Computational Geometry 42 (2009) 269–288

Contents lists available at ScienceDirect

Computational Geometry: Theory and Applications

www.elsevier.com/locate/comgeo

Translational packing of arbitrary polytopes

Jens Egeblad [a], Benny K. Nielsen [a,*], Marcus Brazil [b]

[a] Department of Computer Science, University of Copenhagen, DK-2100 Copenhagen Ø, Denmark
[b] ARC Special Research Center 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

ARTICLE INFO

Article history:
Received 29 October 2007
Received in revised form 7 May 2008
Accepted 16 June 2008
Available online 5 July 2008
Communicated by R. Fleischer

Keywords:
Packing
Heuristics
Translational packing
Packing polytopes
Minimizing overlap
Maximizing overlap
Strip-packing
Guided local search

ABSTRACT

We present an efficient solution method for packing d-dimensional polytopes within the bounds of a polytope container. The central geometric operation of the method is an exact one-dimensional translation of a given polytope to a position which minimizes its volume of overlap with all other polytopes. We give a detailed description and a proof of a simple algorithm for this operation in which one only needs to know the set of $(d-1)$-dimensional facets in each polytope. Handling non-convex polytopes or even interior holes is a natural part of this algorithm. The translation algorithm is used as part of a local search heuristic and a meta-heuristic technique, guided local search, is used to escape local minima. Additional details are given for the three-dimensional case and results are reported for the problem of packing polyhedra in a rectangular parallelepiped. Utilization of container space is improved by an average of more than 14 percentage points compared to previous methods.

The translation algorithm can also be used to solve the problem of maximizing the volume of intersection of two polytopes given a fixed translation direction. For two polytopes with complexity $O(n)$ and $O(m)$ and a fixed dimension, the running time is $O(nm\log(nm))$ for both the minimization and maximization variants of the translation algorithm.

1. Introduction

Three-dimensional packing problems have applications in various industries, e.g., when items must be loaded and transported in shipping containers. The three main problems are bin-packing, knapsack packing, and container loading. In bin-packing the minimum number of equally-sized containers sufficient to pack a set of items must be determined. In knapsack packing one is given a container with fixed dimensions and a set of items, each with a profit value; one must select a maximum profit subset of the items which may be packed within the container. The container loading problem is a special case of the knapsack problem where the profit value of each item is set to its volume. Bin-packing, knapsack packing, and container loading problems involving boxes are classified as orthogonal packing problems and are well-studied in the literature.

In general, three-dimensional packing problems can also involve more complicated shapes; it is not only boxes that are packed in shipping containers. An interesting example is *rapid prototyping* which is a term originally used for the production of physical prototypes of 3D computer aided design (CAD) models needed in the early design or test phases of new products. Nowadays, rapid prototyping technologies are also used for manufacturing purposes. One of these technologies, *selective laser sintering process*, is depicted in Fig. 1. The idea is to build up the object(s) by adding one very thin layer at a time. This is

* Corresponding author.
E-mail addresses: jegeblad@diku.dk (J. Egeblad), benny@diku.dk (B.K. Nielsen), brazil@ee.unimelb.edu.au (M. Brazil).

0925-7721/$ – see front matter
doi:10.1016/j.comgeo.2008.06.003

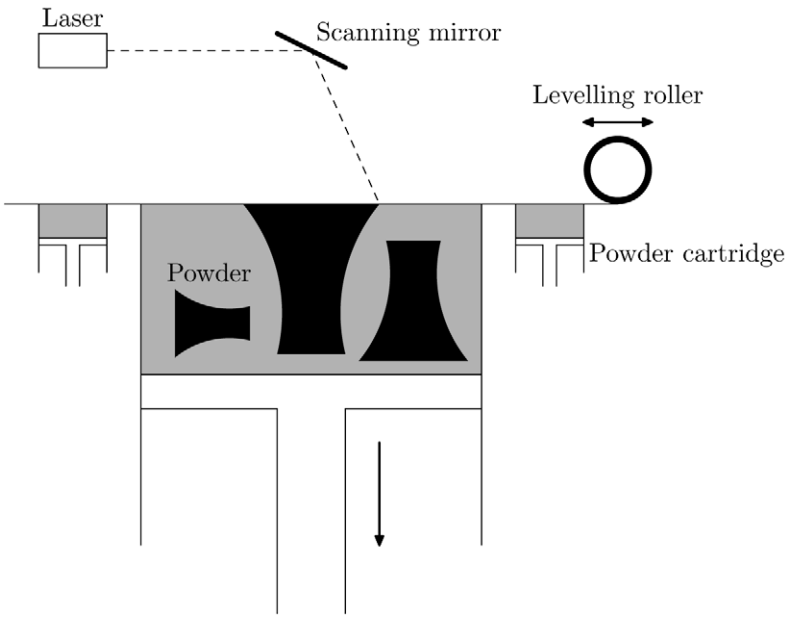

Fig. 1. An illustration of a typical machine for rapid prototyping. The powder is added one layer at a time and the laser is used to sinter what should be solidified to produce the desired objects.

done by rolling out a thin layer of powder and then sintering (heating) the areas/lines which should be solid by the use of a laser. The unsintered powder supports the objects built and therefore no pillars or bridges have to be made to account for gravitational effects. This procedure takes hours ("rapid" when related to weeks) and since the time required for the laser is significantly less than the time required for preparing a layer of powder, it will be an advantage to have as many objects as possible built in one run of the machine. A survey of rapid prototyping technologies is given by Yan and Gu [32].

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. [13]. 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/polyhedron 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 [17]. If $\nu(P)$ denotes the volume of a polyhedron P and this is generalized for sets such that $\nu(\mathcal{S}) = \sum_{P\in\mathcal{S}} \nu(P)$ then a non-overlapping placement for the 3DDPP has a *utilization* (of the container) of $\nu(\mathcal{S})/\nu(C)$. Based on the decision problem we can define the following optimization problem.

The 3D Strip Packing Problem (3DSPP). Given a set of polyhedra $\mathcal{S}$ and a rectangular parallelepiped C (the container) with fixed width w and length l, find the minimum height h of the container for which the answer to the 3D decision packing problem is positive.

An optimal solution to 3DSPP has a utilization of $\nu(\mathcal{S})/\nu(C) = \nu(\mathcal{S})/(w \cdot l \cdot h)$, i.e., the utilization only depends on the height of the parallelepiped and not on a particular placement corresponding to this height. The word *strip* is based on the terminology used for the 2-dimensional variant of the problem.

While the solution method discussed in the following sections could be applied to the bin-packing problem or other variants of multi-dimensional packing problems, we limit our description to the strip-packing problem. The strip-packing variant has been chosen mainly because it allows a comparison with results from the existing literature. In the typology of Wäscher et al. [31], the problem we consider, 3DSPP, is a three-dimensional irregular open dimension problem (ODP) with fixed orientations of the polyhedra.

The polyhedra handled in this paper are very general. Informally, a polyhedron can be described as a solid whose boundary consists of a finite number of polygonal faces. Note that every face must separate the exterior and the interior of the polyhedron, but convexity is not required and holes and interior voids are allowed. A polyhedron is even allowed to consist of several disconnected parts and holes may contain smaller individual parts.

The problem formulations above are easily generalized to higher dimensions. Simply replace polyhedron with polytope and consider a rectangular d-dimensional parallelepiped for the strip-packing problem. We denote the corresponding problems dDDPP and dDSPP, where d is the number of dimensions. For simplicity, the faces are required to be convex, but this is not a restriction on the types of polytopes allowed, as a non-convex face can be partitioned into a finite number of convex faces. The polytopes themselves can be non-convex and contain holes.

Since dDDPP is $\mathcal{NP}$-complete dDSPP is an $\mathcal{NP}$-hard problem. Our solution method for dDSPP is heuristic and therefore not guaranteed to find the optimal solution of dDSPP.

