Available online at www.sciencedirect.com
ScienceDirect
European Journal of Operational Research 183 (2007) 1249–1266
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

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

Jens Egeblad, Benny K. Nielsen *, Allan Odgaard

Department of Computer Science, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen Ø, Denmark

Received 30 September 2004; accepted 22 November 2005
Available online 21 June 2006

Abstract

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

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

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

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

Keywords: Cutting; Packing; Nesting; 3D nesting; Guided local search

1. Introduction

Nesting is a term used for many related problems. The most common problem is strip-packing where a number of irregular shapes must be placed within a rectangular strip such that the strip-length is minimized and no shapes overlap. The clothing industry is a classical example of an application for this problem. Normally, pieces of clothes are cut from a roll of fabric. A high utilization is desirable and it requires that as little of the roll is used as possible. The width of the roll is fixed, hence the problem is to minimize the length of the fabric. Other nesting problem variations exist, but in the following the focus is on the strip-packing variant. Using the typology of Wäscher et al. (2006) this is a two-dimensional irregular open dimension problem (ODP).

In the textile industry the shapes of the pieces of clothes are usually referred to as *markers* or *stencils*.

* Corresponding author. Tel.: +45 35 32 14 00; fax: +45 35 32 14 01.
E-mail addresses: jegeblad@diku.dk (J. Egeblad), benny@diku.dk (B.K. Nielsen), duff@diku.dk (A. Odgaard).

0377-2217/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2005.11.063

In the following we will use the latter term except when the pieces need to be defined more precisely e.g. as polygons.

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

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

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

The strip-packing variant can now be stated as

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

The Strip Nesting Problem is NP-hard (e.g. Nielsen and Odgaard, 2003).

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

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

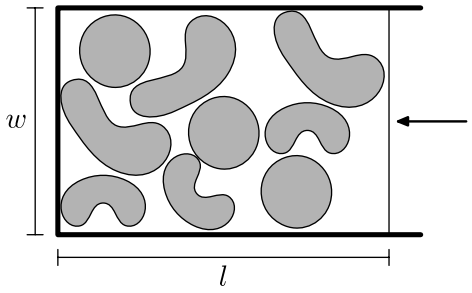

Fig. 1. The Strip Nesting Problem. Place a number of stencils on a strip with width w such that no two stencils overlap and the length l of the strip is minimized.

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

2. Existing approaches to nesting problems

There exists numerous solution methods for nesting problems. A thorough survey by Dowsland and Dowsland (1995) exists, but a more recent survey has also been done by Nielsen and Odgaard (2003). Meta-heuristics are one of the most popular tools for solving nesting problems. A detailed discussion of these can be found in the introductory sections of Bennell and Dowsland (1999).

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

2.1. Basic solution methods

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

2.1.1. Legal placement methods

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

Most methods for strip-packing follow the basic steps below.

(1) Determine a sequence of stencils. This can be done randomly or by sorting the stencils according to some measure e.g. the area or the degree of convexity (Oliveira et al., 2000).
(2) Place the stencils with some first/best fit algorithm. Typically a stencil is placed at the contour of the stencils already placed. Some algorithms also allow hole-filling i.e. placing a stencil in an empty area between already

placed stencils (Dowsland et al., 1998, 2002; Gomes and Oliveira, 2002).

(3) Evaluate the length of the solution. Exit with this solution (Art, 1966) or repeat at step 2 after changing the sequence of stencils (Dowsland et al., 1998; Gomes and Oliveira, 2002).

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

Legal placement methods not doing a sequential placement do exist. These methods typically construct a legal initial solution and then introduce some set of moves (e.g. swapping two stencils) that can be controlled by a meta-heuristic to e.g. minimize the length of a strip (Blazewicz et al., 1993; Burke and Kendall, 1999a,b,c; Gomes and Oliveira, 2006).

2.1.2. Relaxed placement methods

The obvious alternative is to allow overlaps to occur as part of the solution process. The objective is then to minimize the amount of overlap. A legal placement has been found when the amount of overlap reaches 0. Numerous papers applying such a scheme exist with varying degrees of success (Bennell and Dowsland, 1999, 2001; Heckmann and Lengauer, 1995; Jain et al., 1992; Jakobs, 1996; Lutfiyya et al., 1992; Oliveira and Ferreira, 1993; Theodoracatos and Grimsley, 1995). Especially noteworthy is the work of Heckmann and Lengauer (1995).

