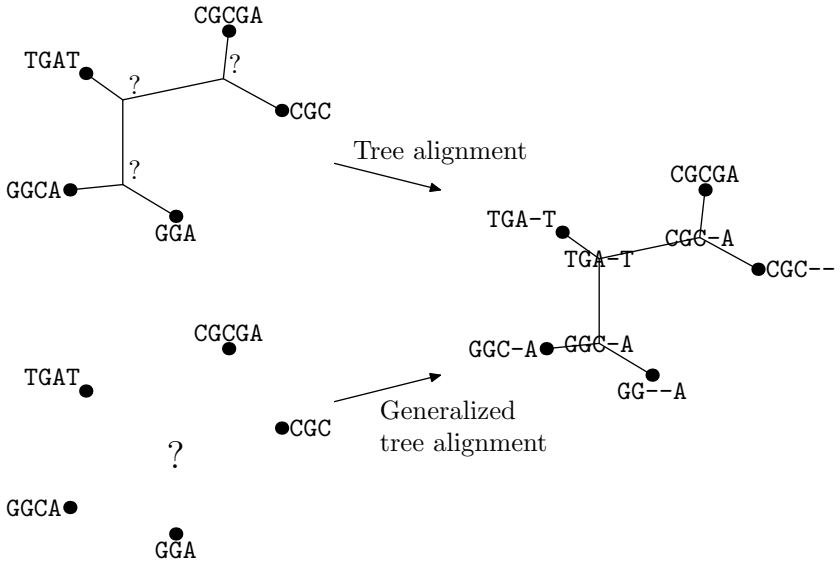


Figure 8: Illustration of two alignment problems: The tree alignment problem and the generalized tree alignment problem. The two problems differ with respect to whether the topology of the tree is known.

interesting question is now which of the methods actually produce the best tree alignments. This turns out to be the integration of NJ in DPH (also shown in Table 1).

4 Conclusions

In this introduction two very different applications of Steiner trees have been illustrated. For a computer scientist, the VLSI application has the nice feature of being straightforward and the link between short trees and fast computer chips is immediate. In practice, there are many other parameters than the wiring length which need to be considered in VLSI design, but that does not change the fact that the shortest tree is always better if all other objectives are ignored.

Phylogenetic trees are not obvious Steiner trees — or any other kind of mathematically well defined trees. They are just similar to Steiner trees and it might or might not be a good idea to construct phylogenetic trees based on algorithms for the construction of Steiner trees. The best approach to evaluate the usefulness of Steiner tress is to make comparisons with other solution methods. But the comparison of phylogenetic tree reconstruction methods is one of the major problems of doing research in the area of computational biology. The best phylogenetic tree is the one that describes the true evolutionary history of species, but there are several major problems with finding such a tree. First of all, it is impossible to define a perfect model of evolution. There are simply too many factors involved. Worse is the fact that the data provided by living species is incomplete. Part of the evolutionary process is the loss of information (deleted nucleotides), so one is actually working on incomplete data. Furthermore, the correct trees are very rarely known which means that it is not even possible to compare solution methods experimentally. Artificial problem instances can be constructed to obtain the knowledge of "correct" trees, but this again requires assumptions about the

	Number of species					
	Similarity			Tree alignment cost		
	10	20	40	10	20	40
Relatedness 10						
DPH	2.40	7.60	18.00	382.30	420.15	863.48
NJ	0.00	1.11	0.53	375.07	397.32	809.05
DPH+NJ	0.00	0.37	0.53	373.68	396.98	809.65
Relatedness 50						
DPH	2.60	8.00	20.40	1096.05	1910.08	3556.22
NJ	0.00	0.00	0.70	1085.92	1875.65	3437.78
DPH+NJ	0.83	0.74	0.88	1079.00	1868.83	3430.85

Table 1: Experiment with reconstructions of phylogenetic trees done by three different heuristics; DPH, NJ and a combination. Sequences of varying length and relatedness are used in the experiment and all entries are averages of 10 runs. Three columns show the similarity with the known correct tree. This value ranges from 0 to 100 where 0 is a perfect match. Three columns show the average tree alignment cost for the same experiments. The number is relative to the number of mutations needed to describe the changes necessary to explain the relations in the tree. Note that the tree alignment costs shown for NJ are the ones obtained by solving the tree alignment problem using DPH.

model of evolution which are most likely not accurate. Finally, the assumption that the evolution of a set of species can be described as a tree is not even necessarily correct. To describe horizontal gene transfers and other phenomena, a data structure more sophisticated than trees is required.

References

[1] M. Brazil, B. K. Nielsen, D. Thomas, P. Winter, C. Wulff-Nilsen, and M. Zachariasen. A novel approach to phylogenetic trees: d-dimensional geometric Steiner trees. *Networks*. In Press.

[2] M. Brazil, B. K. Nielsen, P. Winter, and M. Zachariasen. Rotationally optimal spanning and Steiner trees in uniform orientation metrics. *Computational Geometry*, 29(3):251–263, 2004.

[3] M. Brazil, D. A. Thomas, J. F. Weng, and M. Zachariasen. Canonical forms and algorithms for Steiner trees in uniform orientation metrics. *Algorithmica*, 44(4):281–300, 2006.

[4] C. Darwin. *On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life*. John Murray, London, 1st edition edition, 1859.

[5] J. Felsenstein. *Inferring phylogenies*. Sinauer Associates, Inc., Sunderland, MA, 2003.

[6] F. K. Hwang. A linear time algorithm for full Steiner trees. *Operations Research Letters*, 4(5): 235–237, 1986.

[7] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner tree problem*. Annals of Discrete Mathematics 53. Elsevier Science Publishers, Netherlands, 1992.

[8] B. K. Nielsen, P. Winter, and M. Zachariasen. On the location of Steiner points in uniformly-oriented Steiner trees. *Information Processing Letters*, 83(5):237–241, 2002.

[9] B. K. Nielsen, P. Winter, and M. Zachariasen. An exact algorithm for the uniformly-oriented Steiner tree problem. *Algorithms - ESA 2002: 10th Annual European Symposium, Rome, Italy, September 17-21*, pages 237–266, 2002.

[10] B. K. Nielsen, S. Lindgreen, P. Winter, and M. Zachariasen. Deferred path heuristic for phylogenetic trees revisited. In M.-F. Sagot and K. S. Guimarães, editors, *CompBioNets 2005: Algorithmics and Computational Methods for Biochemical and Evolutionary Networks*, volume 5 of *Texts in Algorithmics*, pages 75–92, King's College London, Strand, London, 2005. College Publications.

[11] N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987.

[12] B. Schwikowski and M. Vingron. Weighted sequence graphs: Boosting iterated dynamic programming using locally suboptimal solutions. *Discrete Applied Mathematics*, 127(1):95–117, 2003. ISSN 0166-218X.

[13] C. Semple and M. Steel. *Phylogenetics*. Oxford University Press, New York, 2003.

[14] J. Stoye, D. Evers, and F. Meyer. Rose: Generating sequence families. *Bioinformatics*, 14(2): 157–163, 1998.

[15] P. E. Sweeney and E. R. Paternoster. Cutting and packing problems: A categorized, application-orientated research bibliography. *Journal of the Operational Research Society*, 43(7):691–706, 1992.

[16] D. M. Warme, P. Winter, and M. Zachariasen. Exact Algorithms for Plane Steiner Tree Problems: A Computational Study. In D.-Z. Du, J. M. Smith, and J. H. Rubinstein, editors, *Advances in Steiner Trees*, pages 81–116. Kluwer Academic Publishers, Boston, 2000.

Published in *Information Processing Letters, 2002*

On the location of Steiner points in uniformly-oriented Steiner trees*

Benny K. Nielsen† Pawel Winter† Martin Zachariasen†

Abstract

We give a fundamental result on the location of Steiner points for Steiner minimum trees in uniform orientation metrics. As a corollary we obtain a linear time algorithm for constructing a Steiner minimum tree for a given full topology when the number of uniform orientations is $\lambda = 3m$, $m \geq 1$.

Keywords: Computational geometry, interconnection networks, Steiner trees

1 Introduction

The *Steiner tree problem* asks for a shortest possible network interconnecting a given set of points. Unlike the minimum spanning tree problem, three or more edges of *Steiner minimum trees (SMTs)* can meet anywhere and not only at the given points. These additional junctions are usually referred to as *Steiner points*.

There are two important variants of the geometric Steiner tree problem in the plane: Euclidean and rectilinear. The Euclidean Steiner tree problem is one of the oldest optimization problems dating back to Fermat. In this variant, distances are measured by the standard L_2-metric. In the rectilinear version, the edges of the SMT are allowed to have horizontal and vertical directions only. This variant has important practical applications in VLSI design. We consider a generalization of the rectilinear Steiner tree problem where the edges are permitted to have any fixed number λ of orientations. More specifically, given λ, the orientations are represented by the lines making $i\omega$ angles with the x-axis, where $\omega = \pi/\lambda$ and $i = 1, 2, ..., \lambda$. Clearly, the rectilinear Steiner tree problem is a special case of the Steiner tree problem with uniform orientations where $\lambda = 2$. In the last few years there has been an increasing interest in the Steiner tree problem with uniform orientations due to potential applications in VLSI design. Some very important structural properties of SMTs with uniform orientations, also referred to as λ-SMTs, were proved by Brazil et al. [1, 2].

Suppose that apart from the coordinates of the points that are to be interconnected, we are also told how to interconnect given points and Steiner points (their locations are unknown). We say that the *topology* $\mathcal{T}$ of the tree is given. More specifically, we consider *full topologies* in which all terminals are leaves. Let $S = \{s_1, \ldots, s_k\}$ denote the set of k Steiner points in $\mathcal{T}$. Note that an assignment of coordinates to the Steiner points in S is a point in $\mathbb{R}^{2k}$. Let

*The research was partially supported by the Danish Natural Science Research Council (contract number 56648).

†Department of Computer Science, University of Copenhagen, DK-2100 Copenhagen Ø, Denmark. E-mail: {benny,pawel,martinz}@diku.dk.

$l_\lambda(S)$ denote the length of the λ-tree when the Steiner points are located in $S \in \mathbb{R}^{2k}$: Our task is now to minimize $l_\lambda(S)$, that is, to construct a λ-SMT having topology $\mathcal{T}$.

We may restrict our attention to full topologies for which every λ-SMT has only positive length edges (i.e., no Steiner points overlap with other Steiner points or with terminals); also, it may be assumed that all Steiner points have degree three [2]. Such topologies will in the following be denoted full *Steiner* topologies.

The construction of λ-SMTs for full Steiner topologies can be performed in linear time for $\lambda = 2, 3$ and ∞, but for other values of λ the complexity is unknown [1, 2]. In this paper we partially answer this open question by showing that the problem can be solved in linear time when $\lambda = 3m$, $m \geq 1$. We prove that the locations of Steiner points for the corresponding Euclidean SMT problem provide possible optimal locations of Steiner points in the λ-tree. Due to the result of Hwang [3], the construction of the Euclidean SMT for a given full Steiner topology — or deciding that it does not exist — can be performed in linear time. Consequently, the construction of a λ-SMT for a given full Steiner topology and $\lambda = 3m$ can also be performed in linear time. Finally, we show that for $\lambda \neq 3m$, the locations of Steiner points in a λ-SMT *never* coincide with the locations of the Euclidean Steiner points.

Preliminary results [5, 6] indicate that our prototype implementation of an exact algorithm for the problem is much slower for $\lambda = 3m$ than for other values of λ. We therefore expect that the result obtained in this paper can be incorporated into new algorithms for the Steiner tree problem with uniform orientations so that a substantial speed-up for $\lambda = 3m$ can be obtained.

2 Basic definitions and results

A λ-edge is a shortest path between two points u and v consisting of at most two line segments that have legal directions. If a λ-edge has only one line segment it is said to be *straight*; if it has two line segments it is said to be *non-straight*. Note that there are two different non-straight λ-edges connecting two points (Figure 1). A Euclidean edge is a line segment between two points, not necessarily in a legal direction.

Consider two λ-edges uv and uw meeting at a node u. If both λ-edges are non-straight they can make three different angles at u, depending on which lines segments form the λ-edges. If $\phi_\lambda^{\min}$ and $\phi_\lambda^{\max}$ denote the minimum and maximum angle at which λ-edges uv and uw can meet at u, the angle ϕ between the Euclidean edges uv and uw is clearly $\phi_\lambda^{\min} \leq \phi \leq \phi_\lambda^{\max}$ (Figure 1).

The main results in this paper will be obtained from the following known result on λ-SMT angles:

Theorem 4. *[2] Let $\lambda = 3m + i$, where $i \in \{0, 1, 2\}$. Let $k\omega$ be the angle between two neighbouring λ-edges meeting at a* Steiner point *in a λ-SMT (k is a positive integer). Then,*

$$\begin{array}{ll} k \in \{2m-1, 2m, 2m+1\} & \text{if } i = 0, \\ k \in \{2m, 2m+1\} & \text{if } i = 1, \\ k \in \{2m+1, 2m+2\} & \text{if } i = 2. \end{array}$$

Informally, the minimum angle is the largest multiple of ω that is strictly less than $2\pi/3$; similarly, the maximum angle is the smallest multiple of ω that is strictly larger than $2\pi/3$. For neighbouring λ-edges meeting at *terminals* the lower bound still holds.

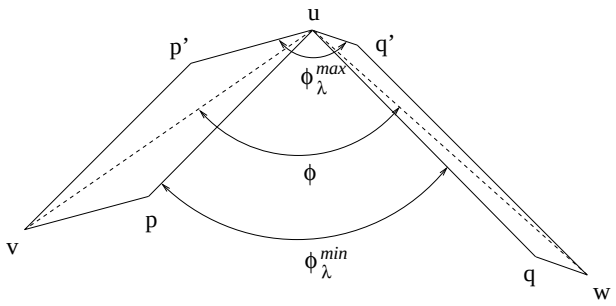


Figure 1: Two λ-edges uv and uw incident to the same node u. The corresponding Euclidean edges are drawn as dashed line segments.

Lemma 4. *[2] Let u be a node in a λ-SMT ($\lambda = 3m$) having two incident λ-edges making an angle of $(2m-1)\omega$ with each other. Consider any of the two rays from u that make angles $(m-1)\omega$ and $m\omega$ with the edges incident at u. Then we may insert a Steiner point s on the ray, sufficiently close*[1] *to u,* without *changing the length of the λ-SMT (Figure 2a).*

This lemma implies that a Steiner point having two incident λ-edges making an angle of $(2m-1)\omega$ with each other can be *moved* — without changing the length of the tree — along a direction that approximately is a bisector of this (minimum) angle. For λ-edges meeting at a terminal, this means that a Steiner point can be *created* along the same direction. Brazil et al. [2] used this so-called variational argument to prove that for $\lambda = 3m$ there exists a λ-SMT in which all edges meeting at a node make an angle of at least $2\pi/3$ with each other (as in the Euclidean Steiner tree problem).

Consider two λ-edges of a λ-SMT making an angle $(2m-1)\omega$. The *λ-bisector wedge* is defined as the area given by the two rays making $(m-1)\omega$ and $m\omega$ with the two edges; the bounding rays are assumed to belong to the λ-bisector wedge (Figure 2a). We can now prove the following straightforward generalization of Lemma 4.

Lemma 5. *Let u be a node in a λ-SMT ($\lambda = 3m$) having two incident edges making an angle of $(2m-1)\omega$ with each other. Then a Steiner point s may be inserted at any point in the λ-bisector wedge, sufficiently close to u,* without *changing the length of the λ-SMT.*

Proof. A Steiner point s in the λ-bisector wedge can be reached by performing two Steiner point insertions along the rays defining the wedge (Figure 2b). First the Steiner point s' is inserted. The incident λ-edges can always be made to make an angle of $(2m-1)\omega$ with each other — in fact to be parallel with the original edges incident at u. Then Steiner point s is inserted along the second ray. □

3 Steiner points for $\lambda = 3m$

Let $\mathcal{T}$ be a full Steiner topology with k Steiner points. Let $T_\lambda(S^*)$ be the λ-tree for topology $\mathcal{T}$ when the Steiner points are located in $S^* \in \mathbb{R}^{2k}$. Assume that $T_\lambda(S^*)$ is a λ-SMT for topology $\mathcal{T}$; as noted in Section 1, no Steiner point in S^* overlaps with a terminal or another

[1] The distance from u to s is always positive, but it depends on the length of the two incident line segments (up and uq in Figure 2a).

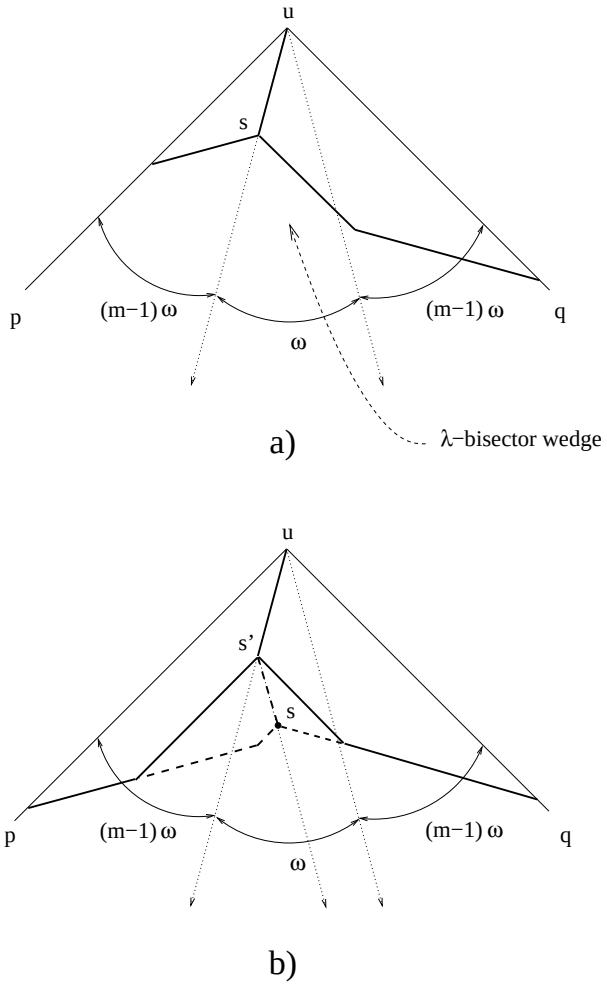


Figure 2: Inserting a Steiner point. (a) Ray insertion: Steiner point s is inserted on a ray making angles $(m-1)\omega$ and $m\omega$ with the edges incident at u. (b) Wedge insertion: Steiner point s is inserted at a point in the λ-bisector wedge by two ray insertions. Note how one of the edges incident at s' is flipped in order to obtain a $(2m-1)\omega$ angle so that the second ray insertion can be performed.