When solving a problem in 3D for physical applications such as rapid prototyping, one should be aware that some feasible solutions are not very useful in practice since objects may be interlocked. Avoiding this is a very difficult constraint which is not considered in this paper.

3. Related work

Cutting and packing problems have received a lot of attention in the literature, but focus has mainly been on one or two dimensions and also often restricted to simple shapes such as boxes. A survey of the extensive 2D packing literature is given by Sweeney and Paternoster [29] and a survey of the 2D packing literature concerning irregular shapes (nesting) is given by Dowsland and Dowsland [12]. Recent heuristic methods for orthogonal packing problems include the work of Lodi et al. [23] and Faroe et al. [16] for the bin-packing problem, and Bortfeldt et al. [2] and Eley [15] for the container loading problem. The meta-heuristic approach utilized in this paper is based on the ideas presented by Faroe et al.

In the following, we review solution methods presented for packing problems in more than two dimensions which also involve shapes more general than boxes. A survey is given by Cagan et al. [4] in the broader context of *three-dimensional layout problems* for which maximum utilization may not be the only objective. Their focus is mainly on various meta-heuristic approaches to the problems, but a section is also dedicated to approaches for determining intersections of shapes. A survey on *3D free form packing and cutting problems* is given by Osogami [27]. 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. [21] have developed one of the earliest approaches to a non-rectangular 3D packing problem. Using a genetic algorithm they can handle non-convex shapes with holes and a fixed number of orientations (45° increments on all three axes). To evaluate if two shapes overlap, their bounding boxes (the smallest axis-aligned circumscribing box) are first tested for intersection and, if they intersect, triangles are subsequently tested for intersection. For each pair of intersecting triangles it is calculated how much each edge of each triangle intersects the opposite triangle.

Cagan et al. [3] use the meta-heuristic *simulated annealing* and they allow rotation. They can also handle various additional optimization objectives such as routing lengths. Intersection checks are done using *octree decompositions* of shapes. As the annealing progresses the highest resolution is increased to improve accuracy. Improvements of this work using variants of the meta-heuristic *pattern search* instead of simulated annealing are later described by Yin and Cagan [33,34].

Dickinson and Knopf [10] focus on maximizing utilization, but they introduce an alternative metric to determine the compactness of a given placement of shapes. In short, this metric measures the compactness of the remaining free space. The best free space, in three dimensions, is in the form of a sphere. The metric is later used by Dickinson and Knopf [11] with a sequential placement algorithm for three-dimensional shapes. Items are placed one-by-one according to a predetermined sequence and each item is placed at the best position as determined by the free-space metric. To evaluate if two shapes overlap they use depth-maps. For each of the six sides of the bounding box of each shape, they divide the box-side

into a uniform two-dimensional grid and store the distance perpendicular to the box-side from each grid cell to the shape's surface. Determining if two shapes overlap now amounts to testing the distance at all overlapping grid points of bounding box sides when the sides are projected to two dimensions. For each shape up to 10 orientations around each of its rotational axes are allowed. Note that the free-space metric is also generalized for higher dimensions and thus the packing algorithm could potentially work in higher dimensions.

Hur et al. [20] use voxels, a three-dimensional uniform grid structure, to represent shapes. In both voxel and octree representations a 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. [20] 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. [14] 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. [28] 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. [28] use Φ-functions to model non-intersection requirements. A Φ-function for a pair of shapes is defined as a real-value calculated from their relative placement. If the shapes overlap, abut or do not overlap the value of the Φ-function is larger than, equal or less than 0, respectively. A tree-search is proposed to solve the problem to optimality, but due to the size of the solution space Stoyan et al. opt for a method that finds locally optimal solutions instead. Computational results are presented for three problem instances with up to 25 polyhedra. A comparison with the results of the solution method presented in this paper can be found in Section 6.

4. Axis-aligned translation

As mentioned in the introduction, our solution method for packing polytopes is based on an algorithm for translating a given polytope to a minimum volume of overlap position in an axis-aligned direction. This problem is polynomial-time solvable and we present an efficient algorithm for it here. The algorithm can easily be modified to determine a maximum overlap translation. Note that the position of a polytope is specified by the position of a given reference point on the polytope; hence its position corresponds to a single point.

Without loss of generality, we assume that the translation under consideration is an x-axis-aligned translation. In three dimensions, the problem we are solving can be stated as follows:

1-Dimensional Translation Problem in 3D (1D3DTP). Given a fixed polyhedral container C, a polyhedron Q with fixed position, and a polyhedron P with fixed position with respect to its y and z coordinates, find a horizontal offset x for P such that the volume of overlap between P and Q is minimized (and P is within the bounds of the container C).

By replacing the term polyhedron with polytope this definition can easily be generalized to higher dimensions, in which case we denote it 1DdDTP. In Section 4.1 we give a more formal definition and present a number of properties of polytopes which we use in Section 4.2 to prove the correctness of an algorithm for 1DdDTP. Since the algorithm solves the problem of finding a minimum overlap position of P we refer to it in the following as the translation algorithm. In Section 4.3 we provide additional details for the three dimensional case.

Egeblad et al. [13] have proved the correctness of the two-dimensional special case of the algorithm described in the following and they have also sketched how the ideas can be generalized to 3D. Here we flesh out the approach sketched by Egeblad et al., and generalize it to d dimensions.

4.1. Polytopes and their intersections

In the literature, the term *polytope* is usually synonymous with *convex polytope*, which can be thought of as the convex hull of a finite set of points in d-dimensional space, or, equivalently, a bounded intersection of a finite number of half-spaces. In this paper we use the term *polytope* to refer to a more general class of regions in d-dimensional space, which may be non-convex and can be formed from a finite union of convex polytopes.

By definition, we assume that the boundary of a polytope P is composed of *faces*, each of which is a convex polytope of dimension less than d. Following standard notation, we refer to a one-dimensional face of P as an *edge*, and a zero-dimensional face as a *vertex*. The faces must satisfy the following properties:

1. The $(d-1)$-dimensional faces of P (which we refer to as *facets*) have the property that two facets do not intersect in their interiors.

2. The facets must be simple, i.e., on the boundary of a facet each vertex must be adjacent to exactly $d-1$ edges. Note that this only affects polytopes of dimension 4 or more.
3. Each face of P of dimension $k < d-1$ lies on the boundary of at least two faces of P of dimension $k+1$ (and hence, by induction, on the boundary of at least two facets).

Note that our definition of a polytope allows two adjacent facets to lie in the same hyperplane. This allows our polytopes to represent very general shapes, while imposing the condition that all faces are convex (by partitioning any non-convex facets into convex $(d-1)$-dimensional polytopes). Most importantly, our definition of polytopes does not require boundaries to be triangulated in 3D. Note that such a requirement would only allow minor simplifications in the proofs and the algorithm described later in this paper.

Given a polytope P, we write $p \in P$ if and only if p is a point of P including the boundary. More importantly, we write $p \in \text{int}(P)$ if and only if p is an interior point of P, i.e., $\exists \varepsilon > 0\colon\ \forall p' \in \mathbb{R}^d \Rightarrow \|p - p'\| < \varepsilon,\ p' \in P$.

We next introduce some new definitions and concepts required to prove the correctness of the translation algorithm in d dimensions.

Let e_i be the ith coordinate system basis vector, e.g., $e_1 = (1, 0, \ldots, 0)^T$. As stated earlier we only consider translations in the direction of the x-axis (that is, with direction $\pm e_1$). This helps to simplify the definitions and theorems of this section without loss of generality since translations along other axes work in a similar fashion. In the remainder of this section it is convenient to refer to the direction $-e_1$ as *left* and e_1 as *right*.

Given a polytope P, we divide the points of the boundary of P into three groups, *positive*, *negative*, and *neutral*.