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

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

2.2. Geometric approaches

The first problem encountered when handling nesting problems is how to represent the stencils. If the stencils are not already given as polygons then they can quite easily be approximated by polygons which is also what is done in most cases. A more crude approximation can be done using a *raster model* (Chen et al., 2004; Heckmann and Lengauer, 1995; Lutfiyya et al., 1992; Oliveira and Ferreira, 1993). This is a discrete model of the stencils created by introducing a grid of some size to represent the material i.e. each stencil covers some set of raster squares. The stencils can then be represented by matrices. An example of a simple polygon and its raster model equivalent is given in Fig. 2. A low granularity of the raster model provides fast calculations at the expense of limited precision. Better precision requires higher granularity, but it will also result in slower calculations.

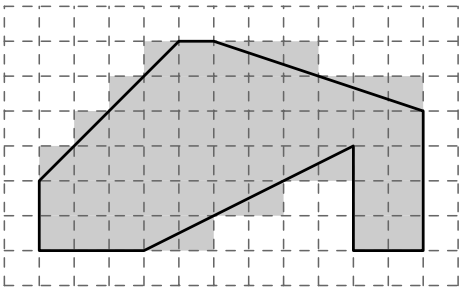

Fig. 2. The raster model requires all stencils to be defined by a set of grid squares. The drawing above is an example of a polygon and its equivalent in a raster model.

Comparisons between the raster model and the polygonal model were done by Heckmann and Lengauer (1995) and they concluded that the polygonal model was the better choice for their purposes.

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

2.2.1. Overlap calculations

The area of an overlap between two polygons (see Fig. 3a) can be used to determine whether polygons overlap and how much they overlap. This can be an expensive calculation and thus quite a few alternatives have been suggested such as *intersection depth* (Bennell and Dowsland, 1999; Dobkin et al., 1993) (see Fig. 3b) and the Φ-function (Stoyan et al., 2004) which can differentiate between three states of polygon interference: Intersection, disjunction and touching.

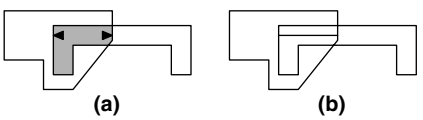

Fig. 3. The degree of overlap can be measured in various ways. Here are two examples: (a) The precise area of the overlap. (b) The horizontal intersection depth.

Solution methods using overlap calculations most often apply some kind of trial-and-error scheme i.e. they try to place or move a polygon to various positions to see if or how much it overlaps. This can then be used to improve some intermediate solution which might be allowed to contain overlap (Bennell and Dowsland, 1999, 2001; Blazewicz and Walkowiak, 1995; Heckmann and Lengauer, 1995).

2.2.2. No-fit-polygons (NFP)

Legal placement methods very often use the concept of the *No-Fit-Polygon* (NFP) (Adamowicz and Albano, 1976; Albano and Sappupo, 1980; Art, 1966; Dowsland et al., 1998, 2002; Gomes and Oliveira, 2002, 2006; Oliveira et al., 2000), although it can also be used in relaxed placement methods as done by Bennell and Dowsland (2001).

The NFP is a polygon which describes the legal/illegal placements of one polygon in relation to another polygon, and it was introduced by Art (1966) (although named *envelope*).

Given two polygons P and Q the construction of the NFP of P in relation to Q can be found in the following way: Choose a reference point for P. Slide P around Q as closely as possible without intersecting. The trace of the reference point is the contour of the NFP. An example can be seen in Fig. 4. To determine whether P and Q intersect it is only necessary to determine whether the reference point of P is inside or outside their NFP. Placing polygons closely together can be done by placing the reference point of P at one of the edges of the NFP. If P and Q have s and t edges, respectively, then the number of edges in their NFP will be $O(s^2t^2)$ (Asano et al., 2002).

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

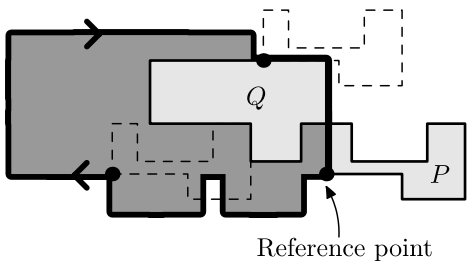

Fig. 4. Example of the *No-Fit-Polygon* (thick border) of stencil P in relation to stencil Q. The reference point of P is not allowed inside the NFP if overlap is to be avoided.