Steiner point. Now, let $T(S^*)$ be the Euclidean tree having the same topology and the same location for all terminals and Steiner points as $T_\lambda(S^*)$, the so-called "underlying" Euclidean tree; all λ-edges have been straightened out in this Euclidean tree.

Lemma 6. *The Steiner points for $T_\lambda(S^*)$ ($\lambda = 3m$) can be located such that $T(S^*)$, the underlying Euclidean tree, is a Euclidean SMT for the same topology.*

Proof. Select the λ-SMT $T_\lambda(S^*)$ such that the underlying Euclidean tree $T(S^*)$ has the smallest possible length. First we prove that such a set of Steiner points S^* actually exists.

Let $l_\lambda(S)$ denote the length of the λ-tree when the Steiner points are located in S; similarly, let $l(S)$ denote the length of the Euclidean tree for the same set of Steiner points. Let $L = \min_{S \in \mathbb{R}^{2k}} l_\lambda(S)$ denote the length of a λ-SMT. Consider the set $\mathcal{C} = l_\lambda^{-1}(\{L\}) \subset \mathbb{R}^{2k}$, that is, the set of optimal Steiner point configurations. Since $\{L\}$ is a closed set and $l_\lambda(S)$ is a continuous function, the set $\mathcal{C}$ is closed, too. Furthermore, the set $\mathcal{C}$ is clearly bounded. Thus the continuous function $l(S)$ attains a minimum $S^* \in \mathcal{C}$.

Assume that $T(S^*)$ is *not* a Euclidean SMT. Then there exists a Steiner point u such that at least one pair of the incident (Euclidean) edges makes angle $\phi < 2\pi/3$ with each other. It is well-known [4] that the Euclidean tree can be shortened by inserting a Steiner point s along the bisector of this angle (Figure 3a).

Since $(2m-1)\omega \leq \phi < 2\pi/3 = 2m\omega$, the incident λ-edges can be assumed to make $(2m-1)\omega$ with each other. We will now show that the (Euclidean) Steiner point s is in fact in the λ-bisector wedge.

Let α and β be the residual angles as shown in Figure 3b, such that $\phi = (2m-1)\omega + \alpha + \beta$, where $0 \leq \alpha < \omega$, $0 \leq \beta < \omega$. The bisector angle is $\phi/2 = (m-1/2)\omega + \alpha/2 + \beta/2$. We have $m\omega + \alpha > m\omega - \omega/2 + \beta/2 + \alpha \geq \phi/2$ and $m\omega + \beta > m\omega - \omega/2 + \alpha/2 + \beta \geq \phi/2$. Thus s is in the λ-bisector wedge. Lemma 5 shows that inserting s into the λ-tree will not change its length (assuming that s is sufficiently close to u). We have constructed a new λ-SMT for which the underlying Euclidean tree has strictly smaller length, a contradiction to the minimality of $T(S^*)$ over the set $\mathcal{C}$. □

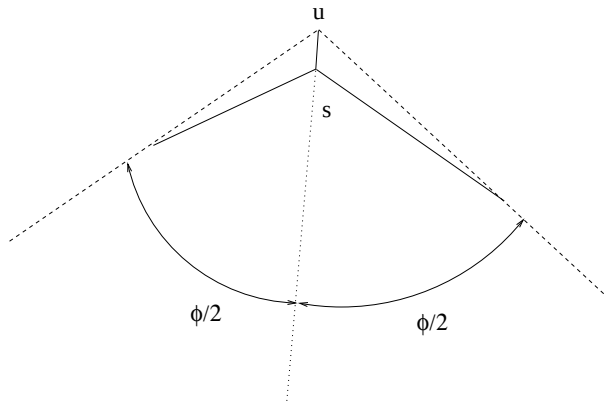
It follows that the Steiner points in a λ-SMT having a given full Steiner topology can be assumed to be identical to the Euclidean Steiner points for the same topology. Therefore, we may use Hwang's linear time algorithm [3] to compute the Steiner points; the terminals and Steiner points are then connected using λ-edges.

Theorem 5. *A λ-SMT for a full Steiner topology can be constructed in linear time for $\lambda = 3m$.*

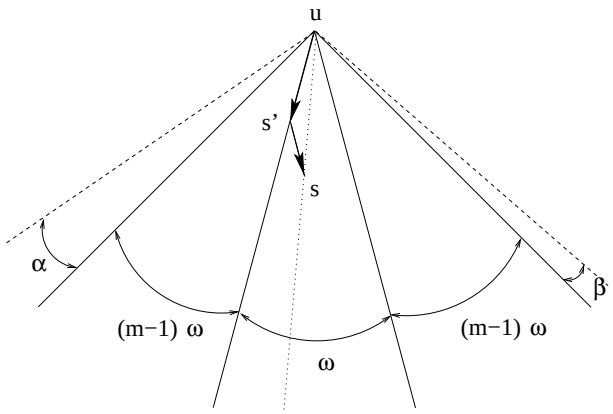
4 Conclusions

In Theorem 5 we gave a partial answer to the open question stated in [1, 2]. For $\lambda \neq 3m$ the complexity of the problem remains unknown, and it should be noted that the Euclidean Steiner points are *never* optimal for a λ-SMT when $\lambda \neq 3m$:

Lemma 7. *Let s be a Steiner point in a λ-SMT ($\lambda \neq 3m$). Then s* cannot *be identical to the Euclidean Steiner point for its neighbours.*



a)



b)

Figure 3: Inserting a Euclidean Steiner point. (a) Steiner point s can be inserted in order to shorten the Euclidean tree when $\phi < 2\pi/3$. (b) Steiner point u can be moved to point s without changing the length of the λ-tree. (The Euclidean tree is drawn with dashed line segments.)

Proof. Assume that the Steiner point s in the λ-SMT is identical to the Euclidean Steiner point, such that the Euclidean edges make angle $2\pi/3$ with each other. There are at least two non-straight incident λ-edges in the λ-tree since otherwise there would be an angle of $2\pi/3$ between the two straight λ-edges and this is not possible when $\lambda \neq 3m$. However, the two non-straight λ-edges can meet in three different angles, and by Theorem 4 this is a contradiction to the optimality of the λ-SMT. □

Acknowledgments

The authors would like to thank one referee for many helpful comments and suggestions that improved this paper significantly.

References

[1] M. Brazil. Steiner minimum trees in uniform orientation metrics. In D.-Z. Du and X. Cheng, editor, *Steiner trees in industries*, pages 1–27. Kluwer Academic Publishers, 2001.

[2] M. Brazil, D. A. Thomas, and J. F. Weng. Minimum networks in uniform orientation metrics. *SIAM Journal on Computing*, 30:1579–1593, 2000.

[3] F. K. Hwang. A linear time algorithm for full Steiner trees. *Operations Research Letters*, 4(5): 235–237, 1986.

[4] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner tree problem*. Annals of Discrete Mathematics 53. Elsevier Science Publishers, Netherlands, 1992.

[5] B. K. Nielsen, P. Winter, and M. Zachariasen. An exact algorithm for the uniformly-oriented Steiner tree problem. In preparation, 2001.

[6] D. M. Warme, P. Winter, and M. Zachariasen. GeoSteiner 3.1. Department of Computer Science, University of Copenhagen (DIKU), http://www.diku.dk/geosteiner/, 2001.

Published in *Computational Geometry: Theory and Applications, 2004*

Rotationally optimal spanning and Steiner trees in uniform orientation metrics*

Marcus Brazil† Benny K. Nielsen‡ Pawel Winter‡ Martin Zachariasen‡

Abstract

We consider the problem of finding a minimum spanning and Steiner tree for a set of n points in the plane where the orientations of edge segments are restricted to λ uniformly distributed orientations, $\lambda = 2, 3, 4, \ldots$, and where the coordinate system can be rotated around the origin by an arbitrary angle. The most important cases with applications in VLSI design arise when $\lambda = 2$ or $\lambda = 4$. In the former, so-called rectilinear case, the edge segments have to be parallel to one of the coordinate axes, and in the latter, so-called octilinear case, the edge segments have to be parallel to one of the coordinate axes *or* to one of the lines making 45° with the coordinate axes (so-called diagonals). As the coordinate system is rotated — while the points remain stationary — the length and indeed the topology of the minimum spanning or Steiner tree changes. We suggest a straightforward polynomial-time algorithm to solve the rotational minimum spanning tree problem. We also give a simple algorithm to solve the rectilinear Steiner tree problem in the rotational setting, and a finite time algorithm for the general Steiner tree problem with λ uniform orientations. Finally, we provide some computational results indicating the average savings for different values of n and λ both for spanning and Steiner trees.

Keywords: Steiner trees in uniform orientation metrics, rotational problems, VLSI design

1 Introduction

Suppose that we are given a set P of n points in the plane. We are interested in finding a minimum spanning tree (MST) or a Steiner minimum tree (SMT) for P under the assumption that the edge segments are permitted to have a limited number of orientations. In fact, we will assume that these orientations are evenly spaced. More specifically, let λ be a non-negative integer, $\lambda \geq 2$. Let $\omega = \pi/\lambda$. Legal orientations are then defined by the straight lines passing through the origin and making angles $i\omega$, $i = 0, 1, ..., \lambda - 1$ with the x-axis. Finding an MST in this so-called λ-geometry is not a challenging problem since it can be defined as a minimum spanning tree problem in a complete graph with appropriate edge lengths. On the other hand, computing an SMT in this setting is known to be an $\mathcal{NP}$-hard problem [1].

Suppose that we are permitted to rotate the coordinate system, that is, to rotate all legal orientations simultaneously. Rotation by ω (or a multiple of ω) will have no impact on the

*Partially supported by grants from the Australia Research Council and from the Danish Natural Science Research Council (contract number 56648).

†ARC Special Research Centre for Ultra-Broadband Information Networks, Department of Electrical and Electronic Engineering, The University of Melbourne, Victoria 3010, Australia. E-mail: brazil@unimelb.edu.au.

‡Department of Computer Science, University of Copenhagen, DK-2100 Copenhagen Ø, Denmark. E-mail: {benny, pawel, martinz}@diku.dk.

lengths of the edges. But for any angle $\alpha \in [0, \omega[$, the edge lengths will change. What is the value of α minimizing the length of an MST or SMT for P? Once α is fixed, finding an MST is straightforward, while determining similar SMTs clearly remains $\mathcal{NP}$-hard.

Our interest in rotated MSTs and SMTs with bounded orientations was motivated by recent developments in VLSI technology. Chips with $\lambda = 4$ have already been produced [15] while there are several VLSI-papers addressing the design of chips for $\lambda = 3$ [3, 4]. Furthermore, additional length savings seem to be available when the coordinate system can be rotated. This is in particular useful for small values of λ and for nets with limited number of terminals.

In this paper we give a straightforward polynomial-time algorithm for the rotational MST problem. Also, we give a finite time algorithm for the rotational SMT problem. In particular, for the most important rectilinear case ($\lambda = 2$), we show that no more than $O(n^2)$ rotation angles need to be considered in order to compute a rotationally optimal rectilinear SMT.

Our results on the structure of rotationally optimal MSTs and SMTs appear to give insight into the problem of determining the Steiner ratio in λ-geometry for all values of λ (consult the book by Cieslik [5] for a comprehensive survey).

The paper is organized as follows. As a warm up, in Section 2 we determine how the length of a single edge changes under rotation. In Section 3 we present our polynomial-time algorithm for the MST problem and in Section 4 a finite time algorithm for the SMT problem is given. Computational results for both randomly generated instances and instances from VLSI design are presented in Section 5. Finally, in Section 6 we give some concluding remarks and discuss related and open problems.

2 Length of a Segment

Consider a given λ and rotation angle $\alpha \in [0, \omega[$. In this section we examine the situation where there are only two points $u = (u_x, u_y)$ and $v = (v_x, v_y)$. We let $|uv|_\alpha$ denote the length of the shortest connection between u and v with the restriction that the straight line segments of the connection only use one of the λ legal orientations. We may restrict our attention to the cases where this connection consists of either one or two line segments [13]. In the former case we say that the edge between u and v is *straight* (and $|uv|_\alpha$ is equal to the Euclidean distance $\|uv\|$ between u and v); otherwise the edge is *bent* and the two line segments form the *half-edges* of the edge between u and v. The two half-edges meet at the *corner point* of the bent edge, and the smaller meeting angle is $\pi - \omega$ [13].

The shortest MST or SMT for u and v is obtained when the coordinate system is rotated so that the orientation of the line segment uv overlaps with one of the legal orientations, i.e., the edge between u and v is straight. So our problem is trivial. However, it is still interesting to determine how the length of the edge uv changes as the coordinate system is rotated.

Assume that u and v are on the horizontal x-axis and $u_x < v_x$. When the coordinate system is rotated by increasing angle α, $0 \leq \alpha \leq \omega$, the length of uv increases until $\alpha = \omega/2$, and then decreases until $\alpha = \omega$ (Figure 1). More specifically,

$$|uv|_\alpha = \|uv\| \frac{\sin\alpha + \sin(\omega - \alpha)}{\sin(\omega)}$$

In particular, if $\lambda = 2$ and $\omega = \pi/2$, then

$$|uv|_\alpha = \|uv\|(\sin\alpha + \cos\alpha)$$

It is obvious that the function $f_{uv}(\alpha) = |uv|_\alpha$ is periodically strictly concave (for $\alpha \in [0, 2\pi[$ and with period $\omega = \pi/\lambda$). Furthermore, the minimum is attained when the orientation of the line segment uv overlaps with one of the legal orientations.

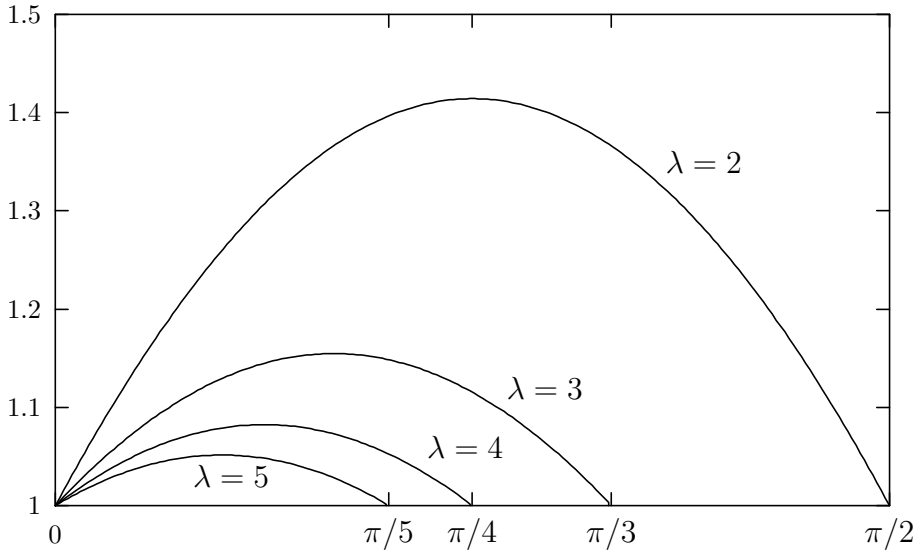


Figure 1: $f_{uv}(\alpha) = |uv|_\alpha$ for $\|uv\| = 1$ and $\alpha \in [0, \frac{\pi}{\lambda}]$, $\lambda = 2, 3, 4, 5$.

3 Minimum Spanning Tree

Consider a collection S of m edges $u_1v_1, \ldots, u_mv_m$. Their total length as a function of the rotation angle α is $|S|_\alpha = \sum_{i=1}^{m} |u_iv_i|_\alpha$.

Lemma 3.1. *The total length $|S|_\alpha$ of the edges in S is minimized when one of the edges u_iv_i is* straight*, i.e., when the orientation of the line segment from u_i to v_i overlaps with one of the legal orientations.*

Proof. Since $|S|_\alpha$ is a sum of piecewise strictly concave functions, $|S|_\alpha$ is also piecewise strictly concave. Therefore, the minimum is attained at one of the non-concave points (or break-points), that is, when one of the edges is straight. □

Consider a set P of n points in the plane. Assume that the coordinate system has been rotated in such a way that an MST T for P is shortest possible.

Theorem 3.2. *One of the edges in a rotationally optimal MST T overlaps with one of the legal orientations.*

Proof. Follows immediately from Lemma 3.1. □

The algorithm to determine T is therefore straightforward. For each pair of points u and v of P, consider the edge uv. Rotate the coordinate system so that the orientation of one of the legal orientations overlaps with the line segment from u to v. Compute an MST for P and store it provided that it is shorter than any MST found so far. Figure 2 shows how

the lengths of MSTs for a set of 10 points and $\lambda = 4$ changes when the coordinate system is rotated. The histogram was generated by computing 1000 MSTs for values of α evenly distributed in the interval $[0, \frac{\pi}{4}]$. The vertical dashed lines indicate a subset (see Section 5) of the MSTs computed by the algorithm, corresponding to those for which the MST length might not be a strictly concave function of the rotation angle. The optimum is on the x-axis.

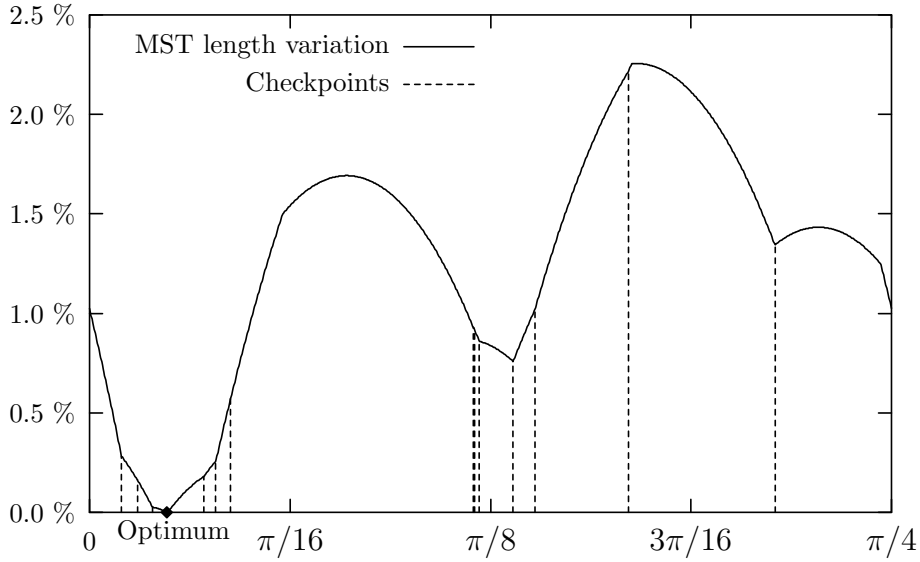


Figure 2: MST length variation as a function of rotation angle.