Definition 1 *(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 $\exists \varepsilon > 0\colon\ \forall \delta \in (0, \varepsilon)\colon\ p + \delta \cdot e_1 \in \text{int}(P)$ and $p - \delta \cdot e_1 \notin \text{int}(P)$,
- negative if $\exists \varepsilon > 0\colon\ \forall \delta \in (0, \varepsilon)\colon\ p - \delta \cdot e_1 \in \text{int}(P)$ and $p + \delta \cdot e_1 \notin \text{int}(P)$,
- and neutral if it is neither positive nor negative.

In other words, a point is positive if the interior of the polytope is only on the right side of the point, it is negative if the interior of the polytope is only on the left side of the point, and it is neutral if the interior of the polytope is on both the left and the right side of the point or on neither the left nor the right side.

Clearly, each point on the boundary is covered by one and only one of the above cases. Furthermore, all points in the interior of a given face have the same sign. Therefore, any set of facets F can be partitioned into three sets F^+, F^-, and F^0 consisting of respectively positive, negative, and neutral facets from F. Examples in two and three dimensions are given in Fig. 2.

In order to handle some special cases in the proofs, we need to be very specific as to which facet a given boundary point belongs. Every positive or negative point p on the boundary is *assigned* to exactly one facet as follows. If p belongs to the interior of a facet f then it cannot belong to the interior of any other facet and thus it is simply assigned to f. If p does not belong to the interior of a facet then it must be on the boundary of two or more facets. If p is positive, then it follows easily that this set of facets contains at least one positive facet to which it can be assigned. Analogously, if p is negative it is assigned to a negative facet. Such an assignment of the boundary will be referred to as a *balanced assignment*. Neutral points are not assigned to any facets. Given a (positive or negative) facet f, we write $p \in f$ if and only if p is a

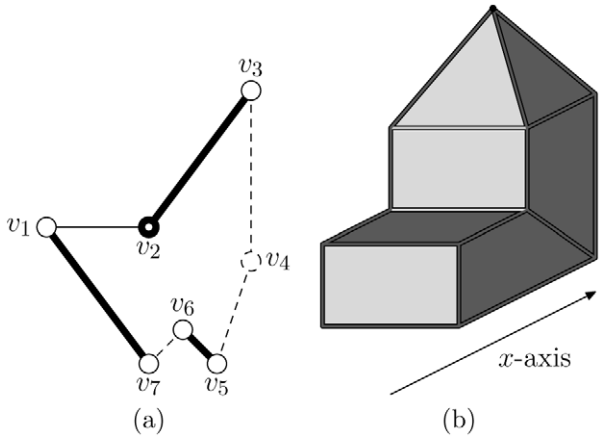

Fig. 2. (a) A polygon with three positive (thick) edges, (v_1, v_7), (v_2, v_3), and (v_5, v_6), three negative (dashed) edges, (v_3, v_4), (v_4, v_5), and (v_6, v_7), and one neutral (thin) edge, (v_1, v_2). Also note that the end-points v_1, v_3, v_5, v_6, and v_7 are neutral (thin), v_2 is positive (thick) and v_4 is negative (dashed). (b) A polyhedron for which only positive (bright) and neutral (dark) faces are visible. Most of the edges are neutral (dark) since the interior of the polyhedron is neither to the left nor the right of the edges. Two edges are positive (bright) since the interior is only to the right of them.

point assigned to f. Note that since all points in the interior of a face have the same sign, it follows that the assignment of all boundary points of the polytope can be done in bounded time; one only needs to determine the sign of one interior point of a face (of dimension 1 or more) to assign the whole interior of the face to a facet.

It follows from the definition of balanced assignment that a point moving in the direction of e_1, that passes through an assigned point of a facet of P, either moves from the exterior of P to the interior of P or vice versa. To determine when a point is inside a polytope we need the following definition.

Definition 2 *(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:

$$\overleftarrow{C}_{F^+}(p) = \left|\left\{f \in F^+ \mid \exists t > 0\colon\ p - te_1 \in f\right\}\right|,$$
$$\overleftarrow{C}_{F^-}(p) = \left|\left\{f \in F^- \mid \exists t \geqslant 0\colon\ p - te_1 \in f\right\}\right|,$$
$$\overrightarrow{C}_{F^+}(p) = \left|\left\{f \in F^+ \mid \exists t \geqslant 0\colon\ p + te_1 \in f\right\}\right|,$$
$$\overrightarrow{C}_{F^-}(p) = \left|\left\{f \in F^- \mid \exists t > 0\colon\ p + te_1 \in f\right\}\right|.$$

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

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

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

The proof is straightforward, and is omitted.

As a corollary, the facet count functions provide an easy way of determining whether or not a given point is in the interior of P.

Corollary 1. *Let P be a polytope with facet set F. Given a balanced assignment of the boundary points, then for every point $p \in \mathbb{R}^d$ we have that p lies in the interior of P if and only if $\overrightarrow{C}_{F^-}(p) - \overrightarrow{C}_{F^+}(p) = 1$. Similarly, p lies in the interior of P if and only if $\overleftarrow{C}_{F^+}(p) - \overleftarrow{C}_{F^-}(p) = 1$.*

Proof. Follows directly from Lemma 1. □

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 3 *(Containment function, inter-facet region).* Given two facets f and g and a point $p' \in \mathbb{R}^d$ define the *containment function*

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

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 Fig. 3.

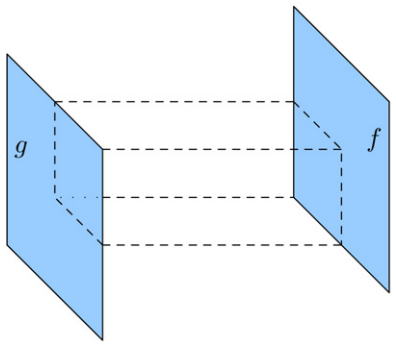

Fig. 3. A simple example of the inter-facet region $R(f,g)$ of two vertical faces f and g in $\mathbb{R}^3$. The dashed lines delimit the region. Note that the inter-facet region $R(g,f)$ is empty.

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*:

$$p \in \operatorname{int}(P \cap Q) \quad \Leftrightarrow \quad w(p) = 1,$$
$$p \notin \operatorname{int}(P \cap Q) \quad \Leftrightarrow \quad w(p) = 0,$$

where

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

Proof. $\mathbf{C}(F^+, G^-, p)$ is equal to $\overrightarrow{C}_{F^+}(p) \cdot \overleftarrow{C}_{G^-}(p)$, since it is equal to the number of facets from F^+ which are to the right of p, times the number of facets from G^- which are to the left of p. Similar observations allows us to deduce that $\mathbf{C}(F^-, G^+, p) = \overrightarrow{C}_{F^-}(p) \cdot \overleftarrow{C}_{G^+}(p)$, $\mathbf{C}(F^+, G^+, p) = \overrightarrow{C}_{F^+}(p) \cdot \overleftarrow{C}_{G^+}(p)$, and $\mathbf{C}(F^-, G^-, p) = \overrightarrow{C}_{F^-}(p) \cdot \overleftarrow{C}_{G^-}(p)$.

If we assume $p \in \operatorname{int}(P \cap Q)$ then from Corollary 1, we know that $\overrightarrow{C}_{F^-}(p) - \overrightarrow{C}_{F^+}(p) = 1$ and $\overleftarrow{C}_{G^+}(p) - \overleftarrow{C}_{G^-}(p) = 1$ and we get:

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

When $p \notin \operatorname{int}(P \cap Q)$ the three cases where $p \notin P$ and/or $p \notin Q$ can be evaluated in similar fashion. □

In order to solve 1D*d*DTP we need to define a measure of the overlap between two polytopes.

Definition 4 *(Overlap measures).* An *overlap measure* is a real-valued function μ such that, for any bounded region R_0, $\mu(R_0) = 0$ if $\operatorname{int}(R_0) = \emptyset$ and $\mu(R_0) > 0$ otherwise.

Preferably, an overlap measure should be computationally efficient and give a reasonable estimate of the degree of overlap of two polytopes. A general discussion of overlap measures in the context of translational packing algorithms can be found in Nielsen [26].