2.3. Meta-heuristics

Both legal and relaxed placement methods can make use of meta-heuristics. The most popular one for nesting problems is *Simulated Annealing* (SA) (Burke and Kendall, 1999c; Gomes and Oliveira, 2006; Heckmann and Lengauer, 1995; Lutfiyya et al., 1992; Oliveira and Ferreira, 1993; Theodoracatos and Grimsley, 1995). The most advanced use of it is by Heckmann and Lengauer (1995) who implemented SA in four stages. The first stage is a rough placement, the second stage eliminates overlaps, the third stage is a fine placement with approximated stencils and the last stage is a fine placement with the original stencils.

Gomes and Oliveira (2006) very successfully combine SA with the ideas for compaction and separation by Li and Milenkovic (1995). A very similar approach had previously been attempted by Bennell and Dowsland (1999, 2001), but they combined it with a *Tabu Search* variant.

More exotic approaches are genetic, ant and evolutionary algorithms (Burke and Kendall, 1999a,b; Jakobs, 1996)—all with very limited success.

3. Solution method outline

In this section we will give a brief outline of our solution method. Our method is a relaxed placement method and it can handle irregular polygons with holes. A new geometric approach is utilized and the *Guided Local Search* meta-heuristic is used to escape local minima. This approach is inspired by a paper by Faroe et al. (2003) which presented a similar approach for the two-dimensional Bin Packing Problem for rectangles.

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

(1) Finding an initial strip length
    An initial strip length is found by using some fast heuristic e.g. a bottom-left bounding box placement algorithm.
(2) Reducing the strip length
    The strip length is reduced by some value. This value could e.g. be based on some percentage

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

(3) Applying local search to reduce overlap
The strip length is fixed now and the search for a solution without overlap can begin. The overlap is iteratively reduced by applying local search. More precisely, in each step of the local search a polygon is moved to decrease the total amount of overlap. The local search and its neighborhood are described in Section 4, and a very efficient search of the neighborhood is the focus of Section 6.
If a placement without overlap is found for the current fixed strip length then we have found a solution, and step 2 can be repeated to find even better solutions. This might not happen though since the local search can be caught in local minima.

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

4. Local search

4.1. Placement

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

4.2. Objective function

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

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

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

4.3. Neighborhood

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

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

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

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

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

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

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

5. Guided local search

To escape local minima encountered during local search we apply the meta-heuristic *Guided Local Search* (*GLS*). GLS was introduced by Voudouris and Tsang (1995) and has previously been successfully applied to e.g. the *Traveling Salesman Problem* (Voudouris and Tsang, 1999) and two- and three-dimensional *Bin-Packing Problems* (Faroe et al., 2003).

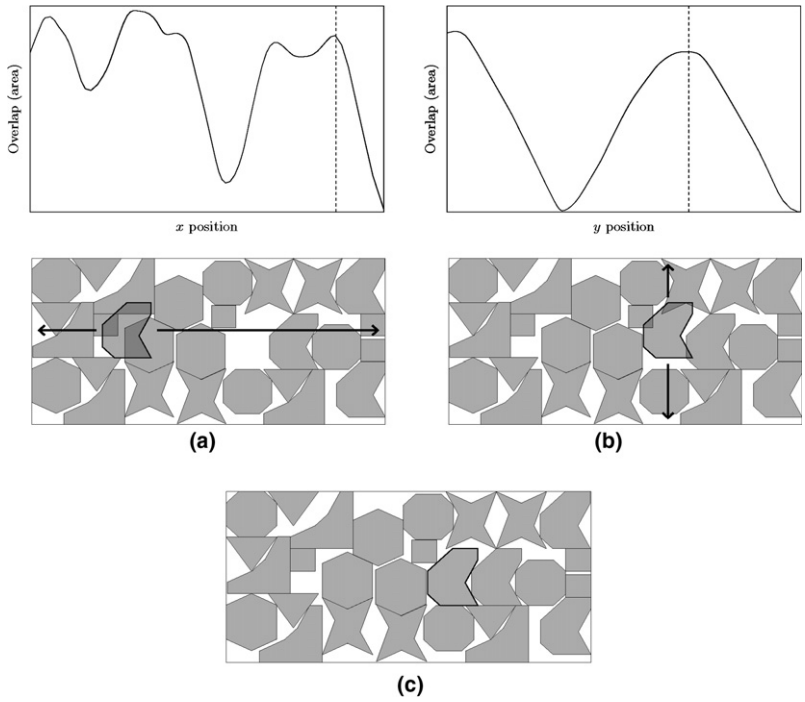

Fig. 5. Example of a local search. (a) One polygon overlaps with several other polygons and is selected for optimization. In the top row we have drawn the amount of overlap as a function of the leftmost x-coordinate of the polygon. The positions beyond the dashed line are illegal since the polygon would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