For a fixed rotation of the coordinate system, an MST can be computed in $O(n \log n)$ time [13]. Since it is only necessary to consider $O(n^2)$ different rotation angles, the total running time for computing a rotationally optimal MST is $O(n^3 \log n)$. In Section 5 we will address the issue of making this algorithm much more efficient in practice.

4 Steiner Minimum Tree

Consider the problem of interconnecting the set P of points by a tree of minimum total length, but *allowing* additional junctions, so-called Steiner points. When a set of λ uniformly spaced legal orientations is given, this is denoted the Steiner tree problem in uniform orientation metrics [1, 10].

A Steiner minimum tree (SMT) is a union of full Steiner trees (FSTs) in which all terminals are leaves (and all interior nodes are Steiner points). In λ-geometry FSTs are denoted λ-FSTs. Our aim in this section is to prove that for a rotationally optimal SMT there exists a λ-FST with no bent edges. We first prove this for the case where λ is a multiple of 3. The proof relies on an elegant theorem showing the relationship between the lengths of such λ-FSTs and equivalent FSTs in Euclidean geometry. This theorem is likely to be of independent interest.

We then prove the result for general λ. Note that a separate proof for the rectilinear case ($\lambda = 2$) appears in [9].

4.1 The case where λ is a multiple of 3

A Simpson line for a *Euclidean* FST F is a line segment extending one of the edges of F such that the Euclidean distance between the endpoints equals $\|F\|$, the total (Euclidean) length of edges in F. It is well known that a Simpson line for F can be constructed in linear time [6, 7]. For λ being a multiple of 3 it turns out that the λ-geometry distance between the endpoints of the Simpson line also equals the length of the corresponding λ-FST:

Theorem 4.1. *Suppose that λ is a multiple of 3. Let F be a Euclidean FST with terminal set N and topology $\mathcal{T}$, and let F_λ be a λ-FST, also with terminal set N and topology $\mathcal{T}$. Let s_1s_2 be a Simpson line of F of length $\|s_1s_2\| = \|F\|$, and let $|s_1s_2|$ be the distance in λ-geometry between s_1 and s_2, the endpoints of the Simpson line. Then the length of F_λ is $|s_1s_2|$.*

Proof. The result is clearly true if $|N| = 2$; in this case the endpoints of the Simpson line coincide with the two terminals in N. So we may assume that $|N| \geq 3$ and that $\mathcal{T}$ contains at least one Steiner point. By Nielsen et al. [8]$^{\mathrm{E}}$, we can choose F_λ so that the Steiner points of F_λ exactly coincide with the Steiner points of F (Figure 3).

Consider an edge uv of F and the corresponding λ-edge uv_λ in F_λ. Let $\theta_1, \theta_2 \in [0, \omega[$ be the smaller angles that the two half-edges of uv_λ make with the straight edge between u and v. Since λ is a multiple of 3 and the edges of the Euclidean FST F use exactly 3 equally spaced orientations (separated by angles which are multiples of $\pi/3$), the angles θ_1 and θ_2 will be the same for all edges in F_λ. Furthermore, if an edge uv is rotated through any multiple of $\pi/3$, the rotated half-edges still overlap with legal orientations.

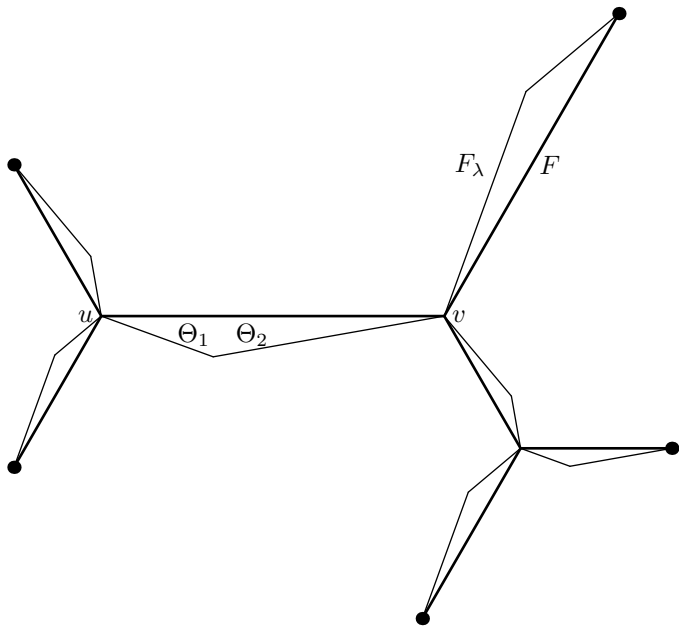


Figure 3: Illustration of Theorem 4.1. Straight edges between terminals and/or Steiner points belong to F while half-edges belong to F_λ.

Consider a transformation of the edges of F, composed of rotations and translations, such that they are placed end-to-end along the Simpson line. Since $\|F\| = \|s_1s_2\|$, the edges

will exactly cover the Simpson line. Now, consider the corresponding transformation of λ-edges. In view of the above, all transformed edges will use the same two (neighbouring) legal orientations. Therefore, their total length will be exactly $|s_1 s_2|$, the λ-geometry distance between the endpoints of the Simpson line. □

Theorem 4.2. *Suppose λ is a multiple of 3. A rotationally optimal SMT in λ-geometry is a union of λ-FSTs, at least one of which contains no bent edges.*

Proof. Replace each λ-FST in the SMT by a geodesic in λ-geometry between the endpoints of a Simpson line for the corresponding Euclidean FST. (Note that this replaces each λ-FST by a simple λ-edge between two fixed points.) By Theorem 4.1, for any fixed rotation angle, the sum of the lengths of these geodesics in λ-geometry equals the length of the SMT. Hence, we can apply exactly the same argument as in the MST case. □

4.2 The case for general λ

In this subsection we show that Theorem 4.2 holds for arbitrary λ. First, assume that the coordinate system is rotated by a fixed angle $\alpha \in [0, \omega[$. In this case the problem to be solved is the Steiner tree problem in uniform orientation metrics [1, 10]. Brazil et al. [1] proved that for any given instance of this problem, there exists an SMT for which every λ-FST has *at most* one bent edge.

Now let us assume that the rotational λ-geometry Steiner tree problem has an optimal solution for a given rotation angle $\alpha^* \in [0, \omega[$. We wish to show that at least one of the λ-FSTs in the SMT contains no bent edge. First we note that if an SMT contains a Steiner point s with degree more than 3, then the λ-FST for which s is an internal node contains *no* bent edges [1]. Thus we may restrict our attention to SMTs for which all Steiner points have degree 3. (For a more systematic coverage of degrees of Steiner points in λ-geometries and in normed planes in general, the reader is referred to [11].)

Let F be a λ-FST with a single bent edge. Consider a rotation of the coordinate system through an angle α. We define F_α to be a λ-FST under this rotated coordinate system on the same terminals as F such that 1) the topology of F_α is the same as the topology of F, and 2) the angles between edges at every Steiner point of F_α are the same as the corresponding angles of F. It is clear that F_α always exists for a sufficiently small angle α (i.e., until the length of some edge or half edge goes to 0). In fact, below we give a construction that proves the existence of F_α. Let $|uv|_\alpha$ be the length of some straight edge or a half-edge uv in F_α as a function of α (for $\alpha \in I_\epsilon =]\alpha^* - \epsilon, \alpha^* + \epsilon[$ where $\epsilon > 0$ is sufficiently small). We will show that $|uv|_\alpha$ is a strictly concave function of α in a sufficiently small neighbourhood I_ϵ of α^*. But first we need the following result.

Lemma 4.3. *Let C and D be two distinct circles intersecting in a point q. Let Φ_α, $\alpha \in I_\epsilon$, be a ray from q with polar angle α intersecting C in s and intersecting D in v, $s \neq v$. Then $|sv|_\alpha$ is a strictly concave function for $\alpha \in I_\epsilon$.*

Proof. By rescaling, translating and rotating, we can assume, without loss of generality, that D is a unit circle, centered at $(1, 0)$, such that the point q is at the origin (Figure 4). In terms of polar coordinates, the equation of D is $r = 2\cos\alpha$. If the centre of C in polar coordinates is (R, ϕ) then C has the polar equation $r = 2R\cos(\alpha - \phi)$, since q also lies on C. Hence

$|sv|_\alpha = 2\cos\alpha - 2R\cos(\alpha - \phi)$, and

$$\frac{d^2|sv|}{d\alpha^2} = -2\cos\alpha + 2R\cos(\alpha - \phi) = -|sv|_\alpha$$

which is strictly negative whenever sv exists. It immediately follows that $|sv|_\alpha$ is strictly concave over a sufficiently small interval I_ϵ. Note that the lemma holds when C degenerates to a point. □

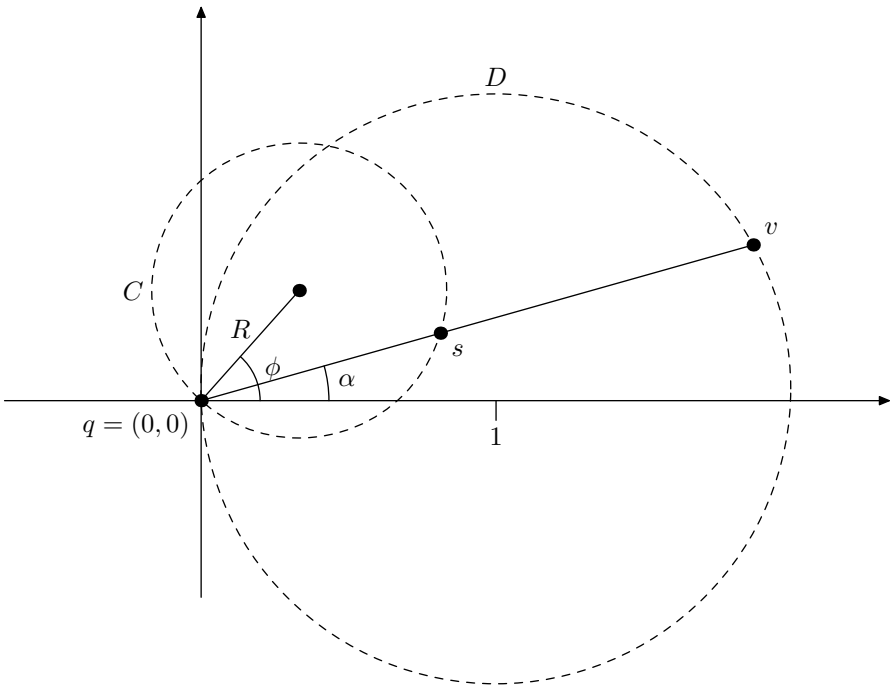


Figure 4: Computing the length of edge uv using polar coordinates.

The following lemma makes use of Thales theorem, from elementary geometry, which states that for any circle the angle subtended by an arc at the circumference of the circle is half the angle at the centre of the circle (and hence is fixed for a given arc).

Lemma 4.4. *Let F be a λ-FST with a single bent edge. As the coordinate system is continuously rotated by $\alpha \in I_\epsilon$, the length of each straight edge and half-edge of F defines a strictly concave function of α.*

Proof. We will show that, for every straight edge or half-edge sv of F, there exist circles C and D such that the conditions of Lemma 4.3 are satisfied. Furthermore, we will show that if sv is incident with a terminal, then we have the degenerate case, where one of the circles is a single point. This clearly suffices to prove the lemma.

The proof is by induction on the number of Steiner points in F. For the base case, if F has no Steiner points, it consists of a bent edge connecting two terminals t_1 and t_2. As the coordinate system is rotated by $\alpha \in I_\epsilon$, terminals t_1 and t_2 remain in their positions while half edges change directions (following legal orientations). But the angle at which the half edges

meet at the corner point v remains the same. By Thales theorem, v moves along the circle through t_1 and t_2 (and v). Hence, for each of the half edges the conditions of Lemma 4.3 are satisfied (where C is a single point). This is equivalent to the discussion in Section 2.

Suppose that F contains at least one Steiner point. Then clearly, F has a Steiner point s adjacent to two terminals t_1 and t_2 such that t_1s and t_2s are straight edges. Let C be the circle through t_1, s, and t_2. Let sv denote the third straight edge or half edge in F incident with s (i.e., v is either a Steiner point or corner point). Let e_1 be the point on C intersected by the line extending the line segment sv (Figure 5). As the coordinate system is continuously rotated by $\alpha \in I_\epsilon$, t_1 and t_2 remain in their positions. The edges t_1s and t_2s change directions (following legal orientations) but they meet at (moving) s at the same angle. Hence, by Thales theorem, s moves continuously on C, and t_1s and t_2s again satisfy the conditions of Lemma 4.3 (where the other circle has degenerated to a point). The location of v also changes during the rotation. However, the angles $\angle t_1sv$ and $\angle vst_2$ remain the same. As a consequence, $\angle e_1st_1$ and $\angle t_2se_1$ remain the same. But this implies that e_1 remains fixed as the coordinate system rotates within I_ϵ.

Consider now an FST F' obtained from F by deleting t_1, t_2 and s (together with their incident edges) and by adding e_1 as a terminal adjacent to v. By the inductive hypothesis, in F', v moves on a circle passing through e_1, say D, as the coordinate system is rotated by $\alpha \in I_\epsilon$ (Figure 5). Hence, in F sv satisfies the conditions of Lemma 4.3. Furthermore, apart from the edge e_1v, all other edges of F' are the same as in F. As a consequence, the remaining straight edges and half edges of F all satisfy the conditions of Lemma 4.3 by the inductive hypothesis. □

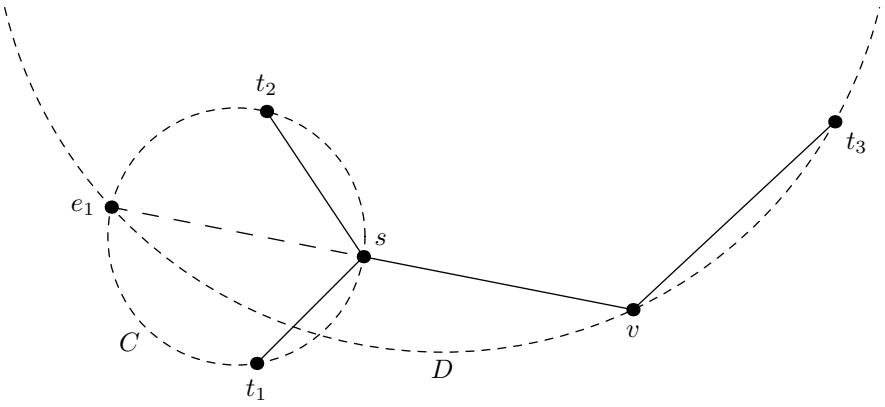


Figure 5: Replacing a pair of terminals with a pseudo-terminal.

Theorem 4.5. *A rotationally optimal SMT in λ-geometry is a union of λ-FSTs, at least one of which contains no bent edges.*

Proof. Assume that every λ-FST of an SMT contains a bent edge. Lemma 4.4 shows that the length of every edge in the SMT is strictly concave under sufficiently small rotations. Hence, the same holds for the sum of all edge lengths. Therefore, the SMT can in fact be shortened, which is a contradiction. □

A consequence of Theorem 4.5 is that there exists a *finite time algorithm* for the rotational SMT problem in λ-geometry. For every subset $N \subseteq P$ of terminals, every topology $\mathcal{T}$ on N, and every legal angle distribution around Steiner points in $\mathcal{T}$, a λ-FST without bent edges is constructed (if it exists). For a given terminal set N, topology $\mathcal{T}$ and angle distribution, a bottom-up construction similar to the one given in the proof of Lemma 4.4 can be used. Whenever such a tree exists, the orientations used by the edges in the tree give a feasible rotation angle α for which the SMT problem in uniform orientation problems is solved in finite time [1].

For general λ a super-exponential number of rotation angles must be tried. However, for $\lambda = 2$ only $O(n^2)$ rotation angles need to be considered — namely those given by the lines through each pair of terminals. This follows from the fact that for $\lambda = 2$ an SMT can be assumed to consist of Hwang-topology FSTs. When no bent edge is present in such an FST, it will contain a straight line segment between a pair of terminals [9].

It is possible to give a stronger version of Theorem 4.5 using some powerful characterizations of λ-FSTs from [2].

Theorem 4.6. *A rotationally optimal SMT in λ-geometry is a union of λ-FSTs, at least one of which uses at most 3 edge orientations.*

Proof. First we may assume by Theorem 4.5 that a rotationally optimal SMT has some λ-FST with no bent edges. Assume that all FSTs with no bent edges use 4 or more edge orientations. Let F be one of these λ-FSTs. Since the edges of F use 4 or more orientations, there exists a length-preserving perturbation of the Steiner points (a so-called zero-shift) in F such that exactly *two* bent edges are created [2].

Consider the connection between the two bent edges in F. Let h be one of the half-edges that is closest to the other bent edge. Pick an interior point on h and fix it (i.e., make it into a pseudo-terminal). Now we have divided F into two (pseudo) FSTs, each with one bent edge. From Lemma 4.4 it follows that the length of each (pseudo) FST is strictly concave under sufficiently small rotations. As a consequence, the length of every λ-FST in the optimal SMT (with as well as without bent edges) is strictly concave under sufficiently small rotations. But this means that the sum of their lengths is also strictly concave under sufficiently small rotations. Therefore the SMT can be shortened, which is a contradiction. □

It should be noted that another consequence of the results in [2] is that there in fact must be a λ-FST that uses 3 edge orientations that have a certain well-defined distribution. For λ being a multiple of 3 this means that there must exist a λ-FST in a rotationally optimal SMT that uses 3 edge orientations separated by an angle of $\pi/3$; such a λ-FST will be identical to the *Euclidean* FST spanning the same set of terminals and having the same topology.

5 Computational Results

In this section we give some computational results indicating the effect of allowing rotations of the coordinate system when computing MSTs and rectilinear SMTs. We used two sets of problem instances: VLSI design instances and randomly generated instances (uniformly distributed in a square). The VLSI instances were made available by courtesy of IBM and Research Institute for Discrete Mathematics, University of Bonn. In this study we have focused on one particular chip from 1996.

5.1 Minimum Spanning Tree

Generally most of the edges for a given point set will not be part of a MST for any rotation angle. It is a waste of time to consider the rotation angle given by such pairs of points (or edges). Fortunately, most of these edges can be pruned by using so-called bottleneck Steiner distances.

Consider the complete graph K_n where the vertices represent the points in the plane. Define the weight of the edge between vertices u and v to be the maximum distance between u and v in λ-geometry under any rotation angle. Compute bottleneck (Steiner) distances in K_n. It is defined as the minimum over the largest-weight edges in all paths between u and v in K_n, and can be computed in time $O(n^2)$ for all pairs of points; see [7, 14] for details. Let B_{uv} denote the bottleneck (Steiner) distance between u and v. Whenever $\|uv\| > B_{uv}$, then the edge uv cannot be in any minimum spanning tree generated during the rotation of the coordinate system. Figure 6 indicates how many edges survive the pruning for $n = 50$ and $\lambda = 2, 3, 4, 5$.