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 an overlap measure, and it satisfies the following relation*:

$$V^d(R_0) = V^d\big(F^+, G^-\big) + V^d\big(F^-, G^+\big) - V^d\big(F^+, G^+\big) - V^d\big(F^-, G^-\big).$$

Proof. The theorem essentially follows from Theorem 1 and the proof given for the Intersection Area Theorem in Egeblad et al. [13], 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.

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

4.2. Minimum area translations

In the following we describe an efficient algorithm for solving 1D*d*DTP with respect to volume measure using Theorem 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., $f(t) = \{p(t) \in \mathbb{R}^d \mid p \in f\}$. 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 $\{f(t) \mid f \in F\}$.

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. In order to express volumes in higher dimensions, we use the concept of a *simplex*. A simplex is the d-dimensional analogue of a triangle and it can be defined as the convex hull of a set of $d+1$ affinely independent points.

Before describing how to calculate $V(f(t), g)$, we first make a few general observations on $R(f(t), g)$; recall that $V(f(t), g) = V(R(f(t), g))$.

Definition 5 *(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

$$\operatorname{proj}_f(g) = \big\{p \in f \mid \exists t \in \mathbb{R}\colon\ p(t) \in g\big\}. \tag{2}$$

In other words, $\operatorname{proj}_f(g)$ 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 $f' = \operatorname{proj}_g(f)$ and $g' = \operatorname{proj}_f(g)$; 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 $h = \operatorname{proj}_{\overline{h}}(f) \cap \operatorname{proj}_{\overline{h}}(g)$. 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 Figs. 4a and 4b for an example.

Finding the intersection of the projections of f and g is relatively straightforward. Since we require the facets to be convex, the projections are also convex and can therefore be represented using half-spaces. The intersection can then easily be represented by the intersection of all of the half-spaces from the two sets and the main problem is then to find the corresponding vertex representation.

If the intersection h is empty or has dimension smaller than $d-1$ then the volume $V(f(t), g)$ is 0 for any t value. So, assume we have a $(d-1)$-dimensional intersection h, i.e., a non-degenerate intersection. Let $p_1, \ldots, p_n$ be the vertices of g'. For each point p_i, there exists a translation value t_i such that $p_i(-t_i) \in f$. Each point $p_i(-t_i)$ is a vertex of f', and each t_i value represents the horizontal distance of p_i from the hyperplane $\overline{f}$. Assume that the points p_i are sorted such that $t_1 \leqslant \cdots \leqslant t_n$. We refer to these points as *breakpoints* and the t_i values as their corresponding distances.

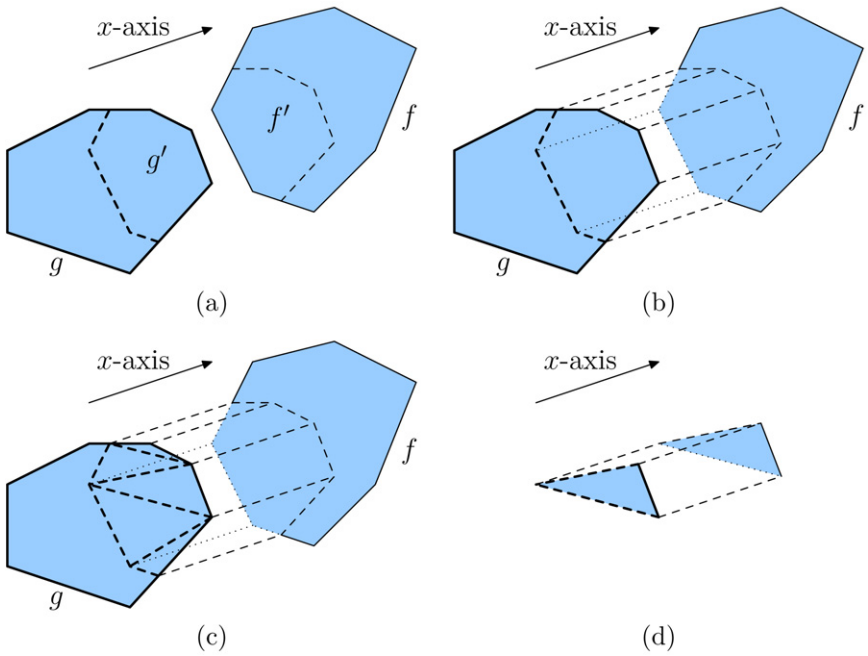

Fig. 4. An example of the inter-facet region $R(f,g)$ between two faces, f and g, in three dimensions, where g is in front of f in the x-axis direction. In general the two facets are not necessarily in parallel planes and they can also intersect. (a) The dashed lines indicate the boundary of the projections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

We now consider how to compute $V(f(t),g)$. If $t \leqslant t_1$ 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 $t \geqslant t_n$ 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.

Fig. 5 is an illustration of what happens when a face in 3D is translated through another face. This is a useful reference when reading the following. First note that the illustration emphasizes that we can also view this as the face f' passing through the plane $\overline{g}$.

An easy special case, for computing $V(f(t),g)$, occurs when $t_1 = t_n$. This corresponds to the two facets being parallel and thus $V(f(t_n),g)=0$.

Now, assume all the t_i are distinct. The case where two or more t_i are equal is discussed in Section 4.3. Each facet is required to be simple by definition and thus each vertex p_i of g' has exactly $d-1$ neighboring points, i.e., vertices of g' connected to p_i by edges. Denote these points $p_i^1,\ldots,p_i^{d-1}$ and denote the corresponding breakpoint distances $t_i^1,\ldots,t_i^{d-1}$.

Consider the value of $V(f(t),g)$ in the first interval $(t_1,t_2]$. To simplify matters, we change this to the equivalent problem of determining the function $V_0(f(t),g) = V(f(t+t_1),g)$ for $t \in (0,t_2-t_1]$. $V_0(f(t),g)$ can be described as the volume of a growing simplex in t with vertex set $\{tv_j \mid j=0,\ldots,d\}$ 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 $1 \leqslant j \leqslant d-1$. The volume of this simplex is:

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

This is illustrated in Fig. 6.

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

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

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

$$\text{2D:}\quad \frac{1}{2}\big|\det(v_2')t^2\big| = \frac{1}{2}\big|v_2^y t^2\big|$$

$$\text{3D:}\quad \frac{1}{6}\big|\det\big([v_2'v_3']\big)t^3\big| = \frac{1}{6}\big|\big(v_2^y v_3^z - v_2^z v_3^y\big)t^3\big|$$

In general $V_0(f(t),g)$ is a degree d polynomial in t.

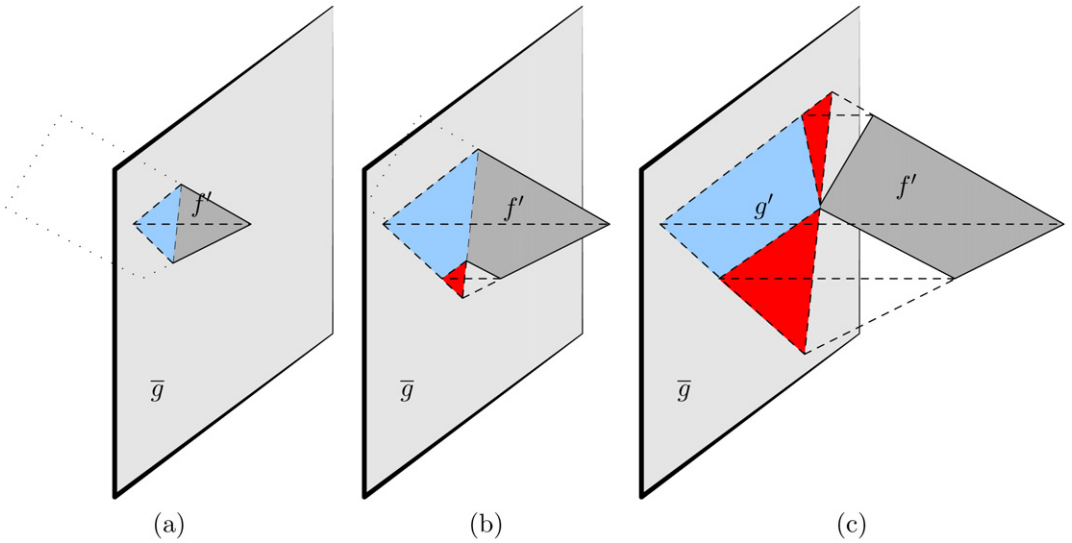

Fig. 5. An illustration of the growing tetrahedron volumes needed when a face f' passes through a face g, or equivalently its plane $\overline{g}$, in order to calculate the volume of $R(f', g')$. (a) The initial growing tetrahedron after the first breakpoint. (b) After the second breakpoint, a growing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