5.1. Features and penalties

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

$$I_{ij}(p) = \begin{cases} 0 & \text{if } overlap_{ij}(p) = 0, \\ 1 & \text{otherwise,} \end{cases}$$

$$i, j \in 1, \ldots, n, p \in \mathscr{P},$$

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

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

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

Algorithm 1. Guided Local Search for Nesting Decision Problem

Given a set of polygons $\mathscr{S}$.
Generate initial placement p.
for each pair of polygons $s_i, s_j \in \mathscr{S}$
  Set $\phi_{ij} = 0$.
while p contains overlap
  // *Local search*:
  while p is not local minimum
    Select polygon s_i.
    Create p' from p using the best neighborhood move of s_i, i.e. such that $h(p')$ is minimized.
    Set $p = p'$.
  // *Penalize*:
  for each pair of polygons $s_i, s_j \in \mathscr{S}$
    Compute $\mu_{ij}(p)$.
  for each pair of polygons $s_i, s_j \in \mathscr{S}$ such that μ_{ij} is maximal
    Set $\phi_{ij} = \phi_{ij} + 1$.
return p

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

5.2. Augmented objective function

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

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

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

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

5.3. Improvements

The efficiency of GLS can be greatly improved by using *Fast Local Search* (FLS) (Voudouris and Tsang, 1995). FLS divides the local search neighborhood into sub-neighborhoods which are active or inactive depending on whether they should be considered during local search. In our context we let the moves of each polygon be a sub-neighborhood resulting in n sub-neighborhoods. Now it is the responsibility of the GLS algorithm to activate each sub-neighborhood and the responsibility of FLS to inactivate them.

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

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

6. Fast neighborhood search

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

6.1. Intersection area theorem

In this section we will present a special way to determine the area of intersection of two polygons. Nielsen and Odgaard (2003) have presented a more general version of the Intersection Area Theorem which dealt with rotation and arbitrary shapes. In this paper however we have decided to limit the theory to polygons and horizontal translation since this is all we need for our algorithm to work. It will also make the proof shorter and easier to understand.

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

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

The condition, $p_y \neq \min(e_{a_y}, e_{b_y})$, is needed to handle some special cases (see Lemma 5).

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

$$\overleftarrow{f_E}(p) = |\{e \in E | \exists x' < p_x : (x', p_y) \in e\}|,$$
$$\overrightarrow{f_E}(p) = |\{e \in E | \exists x' \geqslant p_x : (x', p_y) \in e\}|.$$

Definition 3 (*Polygon*). A polygon P is defined by a set of edges E. The edges must form one or more cycles and no pair of edges from E are allowed to intersect. The *interior* of the polygon is defined by the set

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

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

Note that this is an extremely general definition of polygons. The polygons are allowed to consist of several unconnected components and cycles can

be contained within each other to produce holes in the polygon.

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

Definition 4 (*Sign of Edge*). Given a polygon P defined by an edge set E we say an edge $e \in E$ is *positive* if

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

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

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

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

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

Lemma 5. *Given a vertical coordinate y and some interval I, we say that the horizontal line $l_y(t) = (t, y)$, $t \in I$, crosses an edge e if there exists t_0 such that $l_y(t_0) \in e$. Now assume that P is a polygon defined by an edge set E then all of the following holds.*

1. *If $I =]-\infty, \infty[$ and we traverse the line from $-\infty$ towards ∞ then the edges crossed alternate between being positive and negative.*
2. *If $I =]-\infty, \infty[$ then the line crosses an even number of edges.*
3. *Assume $p \notin P$ then the infinite half-line $l_{p_y}(t)$ for $I = [p_x, \infty[$ will cross an equal number of positive and negative edges. The same is true for $I =]\infty, p_x[$.*
4. *Assume $p \in P$. If $I = [p_x, \infty[$ and if the line crosses n positive edges then it will also cross precisely $n + 1$ negative edges. Similarly, if $I =]-\infty, p_x[$ and if the line crosses n negative edges then it will also cross precisely $n + 1$ positive edges.*