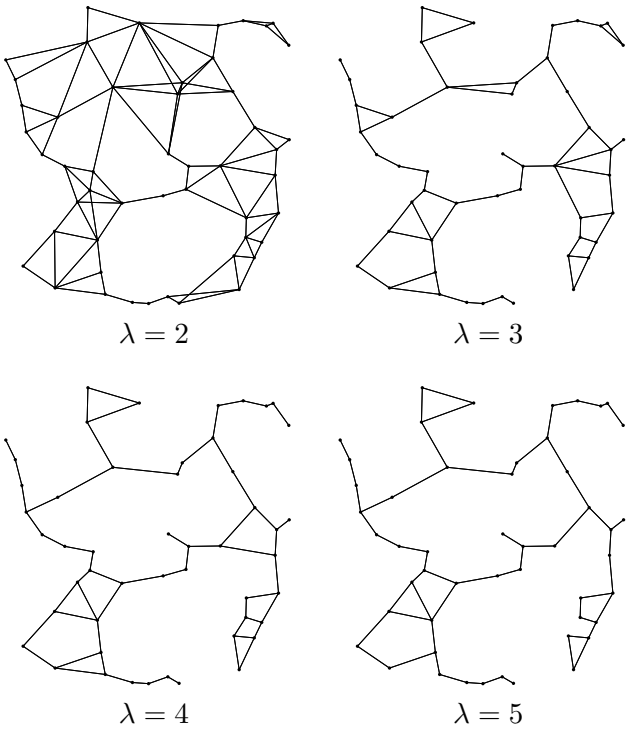


Figure 6: Edges surviving pruning of K_{50} for $\lambda = 2, 3, 4, 5$.

This pruning turns out to be extremely efficient in general. Table 1 shows the number of edges that survive for different values of λ and n. Each entry is an average over 100 instances.

The extraordinary efficiency of pruning makes it possible to solve the problem of finding the best rotated MST much faster (especially for higher values of λ). It will on average require $O(n^2 \log n)$ time. However, it should be noted that it is possible to construct problem instances of any size where the number of edges after pruning is $\Omega(n^2)$. Consider for example a rectangle where sides have lengths a and b, $a << b$ (a much smaller than b). Place $n/2$

λ	n=3	n=4	n=5	n=10	n=20	n=50	n=100
2	2.76	4.86	7.38	20.12	44.77	122.09	253.84
3	2.34	3.84	5.50	12.62	26.77	71.43	146.81
4	2.22	3.44	4.77	10.77	23.05	60.51	123.05
5	2.13	3.34	4.54	10.17	21.28	56.11	113.59
6	2.07	3.27	4.38	9.83	20.54	53.89	108.57
7	2.06	3.21	4.28	9.54	20.05	52.50	105.88
8	2.03	3.12	4.18	9.36	19.76	51.74	104.49
9	2.02	3.09	4.14	9.32	19.64	51.09	103.29
10	2.02	3.07	4.10	9.29	19.56	50.69	102.49
16	2.01	3.03	4.03	9.08	19.17	49.73	100.33
32	2.00	3.00	4.00	9.01	19.03	49.14	99.41

Table 1: Number of surviving edges after pruning for randomly generated instances (averages over 100 instances).

terminals on one of the shorter sides such that two consecutive terminals are $2a/(n-1)$ apart. Place the remaining $n/2$ terminals in the same manner on the other shorter side. It is obvious that for every of $n^2/4$ pairs of terminals, one terminal from each side, the pruning test described in this section will not apply.

Table 2 shows the MST improvement for various values of n and λ where each entry is an average over 100 random instances. The table contains two measures of improvement. The first one is calculated as $1 - \frac{|T_{\min}|}{|T_{\max}|}$ in percent. $T_{\min}$ is the shortest MST while $T_{\max}$ is the longest MST taken over 600 uniformly distributed values of the rotation angle α in the interval $[0, \omega[$ where $\omega = \lambda/\pi$. In the second measure of improvement the value $T_{\max}$ has been replaced by the length of the MST without rotating the points ($\alpha = 0$); a natural alternative.

5.2 Rectilinear Steiner Tree

We used GeoSteiner [12] to compute RSMTs for each of the $O(n^2)$ orientations given by the pairs of terminals (where n is the number of terminals). The RSMT improvement for nets of given size obtained by rotating the coordinate system can be seen in Table 3. The values are percentages which express the improvement compared to not rotating at all. Results for both random and VLSI instances are given. For small problem instances the improvements are highest for randomly generated instances, while for large instances, the VLSI problems result in a higher improvement.

6 Concluding Remarks

We addressed the problem of determining MSTs and SMTs when edge segment orientations are limited to a set of uniformly distributed orientations and where the coordinate system is permitted to rotate by any angle. We suggested a simple polynomial-time algorithm to solve the MST problem. We also provided some computational results indicating how big the savings can be. As it could be expected, the savings become negligible when λ and n grows. On the other hand, for all practical applications, λ is very small. Nets occurring in VLSI design are also rather small (in terms of the number of terminals involved). However, when many nets are to be routed, the overall savings will not be as impressive as for small isolated

	$n = 3$		$n = 4$		$n = 5$		$n = 10$	
λ	$T_{\max}$	$\alpha = 0$	$T_{\max}$	$\alpha = 0$	$T_{\max}$	$\alpha = 0$	$T_{\max}$	$\alpha = 0$
2	21.86	14.48	17.42	9.67	15.86	8.38	10.29	4.89
3	8.86	5.34	7.38	4.34	6.37	3.69	4.31	2.52
4	5.55	3.48	4.32	2.50	3.81	2.14	2.37	1.31
5	3.40	2.10	2.85	1.66	2.42	1.39	1.59	0.89
6	2.36	1.46	1.88	1.17	1.55	0.97	1.05	0.67
7	1.76	1.12	1.41	0.86	1.21	0.74	0.78	0.42
8	1.36	0.85	1.11	0.66	0.94	0.53	0.61	0.33
9	1.04	0.66	0.80	0.49	0.73	0.43	0.49	0.28
10	0.88	0.52	0.70	0.40	0.58	0.31	0.40	0.21
16	0.32	0.20	0.26	0.16	0.22	0.14	0.16	0.10
32	0.08	0.05	0.07	0.04	0.06	0.04	0.04	0.02

	$n = 20$		$n = 50$		$n = 100$	
λ	$T_{\max}$	$\alpha = 0$	$T_{\max}$	$\alpha = 0$	$T_{\max}$	$\alpha = 0$
2	7.47	3.19	5.03	2.12	3.46	1.51
3	3.01	1.47	1.70	0.78	1.23	0.61
4	1.72	0.93	1.02	0.51	0.72	0.39
5	1.09	0.58	0.72	0.39	0.47	0.25
6	0.73	0.39	0.44	0.21	0.30	0.16
7	0.57	0.32	0.35	0.19	0.24	0.14
8	0.44	0.24	0.25	0.13	0.18	0.09
9	0.30	0.16	0.20	0.11	0.15	0.08
10	0.26	0.13	0.18	0.09	0.11	0.05
16	0.11	0.06	0.07	0.03	0.04	0.02
32	0.03	0.01	0.02	0.01	0.01	0.01

Table 2: MST improvement in percent in relation to two values: The first one is the length of the worst case MST ($T_{\max}$) which is the longest MST found among 600 uniformly distributed values of α. The second one is the length of the MST when there is no rotation at all ($\alpha = 0$). All values are averages on 100 random instances.

nets. For $\lambda = 2$, the majority of 2-pin nets might be aligned along the horizontal and vertical lines already in the placement phase. If this is the case, the rotation will probably make no sense. If the rotation possibility is known during the placement phase and especially when λ is greater than 2, the need of placing 2-pin nets along horizontal or vertical lines will no longer be of the same importance as in the traditional Manhattan VLSI design. This can lead to fewer congestion problems and lower number of vias. However, it should be emphasized that such rotation is not applicable to current placements which places adjacent pins on the same vertical or horizontal line. In order to use rotation during routing it is necessary to develop new placement techniques taking in account possible postplacement rotations.

There are several other geometric combinatorial optimization problems which require a selection of a subset of edges and can in the rotational setting be approached in the same way as the MST problem. The *travelling salesman problem* and the *matching problem* are probably the most well-known. Another straightforward generalization occurs when the orientations are fixed but not necessarily evenly spaced. Determination of rotated MSTs in higher dimensions seems also to require a straightforward generalization of the 2-dimensional case.

	n=2	n=3	n=4	n=5	n=10	n=20	n=50	n=100
Random	21.21	11.46	8.19	7.00	3.17	2.17	1.28	0.92
VLSI	14.61	7.01	5.96	7.62	4.88	2.90	-	-

Table 3: RSMT improvement in percent obtained by rotating as compared to not rotating at all. Random: Randomly generated instances (averages over 100 instances). VLSI: VLSI design instances (averages over 100 instances).

Computing rotationally optimal SMTs for any fixed, but not necessarily uniform set of orientations is still partially an open research problem. However, if we may limit our attention to FSTs with at most one bent edge and having Steiner points with degree 3 only, Lemma 4.4 can immediately be applied — resulting in a finite time algorithm.

Acknowledgments

The authors would like to thank IBM and the Research Institute for Discrete Mathematics, University of Bonn, for providing industrial benchmarks circuits.

References

[1] M. Brazil, D. A. Thomas, and J. F. Weng. Minimum networks in uniform orientation metrics. *SIAM Journal on Computing*, 30:1579–1593, 2000.

[2] M. Brazil, D. A. Thomas, J. F. Weng, and M. Zachariasen. Canonical forms and algorithms for Steiner trees in uniform orientation metrics. Technical Report 02/22, DIKU, Department of Computer Science, University of Copenhagen, 2002.

[3] H. Chen, C. K. Cheng, A. B. Kahng, I. I. Mandoiu, Q. Wang, and B. Yao. The Y-architecture for on-chip interconnect: Analysis and methodology. In *Proceedings ACM/IEEE International Conference on Computer-Aided Design (ICCAD)*, pages 13–19, 2003.

[4] H. Chen, B. Yao, F. Zhou, and C. K. Cheng. The Y-architecture: Yet another on-chip interconnect solution. In *Proceedings Asia-Pacific Design Automation Conference*, pages 840–846, 2003.

[5] D. Cieslik. *The Steiner ratio*. Kluwer Academic Publishers, Boston, 2001.

[6] F. K. Hwang. A linear time algorithm for full Steiner trees. *Operations Research Letters*, 4(5): 235–237, 1986.

[7] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner tree problem*. Annals of Discrete Mathematics 53. Elsevier Science Publishers, Netherlands, 1992.

[8] B. K. Nielsen, P. Winter, and M. Zachariasen. On the location of Steiner points in uniformly-oriented Steiner trees. *Information Processing Letters*, 83(5):237–241, 2002.

[9] B. K. Nielsen, P. Winter, and M. Zachariasen. Rectilinear trees under rotation and related problems. In *Proceedings of the 18th European Workshop on Computational Geometry*, pages 18–22, 2002.

[10] B. K. Nielsen, P. Winter, and M. Zachariasen. An exact algorithm for the uniformly-oriented Steiner tree problem. *Algorithms - ESA 2002: 10th Annual European Symposium, Rome, Italy, September 17-21*, pages 237–266, 2002.

[11] K. J. Swanepoel. The local Steiner problem in normed planes. *Networks*, 36:104–113, 2000.

[12] D. M. Warme, P. Winter, and M. Zachariasen. GeoSteiner 3.1. Department of Computer Science, University of Copenhagen (DIKU), http://www.diku.dk/geosteiner/, 2001.

[13] P. Widmayer, Y. F. Wu, and C. K. Wong. On some distance problems in fixed orientations. *SIAM Journal on Computing*, 16(4):728–746, 1987.

[14] P. Winter and M. Zachariasen. Euclidean Steiner minimum trees: An improved exact algorithm. *Networks*, 30:149–166, 1997.

[15] XInitiative. X Initiative home page. `http://www.xinitiative.com`, 2001.

Published in *CompBioNets 2005: Algorithmics and Computational Methods for Biochemical and Evolutionary Networks*

Deferred path heuristic for phylogenetic trees revisited

Benny K. Nielsen* Stinus Lindgreen† Pawel Winter* Martin Zachariasen*

Abstract

We reexamine the deferred path heuristic originally developed by Schwikowski and Vingron in the late 1990s. Given a set of sequences, e.g. DNA, this heuristic simultaneously finds a multiple alignment and a phylogenetic tree. We give a unified description of the heuristic where such intermediate constructs as fork alignment graphs are made obsolete. Based on this description an efficient implementation is straightforward. We discuss two improvements of which one is intended to provide better quality solutions and one is intended to provide a substantial speed-up. This is confirmed by experiments on several real-life problem instances.

1 Introduction

One of the most important problems in biology is to explain how a group of species has evolved over time and how they are related to one another. This is usually done by means of *phylogenetic trees* where leaves represent the species while the interior nodes represent their hypothetical ancestors. The major problem that faces biologists when constructing phylogenetic trees is that the information about ancestors is very limited or indeed non-existent. So the problem is to infer the interior nodes and their relationship with each other and with the studied species.

Several models for the determination of phylogenetic trees have been considered over the years (see for example some of the monographs addressing mathematical and computational aspects of phylogenetic trees that appeared in recent years [1, 11]). From a computational point of view, most models are intractable as they lead to NP-hard problems. In order to obtain reasonable solutions even for relatively small problem instances, one therefore has to turn to heuristics where solutions can be obtained efficiently (but solutions are not necessarily optimal).

Many sequence based heuristics for finding phylogenetic trees are based on multiple alignments. Ideally, such an alignment should be based on the true evolutionary tree. This is not available, and instead some multiple alignment heuristics (such as ClustalW [13]) use a guide tree constructed by using pairwise alignment distances. This has the drawback of making the multiple alignment and subsequently the sought phylogenetic tree biased towards

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

†Bioinformatics Centre, University of Copenhagen, DK-2100 Copenhagen Ø, Denmark. E-mail: stinus@binf.ku.dk.

the guide tree. Alternatively, one can attempt to simultaneously find a multiple alignment and a phylogenetic tree. This is known as the problem of *generalized tree alignment*.

In this paper we reexamine the *deferred path heuristic* for generalized tree alignment. It was originally developed by Schwikowski and Vingron in the late 1990s [8–10] and their work was based on the idea of using *sequence graphs* introduced by Hein [3]. We give a unified description of the heuristic where such intermediate constructs as fork alignment graphs used in [9] are made obsolete. We discuss improvements of the heuristic which either result in better quality solutions or in a substantial speed-up. Finally, we test the heuristic on several real-life problem instances.

The paper is organized as follows. In Section 2 we give a general introduction to the concepts of tree alignment and generalized tree alignment. We also discuss how these concepts are related to pairwise and multiple alignment. Sequence graphs — which form the backbone of the deferred path heuristic — and the heuristic itself are described in Section 3. Some new improvements are discussed in Section 4. Computational experience with some real-life problem instances are covered in Section 5. Conclusions and ideas for further research are discussed in the closing Section 6.

2 Phylogenetic Trees and Sequence Alignment

Construction of phylogenetic trees for a set of species represented by DNA, RNA- or protein sequences is intimately related to the alignments of such sequences. In this section we briefly discuss these alignment problems and their relation to the construction of phylogenetic trees.

2.1 Tree Alignment

Consider a phylogenetic tree $T = (V, E)$ with n leaves where V denotes the set of nodes and E denotes the set of edges. Each leaf of T corresponds to a sequence S_j over a finite alphabet Σ (for example nucleotides in DNA or amino acids in a protein). In this paper we address the problem of determining *unrooted* phylogenetic trees. Identifying a root of a phylogenetic tree is non-trivial (and in fact it is a controversial issue in the "evolutionary" community). Neither do we address evolutionary time issues which often are associated with the construction of phylogenetic trees.

The internal nodes of the phylogenetic tree T are assumed to have degree 3 (realistic assumption for most practical applications). T has therefore $n-2$ internal nodes. If sequences are assigned to all internal nodes, the cost of the tree can be defined as the sum of costs of optimal pairwise alignments (see below) between sequences in adjacent nodes. The *tree alignment* problem for a given tree T is therefore to assign sequences to the internal nodes such that the cost of the tree is minimized.

2.2 Pairwise Alignment

In *pairwise alignment* one considers two sequences S_i and S_j over a finite alphabet Σ. The *gap* symbol "$-$" is a special character not in Σ. Σ extended with the gap symbol is denoted by Σ'. An *alignment* of sequences S_i and S_j is a pair of equal-length sequences S'_i and S'_j over Σ' such that the removal of gaps would yield S_i and S_j, respectively. The sequences S'_i and S'_j can be placed in an *alignment matrix* A (with two rows) such that each column identifies a pair of symbols from Σ' aligned with each other. A column of A that consists of two gaps is

called a *gap column*. Gap columns are normally irrelevant in pairwise alignment and can be removed from consideration. Alignments without gap columns are called *reduced* alignments. All alignments discussed in this paper are reduced alignments.

In this paper, we consider alignments in a minimization framework. In order to measure the quality of a given pairwise alignment, *cost* values are defined for all pairs of symbols in Σ'. Given $s_1, s_2 \in \Sigma'$, we denote the corresponding cost value $\delta(s_1, s_2)$. It is also assumed that $\delta(s_1, s_2) = 0$ if $s_1 = s_2$. In the simplest version, the cost of a pairwise alignment is the sum of costs taken over all columns of A. An *optimal pairwise alignment* is a pairwise alignment with the lowest possible cost. More than one optimal pairwise alignment can exist for the same pair of sequences.

More sophisticated measures incorporate the observation that contiguous gaps of length k are more likely to occur in sequences than k isolated gaps. So the cost of a pairwise alignment consists of two components. The cost of columns with no gaps is still the sum of their individual costs. A *maximal gap* in either S'_i or S'_j is a maximal contiguous sequence of gaps. Each maximal gap in a pairwise alignment contributes to the total cost of the pairwise alignment by the affine gap penalty $g(k) = a + b \cdot k$, where k is the length of the gap, and a and b are the gap open and gap extension penalties, respectively.

Optimal pairwise alignment is a well-understood optimization problem which can be solved in quadratic time by standard dynamic programming techniques [5].