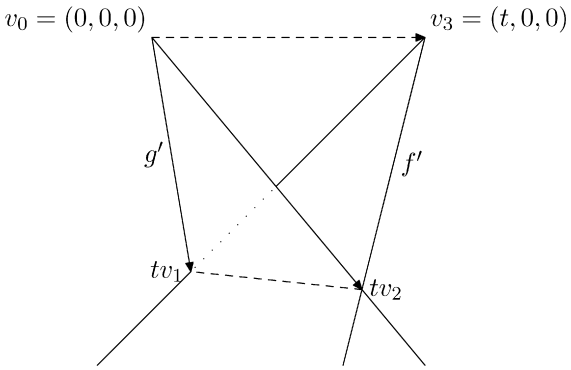

Fig. 6. In 3D, it is necessary to calculate the volume function for a growing tetrahedron based on two points, v_1 and v_2, and an overlap distance t. The coordinates of the points change linearly with respect to a change of the distance t. In d dimensions $d-1$ neighboring points are used.

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 [22] 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 Fig. 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 $\Delta(p_i, t), i \in \{2, \ldots, n-1\}$ 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 [22], we then have

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

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 Fig. 5a, there is only

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 $v(t)$ and $v_C(t)$ be polynomials with maximum degree d and initial value $v(P)$.

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 $v(t)$ by adding the coefficients of $V_i(t)$.
  if $g \in C$ **then**
    Modify $v_C(t)$ by adding the coefficients of $V_i(t)$.
  if $v_C(t) = 0$ *for* $t \in [t_i, t_{i+1}]$ **then**
    Find the minimum value t'_i for which $v(t)$ is minimized in $[t_i, t_{i+1}]$.
return t'_i *with smallest* $v(t'_i)$

Algorithm 1. Determine minimum overlap translation along x-axis in d dimensions.

the first growing simplex (tetrahedron). After that, in Fig. 5b, another growing simplex needs to be subtracted from the first, and finally, in Fig. 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 1. 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 $t_1 \leqslant t_2 \leqslant \cdots \leqslant t_n$. The algorithm traverses the breakpoints in this order while maintaining a total volume function $v(t)$ which describes the volume of the total overlap between P and $Q \cup C$. The volume function $v(t)$ is a linear combination of the volumes of simplices for each breakpoint encountered so far (based on Eq. (3)) and may be represented as a degree d polynomial in t.

Initially $v(t) = v(P)$ for $t \leqslant t_1$, corresponding to P being completely outside the container (overlapping C). As each breakpoint t_i is reached, the algorithm adds an appropriate volume polynomial to $v(t)$. Since $v(t)$ is a sum of polynomials of at most degree d, $v(t)$ 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 $v(t)$ by simply adding its coefficients to the coefficients of $v(t)$.

The subset of volume polynomials which comes from breakpoints related to the container may also be added to a volume polynomial $v_C(t)$. Whenever this function is 0 for a given t-value, it means that $P(t)$ is inside the container.

For each interval $(t_i, t_{i+1}]$ between succeeding breakpoints for which $v_C(t) = 0$, $v(t)$ can be analyzed to determine local minima. Among all these local minima, we select the smallest distance value $t_{\min}$ for which $v(t_{\min})$ 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 $d \leqslant 5$ 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 2. *Given two polytopes P and Q, assume that the* 1*-dimensional translation problem in n dimensions has a solution (a translation distance t') where P does not overlap Q then there also exists a breakpoint distance t_b for which P does not overlap Q.*

Proof. Assume that t' is not a breakpoint distance. First assume t' is in an interval (t_b^1, t_b^2) of breakpoint distances. Assume that it is the smallest such interval. Since the overlap value, $v(t')$, is 0 and t' is not a breakpoint distance then, specifically, no facets from P and Q can intersect for $t = t'$. Since there are no other breakpoint distances in this interval, and facets can only begin to intersect at a breakpoint distance, no facets can intersect in the entire interval (t_b^1, t_b^2). From the discussion

in Section 4.2.1, $\nu(t)$ must be a sum of constants or linear functions for $t \in (t_b^1, t_b^2)$ since no facets intersect in this interval. $\nu(t)$ cannot be linear since t' is not an interval end-point and this would imply a negative overlap at one of the interval end-points which is impossible. Therefore $\nu(t) = 0$ for $t \in (t_b^1, t_b^2)$. By continuity $\nu(t_b^1) = 0$ and $\nu(t_b^2) = 0$ and we may choose either of these breakpoints as t_b.

Now assume t' is within the half-open infinite interval either before the first breakpoint or after the last breakpoint. Again, since $\nu(t)$ is linear on that entire interval and $\nu(t)$ cannot be negative, one can select the breakpoint of the infinite interval as t_b. □

If a non-overlapping position of P exists for a given translation direction, then P is non-overlapping at one of the breakpoint distances. Our solution method for dDDPP, as described in Section 5, repeatedly solves 1DdDTP problems using Algorithm 1. 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 1, we assume that either one does not find minima between breakpoints or one considers this a constant time operation (which is true for 5 dimensions or less). For a fixed dimension d, the time needed for finding the intersection of $(d-1)$-simplices can also be considered a constant, and the same is true for the calculation of the determinants used in the volume functions. Given two polytopes with complexity $O(n)$ and $O(m)$, the number of breakpoints generated is at most a constant times nm. Computing the volume function for each pair of breakpoints takes constant time and thus the running time is dominated by the sorting of breakpoints revealing a running time of $O(nm\log(nm))$. In most cases, the number of breakpoints is likely to be much smaller than $O(nm)$ since the worst case scenario requires very unusual non-convex polytopes—the vertical extents must overlap for all pairs of edges. If the complexity of the facets is bounded (e.g. triangles for $d = 3$), then the number of breakpoints generated for two polytopes with n and m *facets*, respectively, is at most $O(nm)$, and the asymptotic running time is $O(nm\log(nm))$.

In some settings it may be desirable to allow for overlap with the region outside the container. This is easily accommodated by ignoring the condition that $v_C(t) = 0$ in Algorithm 1.

4.3. Special cases in three dimensions

In the previous subsection, it was assumed that either all breakpoints had unique distance values or that all breakpoints had the same distance value. Here we describe how to handle a subset of breakpoints with the same distance value for the case where $d = 3$. In particular, we focus on the special case of the two first breakpoint distances being equal (and different than the subsequent ones).