2.3 Multiple Alignment

In *multiple alignment* the problem is to find an optimal alignment of n sequences, $n \geqslant 2$. In this case, the alignment matrix A has n rows. Most cost functions for multiple alignment are based on column costs and affine gap penalties. Tree alignment has also been suggested as a possible method of evaluating multiple alignments. Unfortunately, this approach is intractable unless the tree has a very simple topology (for example a star topology with one internal node adjacent to all leaf nodes). Even for trees with internal nodes of degree 3, the tree alignment problem is NP-hard. The cost scheme used in the proof of this result was a metric cost scheme with values 0, 1, and 2 and a 4-letter alphabet [4]. Hence, using tree alignment to determine the cost of a multiple alignment seems practically infeasible even for relatively short sequences.

2.4 Generalized Tree Alignment

Generalized tree alignment is an important and non-trivial extension of the tree alignment problem. Furthermore, it can be used to find good quality multiple alignments. Given n sequences $S_1, S_2, \ldots, S_n$, the problem is to find a tree T such that its tree alignment cost is smallest possible (for some appropriately defined pairwise alignment cost function). Once the tree and the sequences in internal nodes are determined, optimal pairwise alignments on adjacent nodes in the tree can be used to obtain a good quality multiple alignment. Unfortunately, the generalized tree alignment problem is not only NP-hard but in fact it is *MAX* SNP-hard [14]. As a consequence, neither efficient exact algorithms nor polynomial approximation algorithms with arbitrarily small error ratio are likely to exist. Use of heuristics seems therefore to be the only feasible way to solve this problem computationally.

The generalized tree alignment problem is closely related to the *Steiner tree problem in graphs*: Given a graph $\mathcal{N} = (\mathcal{W}, \mathcal{F})$ with positive edge costs and a subset $\mathcal{S}$ of the vertices,

the problem is to find a minimum-cost connected subnetwork of $\mathcal{N}$ spanning $\mathcal{S}$. It is obvious that a solution must be a tree. It may contain other vertices of $\mathcal{W}$ than those in $\mathcal{S}$. These additional vertices are usually referred to as *Steiner vertices*. It is straightforward to see that the generalized tree alignment can be formulated as the Steiner tree problem in a (very big) graph. More specifically, let $\mathcal{W}$ represent the set of all sequences over the alphabet Σ. If the lengths of sequences are not bounded, the graph will have an infinite number of vertices. But in practical applications the lengths can be bounded by a small multiple of the longest sequence among $S_1, S_2, \ldots, S_n$. Two vertices in $\mathcal{N}$ are connected by an edge if the corresponding sequences can be obtained from one another by a substitution of one letter, or by insertion or deletion of one or several contiguous letters. The cost of such an edge is of course the cost of the corresponding substitution, insertion or deletion. If we let $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ be the sequences of the set of the given species, the generalized tree alignment is exactly the Steiner tree problem in the *Steiner graph* $\mathcal{N} = (\mathcal{W}, \mathcal{F})$.

3 Deferred Path Heuristic

The main purpose of this section is to give a description of the basic heuristic. The description is fairly detailed and for the sake of clarity, the initial description does not include so-called *detours* or affine gap penalties. The use of detours is instead postponed to Section 3.4 in which affine gap penalties are also briefly discussed.

3.1 Tree Construction

A *sequence graph* $G = (U, A)$ is a connected directed acyclic graph with a unique *source* vertex s (no incoming arcs) and a unique *sink* vertex t (no outgoing arcs). Each arc $a \in A$ has a label s_a taken from a finite alphabet Σ'. The set of all source-sink paths defines a set of sequences, each consisting of the consecutive labels of the arcs on a path. Our interest in sequence graphs stems from the fact that they are extremely well-suited to represent sets of related sequences in a compact manner. Sequence graphs were originally suggested by Hein [3].

Note that the vertices in a sequence graph can be sorted topologically, i.e., if given a graph with $m + 1$ vertices we can make a numbering of the vertices using the integers from 0 to m such that if we refer to an arc a by its end-vertices, (i, j), we can safely assume that $i < j$. Furthermore, we can also assume that the source is assigned 0 and the sink is assigned m.

Suppose that n sequences $S_1, S_2, \ldots, S_n$ over a finite alphabet Σ are given. The objective is to find a phylogenetic tree $T = (V, E)$ with as low tree alignment cost as possible. The basic idea of the heuristic is to represent these n sequences by linear sequence graphs with labels on their unique paths from the source to the sink corresponding to the symbols in the sequences. These sequence graphs will correspond to leaf nodes in the phylogenetic tree $T = (V, E)$ that we want to construct. The heuristic then selects two sequence graphs $G = (U_G, A_G)$ and $H = (U_H, A_H)$ that are "closest" to each other. These two sequence graphs are replaced by another appropriately defined sequence graph $G \otimes H$. An internal node is added to T and it is connected by edges to the nodes corresponding to G and H. This process is repeated until only two sequence graphs remain. The corresponding nodes in T are connected by an edge and the construction of T is completed.

As already pointed out, during each iteration of the heuristic a pair of sequence graphs G and H is replaced by a new sequence graph $G \otimes H$. This new sequence graph can be defined in several ways. In the simplest approach, $G \otimes H$ would be a linear sequence graph representing

exactly one sequence S such that the sum of costs of optimal pairwise alignments between S and the two sequences represented by G and H is minimized.

The problem with such an approach is of course that the choice of S can prove to be very bad in subsequent iterations of the heuristic. Sequence graphs are fortunately capable of representing several sequences in a compact way. The idea is therefore to let $G \otimes H$ represent several sequences and to *defer* the selection of a particular sequence. Suppose that S_G and S_H are sequences from G and H such that the cost of their optimal pairwise alignment is the smallest possible among all pairs of sequences, one from G and one from H. The cost of such an alignment is called the *distance* between G and H and we refer to the corresponding pair of sequences as an *optimal pair*. We can interpret such a pairwise alignment as a shortest path in the Steiner graph $\mathcal{N}$ where intermediate vertices correspond to sequences that all could act as S. Hence, it is natural to let $G \otimes H$ contain them all (including S_G and S_H). In fact, there could be several optimal pairs of sequences, one sequence of such a pair in G and the other sequence in H. So $G \otimes H$ can be extended to contain all sequences on the corresponding shortest paths in the Steiner graph $\mathcal{N}$. The problem of finding all such alignments can be solved, as we will see, by a straightforward dynamic programming algorithm.

Once the number of active sequence graphs has been reduced to two, a single optimal pair of sequences can be selected for the remaining pair of graphs. These sequences together with the tree T can be used to backtrack through preceding sequence graphs and to select appropriate sequences in each of them (using optimal pairs).

When sequence graphs represent several sequences and the choice of a particular sequence can be deferred as explained above, the heuristic will normally produce better solutions than the simple method using linear sequence graphs (representing one sequence only).

3.2 Distances between Sequences Graphs

To be able to select the closest pair of sequence graphs we naturally need to be able to compute the distance between two sequence graphs as defined above. The classic approach for finding the distance between two simple sequences of symbols is a dynamic programming algorithm. Such an algorithm uses the fact that the distance between two sequences can be based on the distances between the immediate prefixes of the sequences, i.e. the problem exhibits the optimal substructure property. This is also true for sequence graphs.

Given two sequence graphs G and H of length m and n, respectively, we denote the distance between them $\delta(G, H)$. Let $G^{\triangleleft}(i)$ denote the *prefix subgraph* of G consisting of all paths from the source 0 to a given vertex i and let $H^{\triangleleft}(k)$ denote a similar prefix subgraph of H. Now $\delta(G^{\triangleleft}(i), H^{\triangleleft}(k))$ is the distance between prefix graphs $G^{\triangleleft}(i)$ and $H^{\triangleleft}(k)$.

The distance between G and H can be found by iteratively filling out the values of a matrix D of size $(m+1) \times (n+1)$ as shown in Algorithm 4. Each entry in the matrix corresponds to a pair of vertices from G and H. Starting from the pair of sources, matrix values are updated by calculating the cost of aligning all outgoing arcs from the current vertices, i and k, with each other and with gaps. The calculations in the inner loops are based on the value of $D[i, k]$. Due to the topological sorting of the vertices we know that all paths to i and k have been handled earlier in the algorithm and thus $D[i, k]$ must be equal to $\delta(G^{\triangleleft}(i), H^{\triangleleft}(k))$. In particular, $D[m, n] = \delta(G^{\triangleleft}(m), H^{\triangleleft}(n)) = \delta(G, H)$.

As previously noted, a sequence of symbols can be represented by a simple linear sequence graph. Note that in this case Algorithm 4 reduces to a standard dynamic programming algorithm for aligning two sequences. An example of the use of Algorithm 4 for two linear

Algorithm 4: Graph variation of the classic pairwise alignment algorithm.

```
Given graphs G and H with vertices 0 to m and 0 to n;
Allocate matrix D[m + 1, n + 1] with all entries set to ∞;
D[0, 0] = 0;
for i = 0 to m do
    for k = 0 to n do
        foreach arc a = (i, j) in G do
            foreach arc b = (k, l) in H do
                D[j, l] ← min(D[j, l], D[i, k] + δ(s_a, s_b));
        foreach arc a = (i, j) in G do
            D[j, k] ← min(D[j, k], D[i, k] + δ(s_a, −));
        foreach arcs b = (k, l) in H do
            D[i, l] ← min(D[i, l], D[i, k] + δ(−, s_b));
return D[m, n]
```

sequence graphs can be seen in Figure 1 where the final values of the D-matrix are given. Figure 1 also illustrates that one could easily store the origins of each entry in the D-matrix and thus generate all possible optimal alignments.

Each pair of edges is compared exactly once in Algorithm 4 thus the running time is $O(|A_G| \cdot |A_H|)$. The space requirement is dominated by the D-matrix and is $O(|U_G| \cdot |U_H|)$.

3.3 All Shortest Paths Sequence Graph

When a pair of sequence graphs, $G = (U_G, A_G)$ and $H = (U_H, A_H)$, has been selected the next step is to create a new sequence graph $G \otimes H$ representing all sequences which are on shortest paths between G and H in the Steiner graph $\mathcal{N}$.

As noted the D-matrix used in Algorithm 4 could easily be extended to save the paths corresponding to all possible optimal alignments (Figure 1). Any combination of the symbols aligned on these paths are then among the sequences we would like to include in the new sequence graph. In the following we describe how to do this efficiently.

Given a vertex i in G and a vertex k in H we have already defined the prefix distance $\delta(G^{\triangleleft}(i), H^{\triangleleft}(k))$ and we have also seen that it is implicitly computed in Algorithm 4. Another set of values has to be computed before the combined sequence graph $G \otimes H$ can be constructed. Let $G^{\triangleright}(j)$ denote the *suffix subgraph* of G consisting of all paths from j to the sink m and let $H^{\triangleright}(l)$ denote a similar suffix subgraph of H. It should be clear that the distance values $\delta(G^{\triangleright}(j), H^{\triangleright}(l))$ for all pairs (j, l) can be computed by Algorithm 4 simply by reversing the directions of all arcs in the graphs.

Now, the vertices of $G \otimes H$ will belong to the set $U_{G \otimes H} \subseteq U_G \times U_H$. More precisely, vertices are added to $G \otimes H$ for all vertex pairs $i \otimes k$ for which $\delta(G^{\triangleleft}(i), H^{\triangleleft}(k)) + \delta(G^{\triangleright}(i), H^{\triangleright}(k)) = \delta(G, H)$. In particular, the source of $G \otimes H$ will be the vertex $0 \otimes 0$ (the pair of sources in G and H) and the sink of $G \otimes H$ is the vertex $m \otimes n$ (the pair of sinks in G and H). An example of such a vertex set can be seen in Figure 2a in which the values of $\delta(G^{\triangleleft}(i), H^{\triangleleft}(k))$ and $\delta(G^{\triangleright}(i), H^{\triangleright}(k))$ are included.

The arcs of $G \otimes H$ are constructed as follows. Given two vertices $i \otimes k$ and $j \otimes l$ in $G \otimes H$, consider an arc $a = (i, j)$ from G and an arc $b = (k, l)$ from H. Let s_a denote the label associated with a and let s_b denote the label associated with b. An arc between $i \otimes k$ and $j \otimes l$

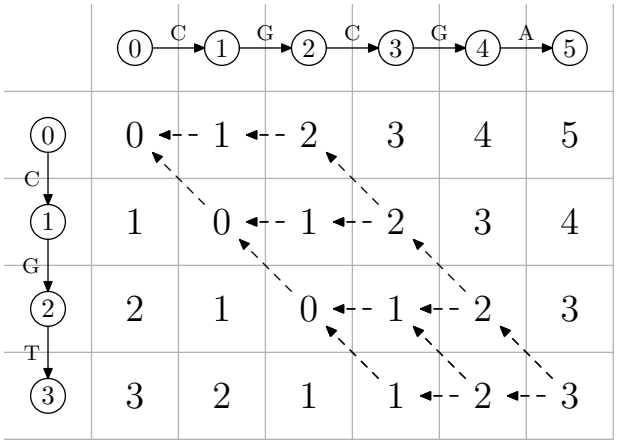


Figure 1: Example of the D-matrix of Algorithm 4 for a simple unit cost scheme i.e. $\delta(s_1, s_2) = 1$ for all $s_1 \neq s_2$. The value in the lower right corner of the matrix (3) is the cost of an optimal alignment. The arrows show all the possible ways back through the matrix corresponding to all possible optimal alignments, e.g. the rightmost path corresponds to aligning `CGCGA` with `--CGT`.

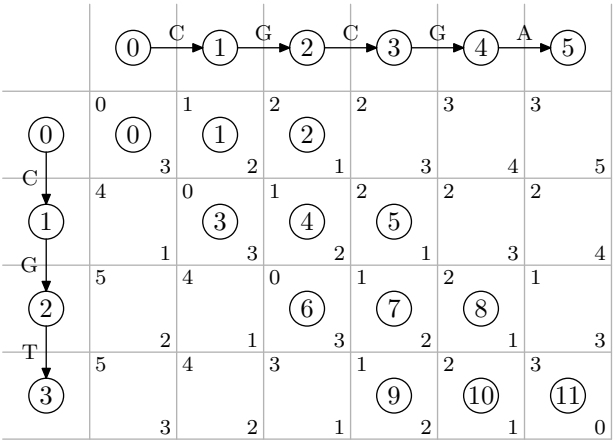
labeled s, $s \in \Sigma'$, is added to $G \otimes H$ if there is a sequence S_G in G and a sequence S_H in H which can be aligned with each other such that labels of a and b are in the same column and the cost of the alignment is exactly $\delta(G, H)$. Any such lowest cost alignment requires that the portion of S_G in the prefix graph $G^{\triangleleft}(i)$ is (optimally) aligned with the portion of S_H in the prefix graph $H^{\triangleleft}(k)$, the label of a is aligned with the label of b and the portion of S_G in the suffix graph $G^{\triangleright}(j)$ is optimally aligned with the portion of S_H in the suffix graph $H^{\triangleright}(l)$. A necessary and sufficient requirement for such a bounded cost alignment to exist is that $\delta(G^{\triangleleft}(i), H^{\triangleleft}(k)) + \delta(s, s_a) + \delta(s, s_b) + \delta(G^{\triangleright}(j), H^{\triangleright}(l))$ is equal to $\delta(G, H)$.

When constructing $G \otimes H$, one also has to decide if there are optimal alignments such that the arc $a = (i, j)$ is aligned with a gap while a subsequence of the prefix graph $G^{\triangleleft}(i)$ is aligned with a subsequence of the prefix graph $H^{\triangleleft}(k)$ for some vertex k of H and a subsequence of the suffix graph $G^{\triangleright}(j)$ is aligned with a subsequence of the suffix graph $H^{\triangleright}(k)$. If $\delta(G^{\triangleleft}(i), H^{\triangleleft}(k)) + \delta(s, s_a) + \delta(s, -) + \delta(G^{\triangleright}(j), H^{\triangleright}(k))$ is equal to $\delta(G, H)$, then an arc between $i \otimes k$ and $j \otimes k$ with label s is added to $G \otimes H$. Note that in the case of affine gap penalties, the situation becomes more complicated as one has to take into account that several symbols can be aligned with contiguous gap symbols.

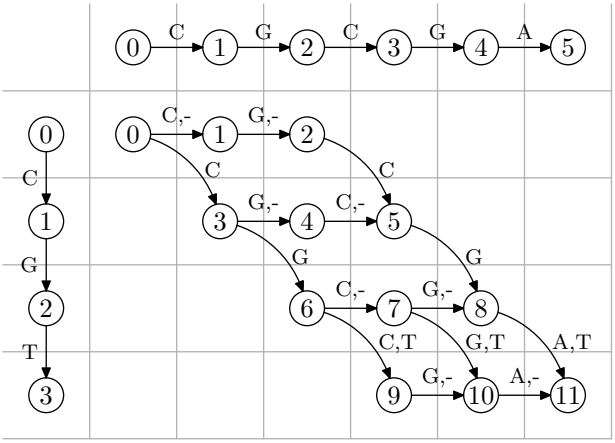
The symmetric situation where an arc from H is to be aligned with a gap symbol is dealt with in a similar manner. This and more details can be seen in Algorithm 5. The running example is completed in Figure 2b in which the vertices found in Figure 2a are supplemented by the arcs described above. Both running time and space requirement is easily seen to be $O(|A_G| \cdot |A_H|)$.

3.4 Detours

As described by Schwikowski and Vingron [9], it is still possible to further improve the flexibility of the heuristic by letting sequence graphs include suboptimal sequences. To facilitate this, *weighted* sequence graphs are introduced. A *weighted* sequence graph is a sequence graph $G = (U, A)$ where each arc a also has a weight w_a. This weight is a measure of the minimum



(a)



(b)

Figure 2: Continuation of the example in Figure 1 showing the use of Algorithm 5. (a) The 12 vertices used in a sequence graph representing all sequences on all shortest paths between the two original sequences in the Steiner graph $\mathcal{N}$. Upper left corner values are prefix distances and lower right corner values are suffix distances. For each vertex the sum of the corner values is equal to 3. (b) The final sequence graph. Note that only one arc is drawn for a connected pair of vertices, but in some cases with multiple symbols. Algorithm 5 would create an arc for each symbol.

Algorithm 5: Construction of a sequence graph containing all sequences on shortest paths between two given sequence graphs

Given graphs $G = (U_G, A_G)$ and $H = (U_H, A_H)$ with vertices 0 to m and 0 to n;
Initialize an empty graph $G \otimes H$. Vertices will be a subset of $U_G \times U_H$;
Add vertices to $G \otimes H$ for all pairs $i \otimes k$ for which $\delta(G^{\triangleleft}(i), H^{\triangleleft}(k)) + \delta(G^{\triangleright}(i), H^{\triangleright}(k)) = \delta(G, H)$;
foreach vertex pair $i \otimes k$ in $G \otimes H$ **do**
  $\delta^{\triangleleft} \leftarrow \delta(G^{\triangleleft}(i), H^{\triangleleft}(k))$;
  // Symbols on arcs from both graphs aligned;
  foreach arc $a = (i, j)$ in G **do**
    foreach arc $b = (k, l)$ in H **do**
      foreach symbol $s \in \Sigma'$ **do**
        if $\delta^{\triangleleft} + \delta(s, s_a) + \delta(s, s_b) + \delta(G^{\triangleright}(j), H^{\triangleright}(l)) = \delta(G, H)$ **then**
          Add/update arc with symbol s from $i \otimes k$ to $j \otimes l$ in $G \otimes H$;
  // Symbols on arcs from one graph aligned with gaps;
  foreach arc $a = (i, j)$ in G **do**
    foreach symbol $s \in \Sigma'$ **do**
      if $\delta^{\triangleleft} + \delta(s, s_a) + \delta(l, -) + \delta(G^{\triangleright}(j), H^{\triangleright}(k)) = \delta(G, H)$ **then**
        Add/update arc with symbol s from $i \otimes k$ to $j \otimes k$ in $G \otimes H$;
  foreach arc $b = (k, l)$ in H **do**
    foreach symbol $s \in \Sigma'$ **do**
      if $\delta^{\triangleleft} + \delta(l, -) + \delta(s, s_b) + \delta(G^{\triangleright}(i), H^{\triangleright}(l)) = \delta(G, H)$ **then**
        Add/update arc with symbol s from $i \otimes k$ to $i \otimes l$ in $G \otimes H$;
return $G \otimes H$

additional cost caused by using this arc in a path from source to sink compared to an optimal path. The weight of a source-sink path in G is defined as the sum of weights of its arcs. In particular, an optimal path only contains 0-weight arcs.

The *weighted distance* between a pair of sequences, one in a weighted sequence graph G and the other in a weighted sequence graph H is defined as the distance between them supplemented by the weights of their paths. The *weighted distance* between G and H is defined as the smallest weighted distance between pairs of sequences, one in G and one in H. This is also denoted by $\delta(G, H)$.

Let Δ be a nonnegative real number. In the *deferred path heuristic with detours*, the weighted sequence graph $G \otimes H$ will contain all sequences S where the sum of the weighted distances from S to a sequence S_G in G and a sequence S_H in H is at most Δ away from the weighted distance between G and H. In other words, $G \otimes H$ will contain all sequences which are at most Δ away from the shortest paths between sequences of an optimal pair from G and H in the Steiner graph $\mathcal{N}$.

The algorithms presented so far need surprisingly few changes to be able to handle the Δ parameter. The distance algorithm simply includes the arc weights introduced when calculating the values of the matrix entries. The changes to the construction algorithm are also quite simple. Some of the changes needed are given in Algorithm 6. Especially note the calculation of the arc weight. When adding an arc a from $i \otimes k$ to $j \otimes l$, the weight is computed by adding the costs of the arc to the cost $\delta(G^{\triangleright}(j), H^{\triangleright}(l))$ of an optimal path from the sink to $j \otimes l$, and then subtracting the cost $\delta(G^{\triangleright}(i), H^{\triangleright}(k))$ of an optimal path from the sink to $i \otimes k$.

Algorithm 6: Some of the changes needed when constructing a sequence graph including detours.

```
... Add vertices to G ⊗ H for all pairs i ⊗ k for which δ(G^◁(i), H^◁(k)) +
δ(G^▷(i), H^▷(k)) ⩽ δ(G, H) + Δ;
foreach arc a = (i, j) in G do
    foreach arc b = (k, l) in H do
        foreach symbol s ∈ Σ' do
            if δ^◁ + δ(s, s_a) + δ(s, s_b) + δ(G^▷(j), H^▷(l)) ⩽ δ(G, H) + Δ then
                w ← δ(G^▷(j), H^▷(l)) + δ(s, s_a) + w_a + δ(s, s_b) + w_b − δ(G^▷(i), H^▷(k))
                Add/update arc with symbol s and weight w from i⊗k to j⊗l in G ⊗ H;
...
```

It it obvious that as Δ increases, so does the number of sequences (represented by paths) in weighted sequence graphs. So the choice of an appropriate Δ is crucial. If chosen too low, the quality of the solution can be poor. If chosen too high, the sizes of weighted sequence graphs can explode. It should also be noted that increasing Δ can also result in worse results for the deferred path heuristic with detours — even for a fixed tree topology. This is due to the fact that a Δ_1 sequence graph in an internal node of such a fixed tree is not guaranteed to be a subgraph of the corresponding Δ_2 sequence graph even if $\Delta_2 > \Delta_1$.

It is possible to extend all of the algorithms described to also work for affine gap penalties without changing the running time with more than a constant factor [8]. The idea is basically the same as for making basic pairwise alignments with affine gap penalties [2]. In short, one needs three values in each entry of the matrix corresponding to each of the possible gap-insertions (a gap in one of the sequences or no gap at all). When building a new sequence graph each of these values can cause a vertex to be created.

```
AGTGAGTCATG      A-GTGAGTCATG-    AGTG  | AGT | CATG
TGCAGTAGGT       T-G-CAGT-AGGT    TGC   | AGT | AGGT
TCTTCAGTGCC      TCTTCAGT--GCC    TCTTC | AGT | GCC
```

(a) (b) (c)

Figure 3: Segmentation example.

4 Improvements

In the following we suggest two improvements to the deferred path heuristic with detours. The first one focuses on decreasing the amount of space and time used by the heuristic with limited sacrifice of solution quality and the second one focuses on increasing solution quality at the cost of time.

4.1 Segmentation

The rapidly increasing size of sequence graphs with increasing Δ can be limited by pruning edges and nodes. Schwikowski and Vingron [9] enforces a limit of 1000 edges by randomly removing edges from any sequence graphs generated. In the following we suggest an alternative approach which we call *segmentation*. The basic idea is to heuristically split the set of input sequences into a series of sets and then work on each set in the series individually when measuring distances and creating new sequence graphs.

Consider the example given in Figure 3a. Three input sequences are given. If solving this problem without using detours ($\Delta = 0$) we get the alignment in Figure 3b. In the middle of the sequences we notice a no-cost alignment of 3 consecutive symbols. When solving the problem with detours these symbols are not likely to be aligned differently and therefore we could fix the location of this segment and instead solve the problem for three smaller sets of sequences as indicated in Figure 3c.

In the example above, a sequence of 3 consecutive perfect columns was used to fix a segment. In general this approach could be parameterized by requiring a minimum of σ consecutive perfect columns. A large value of σ would cause less segments to be found, but also decrease the risk of missing good solutions.

Using perfect columns is not the only possible strategy. Numerous other strategies could be used and one could even use some fast multiple alignment method such as ClustalW [13] to get the initial alignment and then use this to also fix segments when solving the problem for $\Delta = 0$.

4.2 Tree Construction Strategies

As previously described, the basic tree construction heuristic simply makes a greedy choice in each iteration, selecting the sequence graphs which are "closest" in terms of distance. This is also the only strategy considered by Schwikowski and Vingron [9]. The greedy choice is not necessarily the best choice. This is illustrated with a small example in Figure 4 in which the best solution is based on a non-greedy initial choice.

An exhaustive search of all tree topologies is not a viable alternative for most problem instances, but other approaches could be considered. Here we suggest a simple *look ahead*

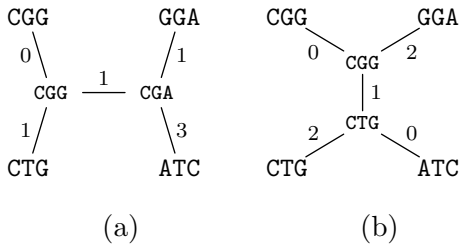


Figure 4: Example with the two possible topologies when solving a problem with four sequences. (a) Using a unit cost scheme an initial greedy choice is to join `CGG` and `CTG` at a cost of 1. The final solution then has a total cost of 6. (b) A non-greedy choice at a cost of 2 results in a better final solution with a total cost of 5.

	Size	Length			Origin
		Min.	Max.	Avg.	
`Sankoff`	9	118	122	120.1	[7]
`rRNA`	602	96	122	116.6	[12]
`SRP_euk`	71	256	317	295.3	[6]

Table 1: Data instances used in the experiments.

scheme in which we improve the greedy choice by extending the search in both width and depth. The idea is to make several choices and then continue the heuristic for each of them comparing the choices at a later stage. Three parameters can be used to describe this scheme. The number of levels to look ahead (the depth), λ_d, before comparing the choices made, the number of (best) pairs of sequence graphs to consider at each level (the width), λ_w, and the number of steps, λ_s, to actually take when the best sequence of choices has been found. The normal greedy choice would correspond to $\lambda_d = 0, \lambda_w = 1$ and $\lambda_s = 1$.

5 Computational Experiments

An implementation of the deferred path heuristic with detours has been done in C++ and the experiments have been executed on a 3GHz Pentium with 1 MB level 2 cache and 3.5 GB of main memory. The source code is available on request.

The main purpose of the computational experiments is to see if the segmentation scheme and the look ahead scheme behave as expected. Three data instances are used in the experiments (Table 1), all containing sequences with nucleotides. The classic `Sankoff` instance only contains 9 sequences and it is included to verify that we obtain the same solution as Schwikowski and Vingron [9]. The other instances contain far too many sequences to be handled by our implementation since the sequence graphs would quickly become too big. Instead we select 10 random sets of 10 sequences from each instance and report average results. This should also ensure that we get results which to some extent indicate the general behaviour of the heuristic and the new improvements.

The cost matrix used for all experiments is the same as the one originally used for the `Sankoff` instance, i.e., $\delta(\mathtt{A},\mathtt{G}) = \delta(\mathtt{C},\mathtt{T}) = 1$, $\delta(\mathtt{A},\mathtt{C}) = \delta(\mathtt{A},\mathtt{T}) = \delta(\mathtt{C},\mathtt{G}) = \delta(\mathtt{G},\mathtt{T}) = 1.75$ and linear gap cost $g(k) = 2.25k$. All experiments are done with Δ ranging from 0 to

3 and segmentation with $\sigma \in 2, 3, 6, \infty$, where ∞ means no segmentation. Only two tree construction schemes are tested. The standard greedy approach and a lookahead scheme with $\lambda_w = 3$, $\lambda_d = 3$ and $\lambda_s = 1$.

The results are presented in Table 2. The best known result for the `Sankoff` instance [9] is found in all cases when $\Delta \geqslant 2$. In general, results are not becoming worse when using segmentation, but these instances also only allow few segments. An average of 9 segments for the `rRNA` instance when $\sigma = 2$ reveals a considerable decrease in running time which is an indication of the potential of segmentation, but other segmentation strategies should be considered to increase the number of segments. The look ahead scheme works as expected finding better solutions in most cases, but the cost in running time is large compared to the results obtained using large Δ values. More experiments are needed to further analyze this time/quality tradeoff.

6 Conclusions

We have given a short and concise description of the deferred path heuristic with detours and we have shown that there is still room for improving both computation speed and quality of solutions. We have also taken the first steps to further examine the potential of using sequence graphs for generalized tree alignment and it is our intention to continue this work and to improve the implementation to make it of practical use. Most interesting is to find new ways of limiting the size of the sequence graphs without sacrificing solution quality.

More experiments are needed, especially a comparison of the quality of our phylogenetic trees with the trees constructed by the best existing solution methods.

References

[1] J. Felsenstein. *Inferring phylogenies*. Sinauer Associates, Inc., Sunderland, MA, 2003.

[2] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, 1982.

[3] J. Hein. A tree reconstruction method that is economical in the number of pairwise comparisons used. *Molecular Biology and Evolution*, 6:669–684, 1989.

[4] T. Jiang, E. L. Lawler, and L. Wang. Aligning sequences via an evolutionary tree: Complexity and approximation. In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 760–769. ACM Press, 1994. ISBN 0-89791-663-8.

[5] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[6] M. A. Rosenblad, J. Gorodkin, B. Knudsen, C. Zwieb, and T. Samuelsson. SRPDB: Signal Recognition Particle Database. *Nucleic Acids Research*, 31(1):363–364, 2003.

[7] D. Sankoff, R. J. Cedergren, and G. Lapalme. Frequency of insertion-deletion, transversion, and transition in evolution of 5S ribosomal RNA. *Journal of Molecular Evolution*, 7:133–149, 1976.

[8] B. Schwikowski. A new algorithmic approach to the construction of multiple alignments and evolutionary trees. Technical Report 11, German National Center for Information Technology, 1998.

	Greedy			Look ahead		
	$\sigma \geqslant 6$	$\sigma = 3$	$\sigma = 2$	$\sigma \geqslant 6$	$\sigma = 3$	$\sigma = 2$
`Sankoff`		Cost			Cost	
$\Delta = 0$	299.50	299.50	299.50	299.50	299.50	299.50
$\Delta = 1$	298.00	298.00	298.00	296.75	296.00	296.00
$\Delta = 2$	295.75	295.75	295.75	295.75	295.75	295.75
$\Delta = 3$	295.75	295.75	295.75	295.75	295.75	295.75
		Seconds			Seconds	
$\Delta = 0$	0.07	0.07	0.07	0.63	0.63	0.62
$\Delta = 1$	0.11	0.07	0.04	0.93	0.62	0.45
$\Delta = 2$	0.20	0.13	0.10	2.04	1.21	0.92
$\Delta = 3$	0.51	0.31	0.24	5.66	3.12	2.37
#segs	1.00	5.00	11.00	1.00	5.00	11.00
`rRNA`		Cost			Cost	
$\Delta = 0$	391.85	391.85	391.85	387.27	387.27	387.27
$\Delta = 1$	383.43	383.43	383.75	382.95	382.95	382.68
$\Delta = 2$	382.73	382.73	383.95	382.52	382.52	382.45
$\Delta = 3$	382.30	382.30	382.62	381.25	381.25	381.18
		Seconds			Seconds	
$\Delta = 0$	0.10	0.11	0.11	1.03	1.25	0.83
$\Delta = 1$	0.13	0.12	0.07	1.77	1.92	0.72
$\Delta = 2$	0.27	0.26	0.15	5.41	6.01	1.99
$\Delta = 3$	0.91	0.89	0.49	24.36	24.63	8.01
#segs	1.00	2.00	9.40	1.00	2.00	9.00
`SRP_euk`		Cost			Cost	
$\Delta = 0$	1193.08	1193.08	1193.08	1184.62	1184.62	1184.62
$\Delta = 1$	1169.55	1169.55	1169.22	1163.65	1163.40	1164.15
$\Delta = 2$	1159.47	1159.47	1160.33	1154.55	1154.33	1152.75
$\Delta = 3$	1155.60	1155.60	1156.40	1151.40	1151.42	1151.65
		Seconds			Seconds	
$\Delta = 0$	0.61	0.61	0.61	6.05	6.05	6.02
$\Delta = 1$	1.40	1.06	0.91	16.06	9.87	9.26
$\Delta = 2$	4.86	3.59	3.04	68.42	42.33	40.88
$\Delta = 3$	23.16	17.50	14.75	357.23	215.61	205.51
#segs	1.00	4.00	5.80	1.00	4.80	6.60

Table 2: Results from the computational experiments. The `Sankoff` data is the result of a single instance with 9 sequences. The rest of the data are averages of 10 runs on instances with 10 sequences. The results in the columns with $\sigma \geqslant 6$ correspond to not using segmentation at all since none of the data instances can be split into segments when requiring 6 or more consecutive no-cost columns.

[9] B. Schwikowski and M. Vingron. Weighted sequence graphs: Boosting iterated dynamic programming using locally suboptimal solutions. *Discrete Applied Mathematics*, 127(1):95–117, 2003. ISSN 0166-218X.

[10] B. Schwikowski and M. Vingron. The deferred path heuristic for the generalized tree alignment problem. *Journal of Computional Biology*, 4(3):415–431, 1997.

[11] C. Semple and M. Steel. *Phylogenetics*. Oxford University Press, New York, 2003.

[12] M. Szymanski, M. Z. Barciszewski, V. A. Erdmann, and J. Barciszewski. 5S Ribosomal RNA Database. *Nucleic Acids Research*, 30(1):176–178, 2002.

[13] J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22(22):4673–4680, 1994.

[14] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computional Biology*, 1(4):337–348, 1994.

To appear in a special issue of *Networks*

A novel approach to phylogenetic trees: d-dimensional geometric Steiner trees*

Marcus Brazil† Benny K. Nielsen‡ Doreen A. Thomas† Pawel Winter‡
Christian Wulff-Nilsen‡ Martin Zachariasen‡

Abstract

We suggest a novel distance-based method for the determination of phylogenetic trees. It is based on multidimensional scaling and Euclidean Steiner trees in high-dimensional spaces. Preliminary computational experience shows that the use of Euclidean Steiner trees for finding phylogenetic trees is a viable approach. Experiments also indicate that the new method is comparable with results produced by Neighbor Joining [20].

Keywords: Phylogeny, Steiner tree, multidimensional scaling

1 Introduction

One of the most important problems of modern biology is to explain how species have evolved over time and how they are related to each other. Such relationships are represented by *phylogenetic trees* where the *leaf nodes* represent the species while the *interior nodes* represent their ancestors. Usually, vertices of a phylogenetic tree represent sequences that identify the species, such as DNA or protein sequences, while edges represent changes (such as mutations, deletions and insertions) that have occurred to obtain one sequence from another. The major problem that faces biologists when constructing such phylogenetic trees is that the information about ancestors is very limited or is in fact non-existent. So the problem is to infer the interior nodes and their relationships with each other and with the studied species while asserting some particular model of evolution.

Several methods for the determination of phylogenetic trees have been considered over the years (see for example some of the monographs addressing mathematical and computational aspects of phylogenetic trees that appeared in recent years: Felsenstein [8]; Semple and Steel [22]). From a computational point of view, these methods are intractable as they lead to $\mathcal{NP}$-hard optimization problems. In order to obtain reasonable solutions even for relatively small problem instances, one therefore has to turn to heuristics where solutions can be obtained efficiently. These solutions may not be optimal with respect to the asserted model of evolution. Figure 1 shows some very small examples of phylogenetic trees. Figure 1a is an

*Partially supported by a grant from the Australia Research Council and by a grant from the Danish Natural Science Research Council (51-00-0336).

†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, d.thomas}@unimelb.edu.au.

‡Department of Computer Science, University of Copenhagen, DK-2100 Copenhagen Ø, Denmark. E-mail: {benny, pawel, koolooz, martinz}@diku.dk.