Given two convex faces f and g, we know that h, the intersection of their 2D projections onto the yz-plane, is also a convex polygon. As before, let f' and g' denote the projections of h onto the given faces. When $\overline{f}$ and $\overline{g}$ are not parallel, then for any given translation of f', the intersection between f' and g' contains at most two corner points. Furthermore, Eq. (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. Fig. 7a illustrates the standard case with only a single initial breakpoint while Figs. 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 Fig. 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 Fig. 7b is given in Fig. 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 Fig. 7d cannot occur. The special case in Fig. 7b can also be avoided if the intersection polygon is triangulated and each triangle is handled separately (see Fig. 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 *d*DSPP

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 1. In the following, we first review the local search and the meta-heuristic technique used and described by Egeblad

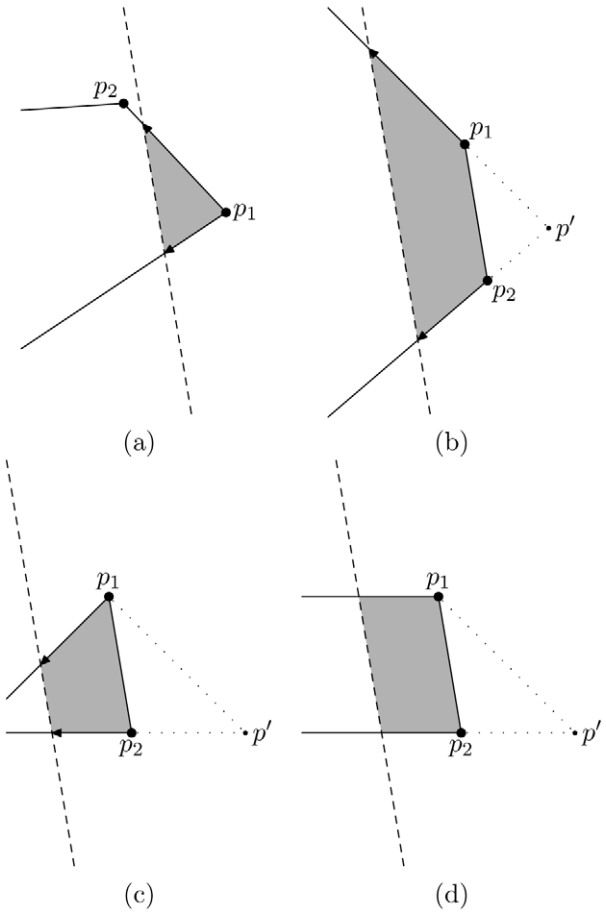

Fig. 7. Various special cases can occur when the two first breakpoint distances are equal. Here it is illustrated in 2D using a dashed line to illustrate the intersection line of the two faces during the translation. (a) The standard case in which the first breakpoint distance is unique. (b) The two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

et al. [13]. 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 *d*DDPP (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 [30]. 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 *d*DDPP and the strip-height is decreased and all polyhedrons are moved inside the smaller container.

Minimum overlap translations are found using the axis-aligned translation algorithm described in the previous section. For each polytope each of the possible d axis-aligned translations are used and the direction which reveals the position with least overlap is chosen. Note that if $P = Q_i$ is the polytope undergoing translation then the polytope Q (in the translation algorithm in the previous section) is actually the union of all other polytopes, i.e., $Q = \bigcup_{j=1,\, j\neq i}^{n} Q_j$. 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

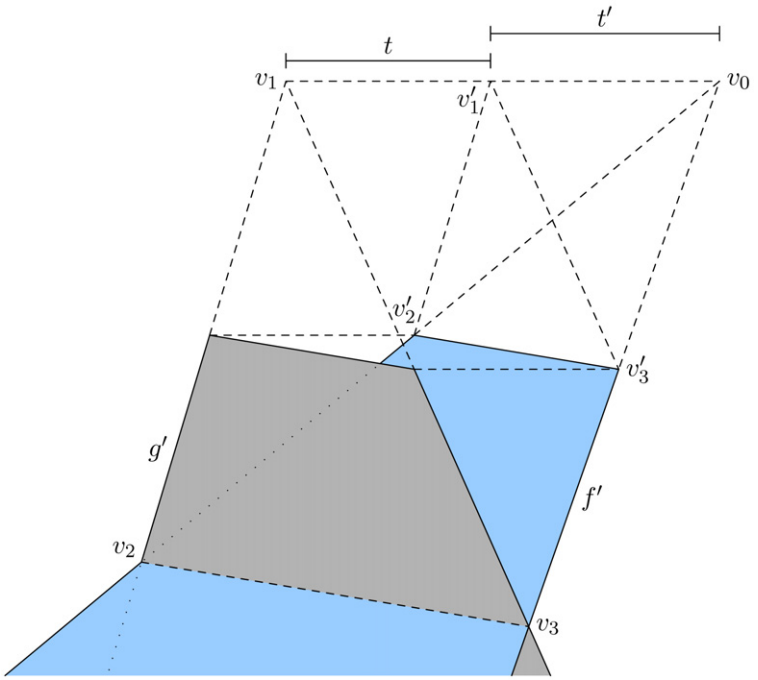

Fig. 8. To calculate the volume between the faces f' and g', one can base the calculations on the extended faces illustrated with dashed lines. The volume is a growing tetrahedron (v_0, v_1, v_2, v_3) subtracted by a constant volume (the tetrahedron (v_0, v'_1, v'_2, v'_3)) and subtracted by a linearly growing volume based on the triangle (v'_1, v'_2, v'_3) and the translation distance t.

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

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

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

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 $1 \leqslant i < j \leqslant n$. Whenever the search reaches a local minimum with $g(\mathbf{P}) > 0$, the value of $\phi_{i,j}$ is increased for the pair Q_i and Q_j with highest $\mu_{i,j}(\mathbf{P})$ where $\mu_{i,j}(\mathbf{P}) = \frac{V_{i \cap j}(\mathbf{P})}{1 + \phi_{i,j}}$. We refer to this as *penalizing* the pair Q_i and Q_j. Intuitively, this change of the objective function $g(\mathbf{P})$ (if large enough), allows the local search to move Q_i and Q_j away from each other, even if it results in greater overlap. Ideally, this move causes the local search to reach a different part of the solution space. To avoid large discrepancy between the real and penalized solution space, the penalties are reset from time to time. To avoid searching the entire neighborhood in each iteration of GLS, we also apply *fast local search* [30].

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

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

5.2. Initial solution

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

In any dimension, a naive initial height can be based on the sum of heights of all polytopes, but in the following, we describe a more ambitious strategy. In short, we use a greedy bounding box based algorithm in which the polytopes are placed one by one inside the container in an order of decreasing bounding box volume. This algorithm is based on residual volumes and is related to the approach used by Eley [15] for the container loading problem in three dimensions. Although the algorithm could be generalized to higher dimensions, we are only going to describe its three-dimensional variant.

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

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

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

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

These represent the volumes left, below, behind, right, above, and in-front of B_i, respectively. If any of the intervals are empty then the new space is empty. Each of the new non-degenerate 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 [13]. 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 [18], i.e., it is verified that no polyhedron is involved in any significant overlap with other polyhedra or the container. In the experiments presented in this section, the largest total volume of overlap allowed in a solution corresponds to 0.01% of the total volume of all polyhedrons for the given problem.

All experiments were performed on a system with a 2.16 GHz Intel Core Duo processor with 2 MB of level 2 cache and 1 GB of RAM. Note that the implementation only uses one core of the processor.

6.1. Problem instances

The literature on the subject of three-dimensional packing contains only few useful problem instances with regard to a comparison of results. We have found two appropriate data sets for our experiments. The first one was introduced by Ikonen et al. [21] and the second one was introduced by Stoyan et al. [28]. 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,

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

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

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. This polyhedron can be found by generating the set, I, of all intersection points of distinct planes p, q, r with $p, q, r \in \mathcal{H}$, and then generate the convex hull $\mathcal{C}$ of the subset of points from I which are contained in all of the half-spaces. An elegant way to find the points of the convex hull is by using the concept of dualisation (see de Berg et al. [9]). Let $\mathcal{C}^d$ be the convex hull of the dual points of $\mathcal{H}$, then the planes of the facets of $\mathcal{C}^d$ are duals of the corner points of $\mathcal{C}$. This allows one to find $\mathcal{C}$ in time $O(n \log n)$ (de Berg et al. [9]) using just a convex hull algorithm.

Given a positive integer n, the construction of an n-piece puzzle commences as follows. Initially, a set of 6 half-spaces, H_0, is generated such that they correspond to a cube. Now let $\mathcal{P}_1 = \{H_0\}$ then we will iteratively construct a sequence of half-space sets $\mathcal{P}_i$. To do this, we select the smallest cardinality half-space set $H \in \mathcal{P}_i$ for each i and generate a random plane which can be used to split H into two new sets H' and H''. We then let $\mathcal{P}_{i+1} = (\mathcal{P}_i \setminus \{H\}) \cup \{H', H''\}$, i.e., the set of half-space sets containing H' and H'' as well as all sets from $\mathcal{P}_i$ except 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 Fig. 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. [21] (see Table 1) are decision problems with a cylindrical container and they have already been shown to be easily solved by Egeblad et al. [13]. The only previous results and thus also the best results for the Stoyan instances are reported by Stoyan et al. [28] and their results are repeated in the first two columns of Table 2.

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. [28] 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.,

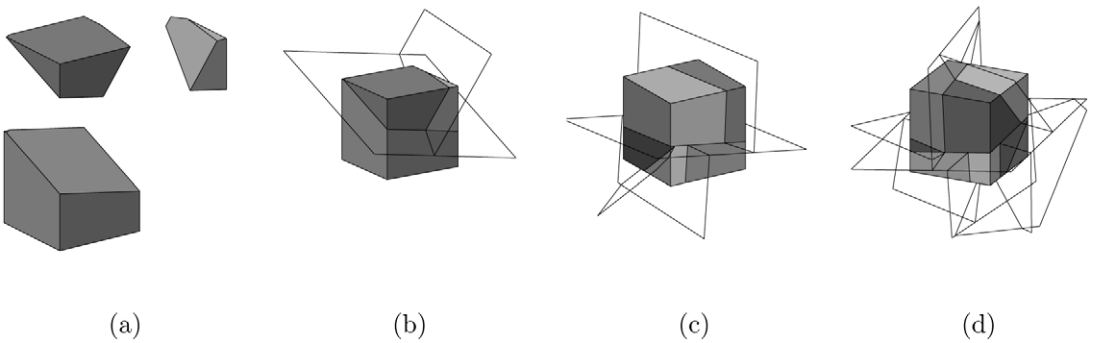

Fig. 9. Examples of three different puzzles and the cutting planes used to generate them. (a) Three convex polyhedra. (b) The corresponding cube and cutting planes. (c) A puzzle with 5 pieces. (d) A puzzle with 10 pieces.

Table 2
The results from Stoyan et al. [28] 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.

Problem	Stoyan		Bounding box		3DNest		Improv.
	Height	Util. (%)	Height	Util. (%)	Height	Util. (%)	
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 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

Problem	Size	Utilization after number of seconds					Max. util.	Std. dev.	Translations per second
		0	10	60	300	600			
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

and the average improvement over the utilization of Stoyan et al. is 50.2%. This demonstrates that 3DNest performs very well in comparison to existing methods.

In order to make some additional experiments, a new problem instance called Merged1 was created by combining the polyhedra from Stoyan and Ikonen. It contains one copy of each of the polyhedra in Table 1 and the Ikonen polyhedra have been scaled with a factor of 4 to better match the size of the Stoyan polyhedra. Larger versions of this problem instance called Merged*i* are simply created by making *i* number of copies of each polyhedron. The dimensions of the container for these instances are chosen such that a solution with 50% utilization is a cube. Furthermore, we have used the puzzle generator described above to generate 40 puzzles with 5, 10, 20, and 40 pieces. Rather than testing each puzzle with 10 different random seeds, 10 different puzzles are tested for each of the four cardinalities. The purpose of this is to illustrate the heuristic's capabilities independently of the input data. Each shape of these puzzles has an average of about 11–13 facets which is very similar to the Stoyan instances.

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

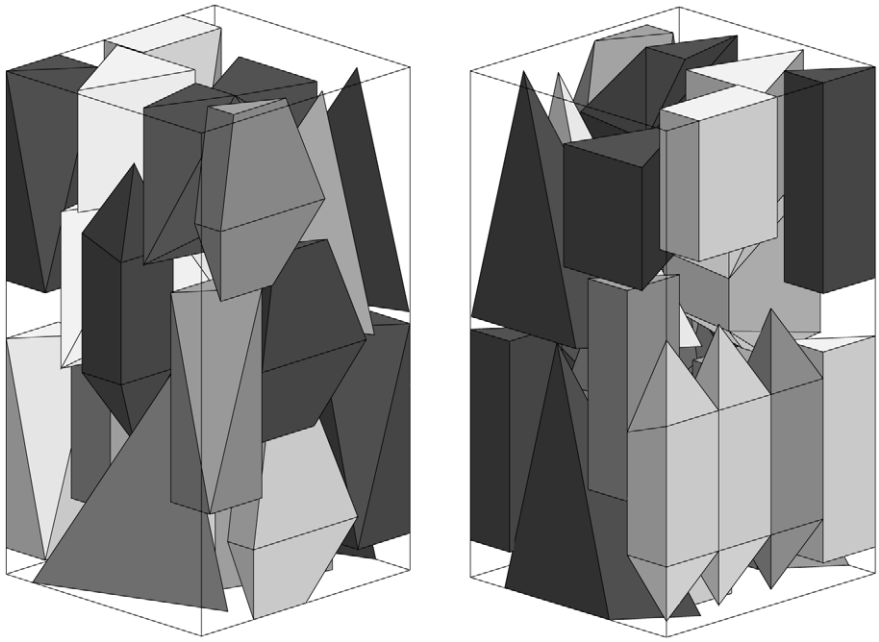

Fig. 10. The best solution found for Stoyan3 without using rotation (from two different angles). The utilization is 46.67%.

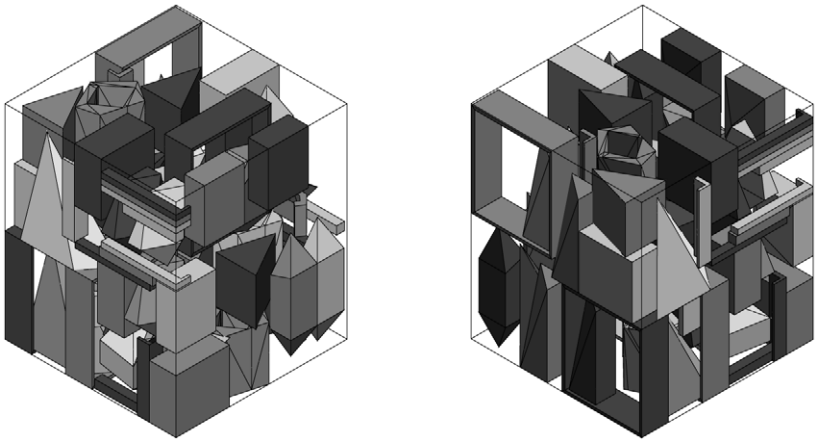

Fig. 11. The best solution found for Merged5 without using rotation (from two different angles). The utilization is 42.12%.

After 10 seconds the results are already better than those of Stoyan et al. for all of the Stoyan instances with an average utilization close to 40%. The heuristic continues to improve solutions but utilization is only improved by less than 1 percentage point after 300 seconds. Solutions for the Merged instances appear to be quite good with utilizations matching the smaller Stoyan instances even with as many as 60 and 75 shapes. Also here, solutions are generally only improved by one percentage point after the first 300 seconds with the exception of Merged5. The optimal utilization for Merged5 is probably higher than it is for Merged1 which is also indicated by the initial solutions, but the slow decline in the number of translations performed is also a strong indication that large problem instances are solved efficiently by 3DNest. Puzzles with 5 or 10 pieces are most often solved to optimality, and even puzzles with 20 pieces are handled quite well within the time limit of 10 minutes. The average utilization of these instances is only 50% after 10 minutes, and the best found utilization is less than 71% which is far from the optimal 100%. The best solutions found for Stoyan3 and Merged5 are shown in Figs. 10 and 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

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

Problem	Size	Average height	Utilization			Translations per second
			Avg.	Min.	Max.	
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

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.