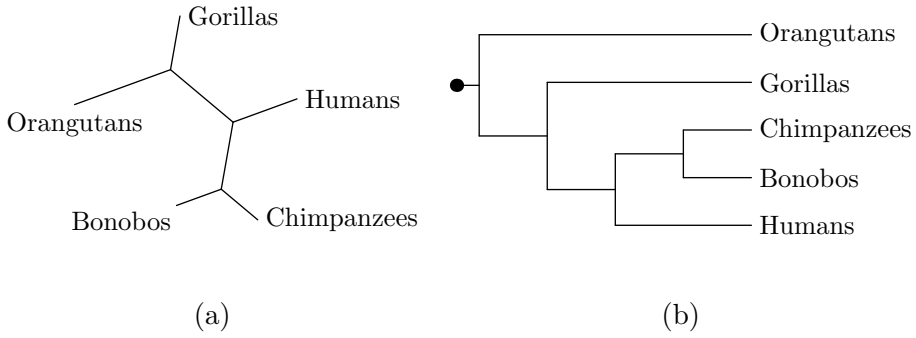


Figure 1: (a) An unrooted phylogenetic tree of a group of *great apes*. (b) A rooted phylogenetic tree of the same group. The large dot represents a hypothetical common ancestor for all of the species in the group.

example of an unrooted phylogenetic tree while Figure 1b shows a rooted phylogenetic tree. In some cases, edge lengths are used to indicate the time spans between "evolutionary events" (this is not the case for the examples in Figure 1). This example is taken from the Tree of Life website: `http://www.tolweb.org/` where many other, much bigger, phylogenetic trees can be found.

Computational approaches to phylogenetic trees can be divided into at least four categories: distance-based methods, maximum parsimony methods, maximum likelihood methods and tree merging methods. *Distance-based methods*, which are the focus of this paper, calculate the distances between all pairs of sequences using appropriately defined pairwise alignment, and then determine a tree based on these distances. Several distance-based methods have been suggested in the literature: *Unweighted Pair Group Method using Arithmetic mean* (UPGMA) suggested by Sneath and Sokal [24], *Neighbor Joining* (NJ) suggested by Saitou and Nei [20], *BioNJ* suggested by Gascuel [10], and *FastME* suggested by Desper and Gascuel [7]. The idea underlying all these methods is to merge pairs of nodes (beginning with the nodes representing the species), recompute distances from each merged node to the remaining nodes and repeat. UPGMA is very fast but implicitly assumes a molecular clock, i.e., it assumes uniform mutation rates along the edges. NJ generates unrooted trees and assumes additive distances in the tree. BioNJ and FastME are both based on the NJ method. BioNJ tries to improve the merging choices of NJ and FastME creates a quick initial phylogeny and then tries to improve it by doing nearest neighbor interchanges.

The purpose of this paper is to suggest a novel heuristic method to determine phylogenetic trees. It differs from other methods by exploiting Euclidean geometry to efficiently construct good phylogenetic trees. The use of Euclidean geometry for phylogeny reconstruction was originally proposed by Cavalli-Sforza and Edwards [3]. More recently, Kitazoe et al [15] presented an alternative geometry-based approach. The current prototype implementation of our approach is a proof of concept, but the long term goal is to provide a fast heuristic for generating good phylogenies for very large problem instances. While our method is clearly distance-based and similar to methods based on minimum evolution [19], it takes a more global view of the distances between species by generating a low-cost Steiner tree for appropriately located species in a d-dimensional space. This tree is an approximate solution to the *d-dimensional Euclidean Steiner tree problem* to determine the shortest tree interconnecting n given points in a d-dimensional space. Additional junctions, called *Steiner points* can be used

in such a tree.

Our approach for determining phylogenetic trees is quite straightforward. We use pairwise alignment to compute costs of optimal pairwise alignments between the species. Once an $n \times n$ symmetric dissimilarity matrix $\mathbf{\Delta}$ of all involved species has been determined, we use *multidimensional scaling* to represent the species as points in a higher dimensional Euclidean space such that their Euclidean distances match the dissimilarity matrix $\mathbf{\Delta}$ as closely as possible. We apply only classical multidimensional scaling although more advanced improvements do exist. The running time of this approach is $O(n^2)$. Once the points in d-dimensional space have been determined, for a given d, we find a heuristic solution to the d-dimensional Euclidean Steiner tree problem, i.e., we approximate a shortest tree spanning the points in d-dimensional space. The Steiner tree problem is an $\mathcal{NP}$-hard optimization problem, hence our main interest is not in the optimal solution but in a Steiner tree obtained by a straightforward heuristic (described in Section 3) which is a generalization of a well-known heuristic for the case $d = 2$. The exact locations of the Steiner points are not so important in this context. In fact, we are only interested in the topology of a low-cost Steiner tree. We do not provide a running time bound for the Steiner tree heuristic described in this paper, but it is not unlikely that a sufficiently efficient heuristic could be devised with a running time somewhere between $O(n^2)$ and $O(n^3)$. This emphasizes the long term goal of our approach to be able to handle large problem instances.

The paper is organized as follows. Multidimensional scaling is discussed in Section 2. Our methods to obtain good quality Steiner trees in d-dimensional Euclidean space are discussed in Section 3. Our method of determining phylogenetic trees is validated in several ways. This is discussed in Section 4. Section 5 presents our computational results. Concluding remarks and suggestions for further research are discussed in Section 6.

2 Multidimensional Scaling

Multidimensional scaling (MDS) is a method of representing a given collection of dissimilarities between pairs of objects as distances between points in a multidimensional metric space. The most well-known application of MDS is as a method of producing approximate visual representations of dissimilarities between objects. This is done by representing the objects as points in 2 or 3 dimensional Euclidean space such that the distances between points in the space match the original dissimilarities as closely as possible.

Our use of MDS in this paper is somewhat different. Here the reason for representing the objects as points in Euclidean space is to exploit geometric properties of the space in order to construct an approximate solution to the Steiner tree problem. Our heuristic methods for solving the Steiner tree problem are efficient even in relatively high dimensions, meaning there is no need to restrict our attention to low-dimensional space. The effectiveness of this approach, however, is dependent on being able to match the dissimilarity matrix for the given species and the distance matrix for the embedded points as closely as possible.

In this section we review classical MDS methods [4] for positive semi-definite dissimilarity matrices and then discuss how they can be adapted to more general dissimilarity matrices.

2.1 Classical Multidimensional Scaling

The techniques of classical multidimensional scaling are based around the assumption that the dissimilarities are actually distances in some higher dimensional Euclidean space. Hence

the aim is to reconstruct an embedding in a suitable space that preserves the set of distances. The first practical method for doing this was developed by Torgerson [28]. The steps in the classical method are as follows.

1. Suppose there are n objects in our set. Let $\mathbf{\Delta}$ be the $n \times n$ symmetric matrix of dissimilarities. Square each term in this matrix to compute $\mathbf{\Delta}^{(2)}$, the matrix of squared dissimilarities.

2. Let $\mathbf{H} = \mathbf{I}_n - n^{-1}\mathbf{1}\mathbf{1}^T$ be the *centering matrix* (where $\mathbf{1}$ denotes a vector of n ones). Apply the operation of *double centering* to $\mathbf{\Delta}^{(2)}$ to obtain the inner product matrix $\mathbf{B} = -\frac{1}{2}\mathbf{H}\mathbf{\Delta}^{(2)}\mathbf{H}$.

3. Compute the spectral decomposition of $\mathbf{B}$: $\mathbf{B} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$ where $\mathbf{\Lambda}$ is the diagonal matrix of eigenvalues $\{\lambda_i\}$ and $\mathbf{V}$ is the corresponding matrix of normalized eigenvectors $\{v_i\}$. Under the assumption that $\mathbf{\Delta}$ is actually a set of Euclidean distances, it can be shown that $\mathbf{B}$ is positive semi-definite, or equivalently that the eigenvalues in $\mathbf{\Lambda}$ are non-negative real numbers. Hence we can assume that the spectral decomposition satisfies $\lambda_1 \geqslant \lambda_2 \geqslant \ldots \geqslant \lambda_n \geqslant 0$.

4. Let d ($<n$) denote the dimensionality of the solution, which is the same as the number of non-zero eigenvalues in $\mathbf{\Lambda}$. Let $\mathbf{\Lambda}_d$ denote the diagonal matrix of the first d eigenvalues, and let $\mathbf{V}_d$ denote the first d columns of $\mathbf{V}$. Then the coordinate matrix of classical scaling for the n objects in Euclidean d-space is $\mathbf{X} = \mathbf{V}_d\mathbf{\Lambda}_d^{1/2}$.

If $\mathbf{\Delta}$ is a Euclidean distance matrix, then $\mathbf{X}$ recovers the original coordinates up to reflection, rotation and translation. To obtain good embeddings of the objects in smaller dimensions we reduce the size of d in the classical method, effectively discarding the smallest eigenvalues and their corresponding eigenvectors. This minimizes the loss function $\|\mathbf{X}\mathbf{X}^T - \mathbf{B}\|$ for the given dimension d.

For the cases we are interested in $\mathbf{\Delta}$ is a metric matrix (but not a Euclidean distance matrix) based on appropriately defined distances between sequences. Here again the classical method can be applied, but now the spectral decomposition for $\mathbf{B}$ may include negative eigenvalues, as $\mathbf{B}$ may not be positive semi-definite. The coordinate matrix $\mathbf{X}$ can be computed using the method above, where the Euclidean dimension d is chosen so that the discarded eigenvalues include all $\lambda_i \leqslant 0$. If the negative eigenvalues are small in magnitude then $\mathbf{X}$ represents an embedding whose distances are close to the original dissimilarities. If the negative eigenvalues are large, then $\mathbf{X}$ is less accurate.

3 Euclidean Steiner Trees in Higher Dimensions

The d-dimensional Euclidean Steiner tree problem is as follows. We are given a set of n points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, also denoted *terminals*, in d-dimensional space. (In our application, the coordinates of point $\mathbf{x}_i$ are given by the i'th row of the matrix $\mathbf{X}$, as computed by the multidimensional scaling.) The problem is to compute a tree of minimum Euclidean length that interconnects the terminals; such a tree is called a *Steiner minimum tree* (SMT). The SMT may contain junctions, so-called *Steiner points*, that are not among the given terminals.

A number of nice properties are known about SMTs in higher dimensions [14]. There are at most $n - 2$ Steiner points, and each of these has exactly three neighbours in the SMT.

Moreover, for a Steiner point $\mathbf{s}$ with neighbours $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{w}$ (which can be terminals or Steiner points), the edges $\{\mathbf{su}, \mathbf{sv}, \mathbf{sw}\}$ must be co-planar, and make 120° angles at the Steiner point $\mathbf{s}$. Edges meeting at terminals must make angles that are at least 120°.

A Euclidean minimum spanning tree (MST) for the terminals is a tree with minimum Euclidean length where only direct connections between the terminals are allowed. The MST problem is — in contrast to the Steiner tree problem — solvable in polynomial time. However, we are interested in Steiner trees rather than in MSTs for two reasons. The first is that Steiner trees are usually substantially shorter than spanning trees. The second, and even more important, reason is that the topology of a Steiner tree with its Steiner points yields a natural candidate for a phylogenetic tree.

For this paper we have designed a simple Steiner tree heuristic to evaluate our general approach to computing phylogenetic trees. In a forthcoming paper we will present a more sophisticated and general heuristic for approximating Steiner trees in higher dimensions for arbitrary metrics [2].

For the first phase of the heuristic we construct an MST T for the set of terminals $\mathbf{x}_1, \ldots, \mathbf{x}_n$. We then perform local improvements as originally suggested by Cavalli-Sforza and Edwards [3]: Let $\mathbf{uv}$ and $\mathbf{uw}$ be two edges meeting at a node $\mathbf{u}$, where the meeting angle is minimum over all such pairs of edges in T. If the meeting angle is less than 120°, the edges $\mathbf{uv}$ and $\mathbf{uw}$ are deleted, and the three vertices $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{w}$ are reconnected by the SMT for $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{w}$; this tree will be a star with a Steiner point $\mathbf{s}$ as its center. The optimal location of this Steiner point (with respect to the tree neighboring vertices) can be exactly computed in constant time. This operation is called a *Steiner point insertion*, and it will decrease the length of the tree. Steiner point insertions are iteratively performed until the decrease in tree length becomes marginal. As a final step in this phase, we apply the iterative approach by Smith [23] which improves the positioning of the Steiner points.

In the second phase of the heuristic we perform *shortcutting*. We iteratively pick a pair of nonadjacent vertices $\mathbf{u}$ and $\mathbf{v}$ in T. Consider the longest edge $\mathbf{u'v'}$ on the path between $\mathbf{u}$ and $\mathbf{v}$ in T. Let T' be the tree obtained from T by deleting the edge $\mathbf{u'v'}$ and adding the edge $\mathbf{uv}$ followed by local improvements. If the new tree T' is shorter than T, then we set $T = T'$ and iterate. The algorithm stops when no shortcutting improves the length of the tree within some small threshold value.

In order to use a Steiner tree as a phylogenetic tree, a postprocessing step is needed which ensures that all terminals are of degree 1 and all Steiner points are of degree 3. A terminal of degree higher than 1 can be replaced by a new Steiner point with the same coordinates and then the terminal can be connected to this Steiner point. A Steiner point of degree higher than 3 can be split into more Steiner points, but an exponentially growing number of different topologies are possible when applying such splitting. In practice this postprocessing step is rarely needed in high-dimensional Euclidean spaces.

4 Validation Methods

The best method to validate any phylogenetic tree method would of course be that it could reconstruct known phylogenies. Unfortunately known phylogenies are rare. One such "known" phylogeny was determined by Hillis *et al.* [11] for the cultures of bacteriophage T7. Unfortunately, the tree obtained was well-balanced and its nodes representing the phages had many restriction site differences. Reconstruction of such a tree by any reasonable phylogenetic

method is fairly easy.

The second best choice of "known" trees is computational simulation. A computer program is provided with a tree and sequences evolve along the edges of this tree according to an accurate probabilistic model of evolution. The resulting leaf sequences can then be used as input to some phylogenetic tree building method. A good method should generate the tree that was used in the simulation.

In our experiments we use the program Rose [26] to generate artificial protein sequences. Rose is also used to generate a random tree and a random root sequence. Multiple substitutions, insertions and deletions are allowed. Since Rose can also supply us with the correct multiple alignment there is not going to be any bias (in the alignment) to any particular tree reconstruction method [16]. In other words we do not need to use any form of pairwise or multiple alignment to be able to calculate distances. To keep everything as simple as possible we base the distances on simple percent differences (Hamming distance), i.e., no so-called distance corrections are made. Default parameters in Rose are used when nothing else is specified for our experiments, and, in general, root sequences are of length 500. One of the most important parameters is the *relatedness* of a set of sequences. The value of this parameter expresses the average relatedness of the sequences measured in *point accepted mutation* (PAM). This is a measure of the number of point mutations or changes per 100 amino acids. The default parameters of Rose include a mutation matrix, based on the work of Dayhoff *et al.* [6], which is used to determine the relative likelihood of each possible mutation.

Trees are compared using the *partition metric* which was originally proposed by Bourque [1]. It is also known as the Robinson-Foulds metric [18], the splits metric [22], and the symmetric difference metric. In short, the partition metric measures how many bi-partitions (induced by removing single edges) are in one tree and not in the other tree, that is, a distance of 0 reflects identical tree topologies. The main properties of the metric are that distances can be found in linear time [5] and that it is sensitive when applied to very similar trees [25]. Given a pair of binary trees with n leaves their maximum distance is $2n - 6$, but in the experiments we are using a normalized version of this metric such that the maximum distance is 1. For pairs of random binary trees the mean value of their distances approach the maximum distance as $n \to \infty$ [25].

In addition to our new solution method we also use the classic Neighbor Joining (NJ) algorithm [20] in our experiments. According to Hollich *et al.* [13] it does very well compared to some of the currently best known tree building methods such as BioNJ [10] and FastME [7].

The first description of the NJ algorithm does not include a running time analysis, but Studier and Keppler [27] described an algorithm with identical behavior and showed that it had a running time of $O(n^3)$ for n species. Their description of the NJ algorithm is as follows. Given a distance (or dissimilarity) matrix for n species with entries d_{ij} for each pair i and j, compute the values

$$S_{ij} = (n-2)d_{ij} - \sum_k d_{ik} - \sum_k d_{jk}.$$

A pair (x, y) with minimum S_{xy} is selected and a new "species" z is introduced to replace them. The columns/rows of x and y are deleted and a new row and a new column with distances from z to all remaining species are added. The distance from z to one of the remaining species u is based on the distances from x and y, such that $d_{zu} = (d_{xu} + d_{yu} - d_{xy})/2$. This procedure is repeated until only 3 species are left in the matrix. An unrooted tree then follows from the sequence of merging choices. Essentially, the NJ algorithm iteratively selects two species

which are close to each other and far away from the remaining species. The distance from a new species to a species u is simply based on an average of the distances from the replaced species to u.

We use a partition distance implementation and an NJ implementation from PHYLIP [9] in our experiments.[1] The d-dimensional Steiner minimum tree heuristic (dSMT) was implemented in C++.

5 Computational Experiments

In the experiments we are going to examine three basic questions. First, described in Section 5.1, we examine how well MDS retains the information needed to be able to obtain good phylogenies. Second, in Section 5.2, we examine whether short Euclidean Steiner trees are equivalent with good phylogenies. And finally, in Section 5.3, we examine how good phylogenies based on short Euclidean Steiner trees are compared to existing methods.

5.1 Evaluation of MDS

NJ can be applied to the original distance matrix, but it can also be applied to the distance matrix based on the Euclidean points found by MDS. Consider the following experiment. Using various problem sizes (number of sequences) and average relatedness values of the sequences in these problems, compare (using the partition metric) the trees found using NJ on the original distance matrix and the MDS matrices using an increasing number of dimensions. Results for an average relatedness value of 20 and 250 PAM for instances of size 20 and 50 and sequence lengths 10, 20, 100, and 500 are given in Figure 2.

The experiment shows that increasing the number of dimensions yields better results. The results of applying NJ to the original distance matrix are not shown, but they are all matched when permitting sufficiently high dimensions (the original trees are almost always found for sequence lengths of 500). Applying NJ to the points embedded in very few dimensions is clearly too crude an approximation. On the other hand, the number of dimensions required to obtain good approximations is bounded, e.g., only about 20 dimensions are needed for the instances of size 50 and length 500. Also note that the number of dimensions required seems to depend more on the number of sequences and sequence lengths than on the degree of average relatedness of the sequences. Observe that instances with short sequence lengths are harder to solve. This is due to the fact that NJ has less information available when reconstructing the trees. Most importantly, the results indicate that the information needed to obtain good phylogenies is preserved when using MDS.

5.2 Euclidean Steiner trees as phylogenies

Our method searches for a good Steiner tree in a greedy fashion starting with a minimum spanning tree. During the search a range of different topologies is found and we can use these to examine the relationship between Steiner tree length and phylogenetic tree quality. The graph in Figure 3a presents the results of such an experiment. The Euclidean lengths have been normalized, as a percentage of the MST length, and collected in intervals (by rounding to the nearest integer) such that they can be averaged. The graph clearly indicates that

[1] The code for MDS was found at `http://odur.let.rug.nl/kleiweg/L04/Manuals/mds.html`.

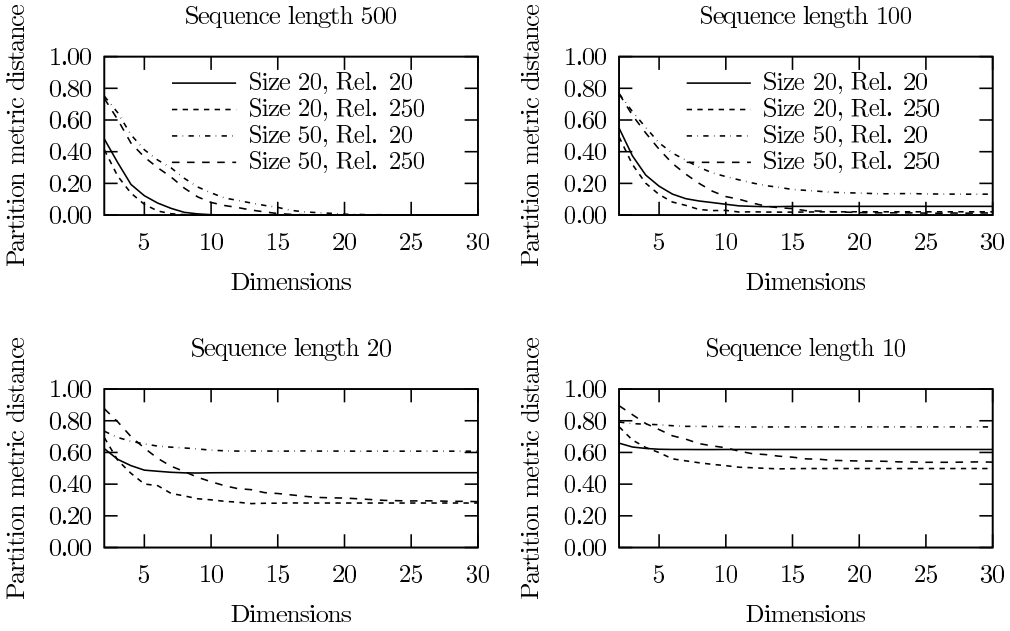


Figure 2: Results of applying Neighbor Joining to the distance matrices obtained by MDS with an increasing number of dimensions. The partition metric distances between NJ trees and the trees supplied by Rose for four different sequence lengths are shown. Values are averaged over 100 runs.

shorter trees are also better phylogenies. The irregular behavior for longer trees is caused by the fact that the heuristic generates very few trees with length close to the length of the MST, i.e., only a few trees contribute to these average values. Conversely, the behavior for shorter trees is based on large numbers of trees (> 100).

A similar experiment can be used to analyze the importance of the number of dimensions included. Results for using various numbers of dimensions on 100 instances of size 100 are shown in Figure 3b. When using a very small number of dimensions, SMTs show no improvement when compared to MSTs with respect to the phylogenies found. This clearly changes when using an increasing number of dimensions. The quality of the phylogenies found is very similar when using 40 dimensions or more although the SMTs get relatively shorter than MSTs with an increasing number of dimensions.

To determine whether the topologies given by Rose induce short Euclidean Steiner trees, we have compared the relative maximum and average difference in Euclidean length of trees obtained by dSMT and trees obtained by dSMT when fixing the topology to that given by Rose. Figure 4 shows the result of such an experiment. The graphs clearly suggest that for larger dimensions, the Rose phylogenetic trees are to be found among short Euclidean Steiner trees.

5.3 Comparison with existing methods

A direct comparison between the phylogenies generated by NJ and the ones generated by our Steiner tree approach is given in Figure 5.

Using Rose, 100 times 50 sequences are generated with relatedness values of 20 and 250

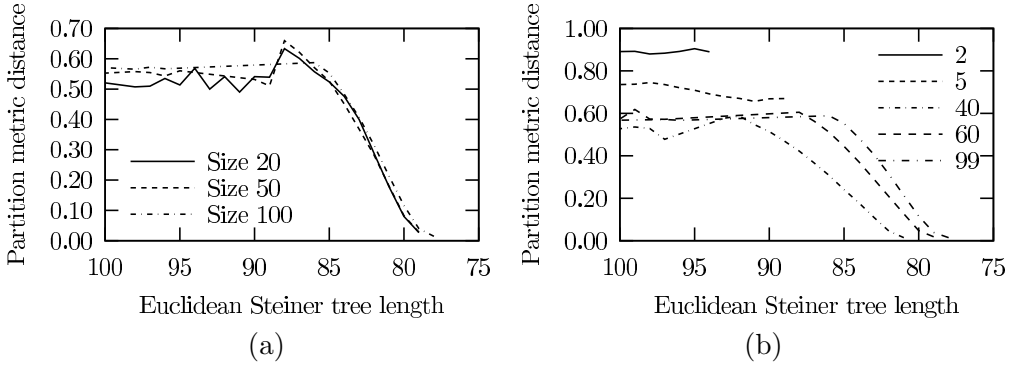


Figure 3: Relationship between Euclidean Steiner tree lengths and partition metric distances to the tree generated by Rose. (a) All trees were found in 100 runs using an average relatedness of 100 PAM. The maximum number of dimensions is used for each instance (equal to the number of positive eigenvalues). (b) A fixed problem size of 100 is used while the choice of the upper dimension bound is varied.

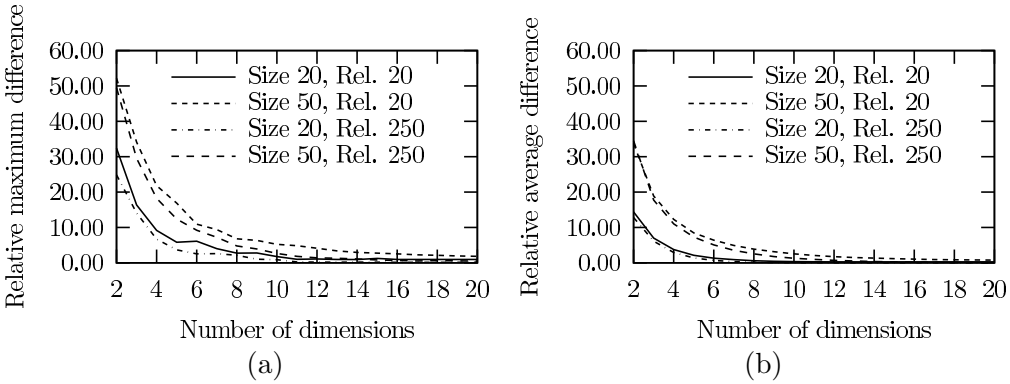


Figure 4: The difference in Euclidean length between Steiner trees obtained by dSMT and Steiner trees obtained by dSMT with the topology from Rose fixed, for various dimensions, problem sizes and relatedness values. All sequences have length 100. (a) The relative maximum length difference (in percent) over 100 instances. (b) The relative average length difference over 100 instances.

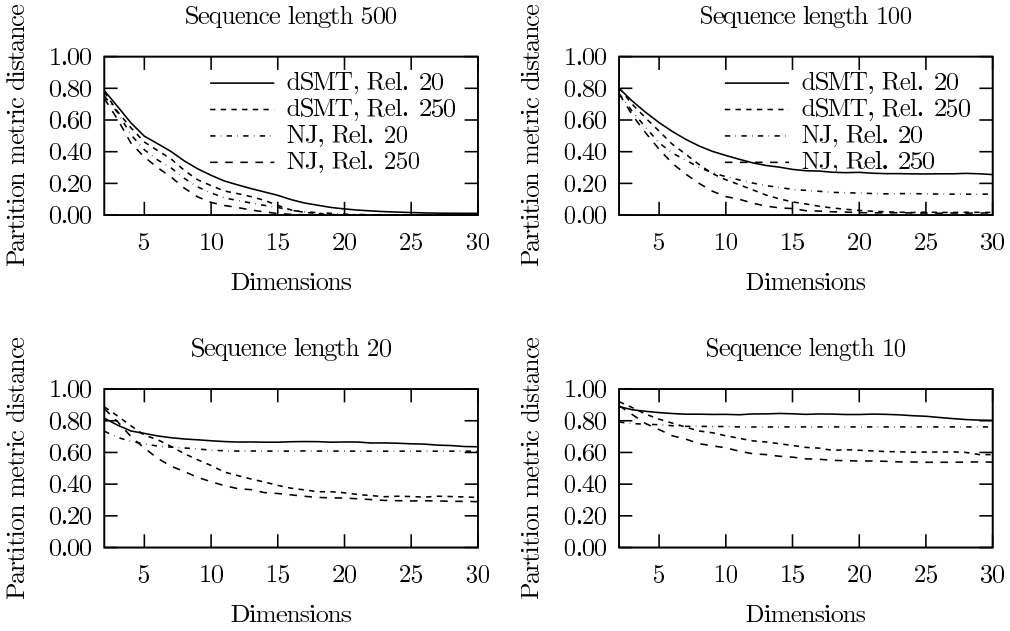


Figure 5: Average results of applying NJ and dSMT to 100 generated data instances with 50 sequences each. The primary axis is the number of dimensions used when using MDS and the secondary axis is the distance to the correct tree. Results for two different average relatedness values and four different sequence lengths are shown.

and sequence lengths 10, 20, 100, and 500. These are then used as input to both methods and the distances to the correct trees are averaged and presented for each number of possible dimensions. For sequence length 500, the results are quite similar since both methods eventually find all correct trees (note that dimensions 31-49 are not included). The Steiner approach seems to need a couple of extra dimensions compared to NJ to find solutions of the same quality. For shorter sequence lengths, NJ seems to find slightly better solutions than dSMT. Results of running NJ on the original distance matrices are not included in Figure 5 but the partition metric distances for this experiment are only marginally better than those obtained by NJ for dimension 30.

We have compared dSMT and NJ on an instance consisting of 143 sequences. The instance has an associated candidate tree [12] describing their relationships[2]. Applying NJ to this instance gives a solution with partition metric distance 0.1 to the given tree.

The result of applying dSMT to the same instance for various dimensions is shown in Figure 6. Observe that in smaller dimensions, the initial solutions found by dSMT are of better quality than in higher dimensions but that the final solutions are not. The best solutions are found in dimensions 80 and 142 with a distance of about 0.171 to the given tree. Hence, dSMT is unable to find solutions of equal or better quality than NJ. This can be explained by the fact that the given tree was generated using the Clustal-W package, which makes use of NJ.

[2]The tree and a multiple alignment of the sequences can be found at `http://www.mrc-lmb.cam.ac.uk/myosin/trees/trees.html`.

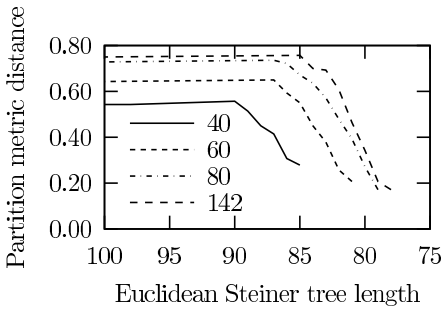


Figure 6: For an instance consisting of 143 sequences, the above graph shows the relationship between Euclidean Steiner tree length and partition metric distance to the given tree for the intermediate trees obtained by dSMT during the iterative search. Euclidean lengths are collected in intervals by rounding to the nearest integer and the minimum distance in each interval is shown. Dimensions considered are 40, 60, 80, and 142.

6 Concluding Remarks and Future Research

We have presented a new method for finding phylogenetic trees using a geometric Steiner tree approach. It is not an easy task to experimentally analyze the quality of such an approach, but the results are promising and we have verified that there is a strong correspondence between short Euclidean Steiner trees and good phylogenies.

The results could be affected, for better or for worse, by a range of possible variations of both the experiments and the solution methods. The experiments could, e.g., be affected by the use of distance correction or simply by other choices of relatedness, problem sizes or other Rose parameters. The major weakness of using Rose is that it is only a model of the evolutionary process. Some phylogenetic methods might be more or less tuned to handle this model and thus one should be careful when comparing methods using only Rose generated data. Rose and the partition metric distance could be replaced by the use of real data instances and some other means of comparison such as the tree alignment score. In fact, in preliminary experiments using a variation of the *Deferred Path Heuristic* [17, 21] to heuristically assess the tree alignment score, MDS/dSMT performed marginally better than NJ.

Note that the current implementation of our approach is an initial prototype. We have already shown that the trees constructed are comparable with NJ trees, but there is still considerable potential for improvement, both regarding quality of trees and speed.

Classical MDS can be improved with various techniques not discussed in this paper and these improvements could in turn improve the results of dSMT. The heuristic method for finding short Euclidean Steiner trees could also be improved and whenever such improvements in length also involve topology changes, it could potentially result in better phylogenies. Improvements regarding the running time would enable the heuristic to handle larger phylogenies. No running time bound is given for the current implementation, but based on experience with geometric Steiner problems in two dimensions, it is likely possible to create an efficient heuristic with a running time below $O(n^3)$ and maybe even close to $O(n^2)$, i.e., asymptotically faster than NJ.

When using a heuristic to obtain a phylogeny, one is often interested in phylogenies which

are similar in quality. In Euclidean space it is easy to define a neighborhood of such phylogenies by simply looking at those which are near by in the sense of Steiner tree length. Such phylogenies could differ a lot topologically and this could be a very interesting approach for generating a set of phylogenies or a phylogenetic network.

Although our experiments show that minimizing the Euclidean tree length gives phylogenetic trees of relatively high quality, the power of the simple NJ algorithm suggests that other cost functions may be more suitable. Experimenting with some hybrid between the NJ algorithm and our algorithm could be a fruitful area of future research.

References

[1] M. Bourque. *Arbres de Steiner et réseaux dont varie l'emplagement de certains sommets*. PhD thesis, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, 1978.

[2] M. Brazil, B. K. Nielsen, D. A. Thomas, P. Winter, and M. Zachariasen. A fast heuristic for computing Steiner trees in higher dimensions. In Preparation.

[3] L. L. Cavalli-Sforza and A. W. F. Edwards. Phylogenetic analysis: Models and estimation procedures. *Evolution*, 21:550–570, 1967.

[4] T. F. Cox and M. Cox. *Multidimensional scaling - 2nd ed.* CRC Press LLC, Florida, 2001.

[5] W. H. E. Day. Optimal algorithms for comparing trees with labeled leaves. *Journal of Classification*, 2(1):7–28, 1985.

[6] M. O. Dayhoff, R. M. Schwarz, and B. C. Orcutt. A model of evolutionary change in proteins. In M. O. Dayhoff, editor, *Atlas of Protein Sequence and Structure*, volume 5, pages 345–352. National Biomedical Research Foundation, Washington, DC, 1978.

[7] R. Desper and O. Gascuel. Fast and accurate phylogeny reconstruction algorithms based on the minimum-evolution principle. *Journal of Computional Biology*, 9(5):687–705, 2002.

[8] J. Felsenstein. *Inferring phylogenies*. Sinauer Associates, Inc., Sunderland, MA, 2003.

[9] J. Felsenstein. PHYLIP (phylogeny inference package) version 3.5c. Technical report, Department of Genetics, University of Washington, Seattle, 1993. URL `http://evolution.genetics.washington.edu/phylip.html`.

[10] O. Gascuel. BIONJ: An improved version of the NJ algorithm based on a simple model of sequence data. *Molecular Biology and Evolution*, 14(7):685–695, 1997.

[11] D. Hillis, J. J. Bull, M. E. White, M. R. Badgett, and I. J. Molineux. Experimental phylogenetics: Generation of a known phylogeny. *Science*, 255:589–592, 1992.

[12] T. Hodge and M. J. Cope. A myosin family tree. *Journal of Cell Science*, 113(19):3353–3354, 2000.

[13] V. Hollich, L. Milchert, L. Arvestad, and E. L. L. Sonnhammer. Assessment of protein distance measures and tree-building methods for phylogenetic tree reconstruction. *Molecular Biology and Evolution*, 22(11):2257–2264, 2005.

[14] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner tree problem*. Annals of Discrete Mathematics 53. Elsevier Science Publishers, Netherlands, 1992.

[15] Y. Kitazoe, H. Kishino, T. Okabayashi, T. Watabe, N. Nakajima, Y. Okuhara, and Y. Kurihara. Multidimensional vector space representation for convergent evolution and molecular phylogeny. *Molecular Biology and Evolution*, 22(3):704–715, 2005.

[16] D. A. Morrison and J. T. Ellis. Effects of nucleotide sequence alignment on phylogeny estimation: A case study of 18S rDNAs of apicomplexa. *Molecular Biology and Evolution*, 14(4):428–441, 1997.

[17] B. K. Nielsen, S. Lindgreen, P. Winter, and M. Zachariasen. Deferred path heuristic for phylogenetic trees revisited. In M.-F. Sagot and K. S. Guimarães, editors, *CompBioNets 2005: Algorithmics and Computational Methods for Biochemical and Evolutionary Networks*, volume 5 of *Texts in Algorithmics*, pages 75–92, King's College London, Strand, London, 2005. College Publications.

[18] D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53(1):131–147, 1981.

[19] A. Rzhetsky and M. Nei. Theoretical foundation of the minimum-evolution method of phylogenetic inference. *Molecular Biology and Evolution*, 10(5):1073–1095, 1993.

[20] N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987.

[21] B. Schwikowski and M. Vingron. Weighted sequence graphs: Boosting iterated dynamic programming using locally suboptimal solutions. *Discrete Applied Mathematics*, 127(1):95–117, 2003. ISSN 0166-218X.

[22] C. Semple and M. Steel. *Phylogenetics*. Oxford University Press, New York, 2003.

[23] W. D. Smith. How to find Steiner minimal trees in Euclidean d-space. *Algorithmica*, 7(2/3): 137–177, 1992.

[24] P. H. A. Sneath and R. R. Sokal. *Numerical taxonomy: The principles and practice of numerical classification*. Freeman, San Francisco, 1973.

[25] M. A. Steel and D. Penny. Distributions of tree comparison metrics — some new results. *Systematic Biology*, 42(2):126–141, 1993.

[26] J. Stoye, D. Evers, and F. Meyer. Rose: Generating sequence families. *Bioinformatics*, 14(2): 157–163, 1998.

[27] J. A. Studier and K. J. Keppler. A note on the neighbor-joining algorithm of Saitou and Nei. *Molecular Biology and Evolution*, 5(6):729–731, 1988.

[28] W. S. Torgerson. Multidimensional scaling I: Theory and method. *Psychometrika*, 17:401–419, 1952.