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. [13] and proved very successful for two dimensions. Three problem instances in three dimensions, by Stoyan et al. [28], were used to show that the presented solution method is able to reach far better results than those by Stoyan et al. [28]. 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 [19], quality regions [19], repeated patterns [25] 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. [24] described what is essentially a 2D translation algorithm in 2D space (solving 2D2DTP). Given polygons with n and m edges, the worst case running time is $O((mn)^2)$. Their approach is based on an *arrangement* of line segments. Decomposition techniques for $d \geqslant 3$ have been studied by several authors (see de Berg et al. [7]), and it would be interesting if these methods can be used to generalize the solution method for 2D2DTP to dDdDTP. It is also an open question how to make an algorithm for 2D3DTP or more generally d'DdDTP for any $d \geqslant 3$ and $d' \in \{2, \ldots, d-1\}$. For the sake of completion, de Berg et al. [8] solve the maximization variant of 2D2DTP with two convex polygons in time $O((n+m)\log(n+m))$. Ahn et al. [1] consider a generalization of the same problem in which they allow rotation.

Finally, Cheong et al. [5] present an approximation algorithm, that finds a translation for two general polygons where the area of overlap is at least $\mu_{\text{opt}} - \varepsilon$, for some given value ε and μ_{opt} is the maximal overlap of any translation. If the polygons have complexity n and m, the running time of their algorithm is $O(m + (n^2/\varepsilon^4)\log^2 n)$ and $O(m + (n^3/\varepsilon^4)\log^5 n)$, if rotations are allowed.

Free orientation of shapes is one of the most important directions for future research. Especially when considering that most applications of packing 3D shapes, e.g., rapid prototyping, do allow free orientation. Another important direction for

future research is how to also handle some of the constraints which are typically part of more general layout problems, e.g., constraints concerning gravity or wire length [4].

References

[1] H.-K. Ahn, O. Cheong, C.-D. Park, C.-S. Shin, A. Vigneron, Maximizing the overlap of two planar convex sets under rigid motions, Computational Geometry 37 (1) (2006) 3–15.
[2] A. Bortfeldt, H. Gehring, D. Mack, A parallel tabu search algorithm for solving the container loading problem, Parallel Computing 29 (2002) 641–662.
[3] J. Cagan, D. Degentesh, S. Yin, A simulated annealing-based algorithm using hierarchical models for general three-dimensional component layout, Computer Aided Design 30 (10) (1998) 781–790.
[4] J. Cagan, K. Shimada, S. Yin, A survey of computational approaches to three-dimensional layout problems, Computer-Aided Design 34 (8) (2002) 597–611.
[5] O. Cheong, A. Efrat, S. Har-Peled, Finding a guard that sees most and a shop that sells most, Discrete & Computational Geometry 37 (4) (2007) 545–563.
[6] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, W. Wright, Simplification envelopes, Computer Graphics 30 (1996) 119–128 (Annual Conference Series).
[7] M. de Berg, L.J. Guibas, D. Halperin, Vertical decompositions for triangles in 3-space, Discrete & Computational Geometry 15 (1) (1996) 35–61.
[8] M. de Berg, O. Cheong, O. Devillers, M. van Kreveld, Computing the maximum overlap of two convex polygons under translations, Theory of Computing Systems 31 (5) (1998) 613–628.
[9] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, Computational Geometry: Algorithms and Applications, second ed., Springer, 2000.
[10] J.K. Dickinson, G.K. Knopf, A moment based metric for 2-D and 3-D packing, European Journal of Operational Research 122 (1) (2000) 133–144.
[11] J.K. Dickinson, G.K. Knopf, Packing subsets of 3d parts for layered manufacturing, International Journal of Smart Engineering System Design 4 (3) (2002) 147–161.
[12] K.A. Dowsland, W.B. Dowsland, Solution approaches to irregular nesting problems, European Journal of Operational Research 84 (1995) 506–521.
[13] J. Egeblad, B.K. Nielsen, A. Odgaard, Fast neighborhood search for two- and three-dimensional nesting problems, European Journal of Operational Research 183 (3) (2007) 1249–1266.
[14] F. Eisenbrand, S. Funke, A. Karrenbauer, J. Reichel, E. Schömer, Packing a trunk: Now with a twist!, in: SPM '05: Proceedings of the 2005 ACM Symposium on Solid and Physical Modeling, New York, NY, USA, ACM Press, 2005, pp. 197–206.
[15] M. Eley, Solving container loading problems by block arrangement, European Journal of Operational Research 141 (2) (2002) 393–409.
[16] O. Faroe, D. Pisinger, M. Zachariasen, Guided local search for the three-dimensional bin packing problem, INFORMS Journal on Computing 15 (3) (2003) 267–283.
[17] R.J. Fowler, M.S. Paterson, S.L. Tanimoto, Optimal packing and covering in the plane are NP-complete, Information Processing Letters 12 (3) (1981) 133–137.
[18] P. Hachenberger, L. Kettner, 3D Boolean operations on nef polyhedra, in: C.E. Board (Ed.), CGAL-3.2 User and Reference Manual, 2006.
[19] J. Heistermann, T. Lengauer, The nesting problem in the leather manufacturing industry, Annals of Operations Research 57 (1995) 147–173.
[20] S.-M. Hur, K.-H. Choi, S.-H. Lee, P.-K. Chang, Determination of fabricating orientation and packing in SLS process, Journal of Materials Processing Technology 112 (2–3) (2001) 236–243.
[21] I. Ikonen, W.E. Biles, A. Kumar, J.C. Wissel, 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 Algorithms, East Lansing, Michigan, Morgan Kaufmann Publishers, 1997, pp. 591–598.
[22] J. Lawrence, Polytope volume computation, Mathematics of Computation 57 (195) (1991) 259–271.
[23] A. Lodi, S. Martello, D. Vigo, Heuristic algorithms for the three-dimensional bin packing problem, European Journal of Operational Research 141 (2) (2002) 410–420.
[24] D.M. Mount, R. Silverman, A.Y. Wu, On the area of overlap of translated polygons, Computer Vision and Image Understanding 64 (1) (1996) 53–61.
[25] B.K. Nielsen, An efficient solution method for relaxed variants of the nesting problem, in: J. Gudmundsson, B. Jay (Eds.), Theory of Computing, Proceedings of the Thirteenth Computing: The Australasian Theory Symposium, Ballarat, Australia, in: CRPIT, vol. 65, ACS, 2007, pp. 123–130.
[26] B.K. Nielsen, Nesting problems and Steiner tree problems, PhD thesis, DIKU, University of Copenhagen, Denmark, 2008.
[27] 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.
[28] Y.G. Stoyan, N.I. Gil, G. Scheithauer, A. Pankratov, I. Magdalina, Packing of convex polytopes into a parallelepiped, Optimization 54 (2) (2005) 215–235.
[29] P.E. Sweeney, E.R. Paternoster, Cutting and packing problems: A categorized, application-orientated research bibliography, Journal of the Operational Research Society 43 (7) (1992) 691–706.
[30] C. Voudouris, E. Tsang, Guided local search and its application to the traveling salesman problem, European Journal of Operational Research 113 (1999) 469–499.
[31] G. Wäscher, H. Haussner, H. Schumann, An improved typology of cutting and packing problems, European Journal of Operational Research (2006).
[32] X. Yan, P. Gu, A review of rapid prototyping technologies and systems, Computer Aided Design 28 (4) (1996) 307–318.
[33] S. Yin, J. Cagan, An extended pattern search algorithm for three-dimensional component layout, Journal of Mechanical Design 122 (1) (2000) 102–108.
[34] S. Yin, J. Cagan, Exploring the effectiveness of various patterns in an extended pattern search layout algorithm, Journal of Mechanical Design 126 (1) (2004) 22–28.