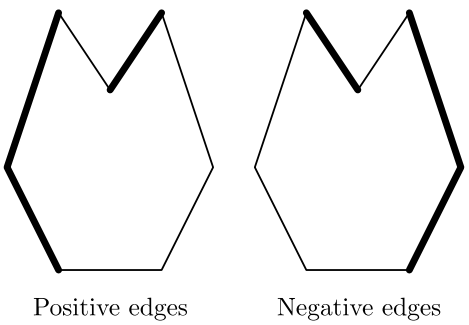

Fig. 6. Positive and negative edges of a polygon according to Definition 4.

Proof. We only sketch the proof. First note that some special cases concerning horizontal edges and the points where edges meet are handled by the inequality in Definition 1.

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

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

Definition 6 (*Containment Function*). Given two edges e_1 and e_2 and a point $p \in \mathbb{R}^2$ define the *containment function*

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

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

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

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

Definition 7 (*Edge Region and Edge Region Area*). Given two edges e_1 and e_2 we define the *edge region* $R(e_1, e_2) = \{p \in \mathbb{R}^2 | \mathbf{C}(e_1, e_2, p) = 1\}$ and the area of $R(e_1, e_2)$ as

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

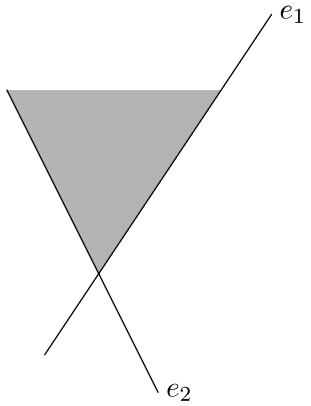

Fig. 7. The edge region $R(e_1, e_2)$ of two edges e_1 and e_2 (see Definition 7).

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

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

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

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

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

$$p \in P \cap Q \Rightarrow w(p) = 1,$$
$$p \notin P \cap Q \Rightarrow w(p) = 0,$$

where

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

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

Now let $m = \overrightarrow{f_{E^+}}(p)$ and $n = \overleftarrow{f_{F^-}}(p)$. By using Lemma 5 we can prove this theorem by counting.

First assume $p \in P \cap Q$ which implies $p \in P$ *and* $p \in Q$. From Lemma 5 we know that $\overrightarrow{f_{E^-}}(p) = m + 1$ and $\overleftarrow{f_{F^+}}(p) = n + 1$. Inserting this into Eq. (2) reveals:

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

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

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

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

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

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

Proof. From the Containment Theorem we know

$$\alpha = \int \int_{p \in \mathbb{R}^2} w(p)\, \mathrm{d}A.$$

Using Eq. (2) we get

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

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

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

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

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

6.2. Translational overlap

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

6.2.1. Calculating the area of edge regions

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

(1) Edge e is completely to the left of edge f (Fig. 8a).
(2) Edge e intersects edge f (Fig. 8c).
(3) Edge e is completely to the right of edge f (Fig. 8e).

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

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

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

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

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

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

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

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

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

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

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

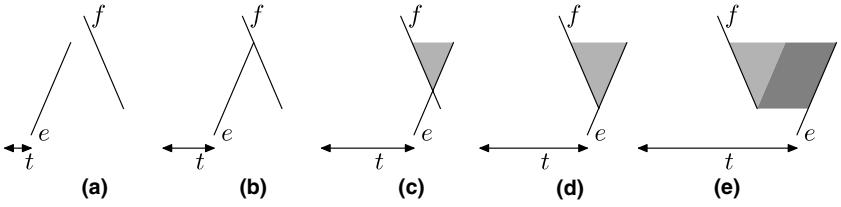

Fig. 8. The edge region $R(e,f)$ of two edges as e is translated t units from left to right. (a) e is completely left of f. (b) e adjoins f. (c) e crosses f. (d) e and f are adjoined again. (e) e is to the right of f. Notice that $R(e,f)$ is either $\emptyset$, a triangle or a triangle combined with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

6.2.2. Determining the minimum overlap translation

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

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

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

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

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

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

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

Algorithm 2. Determine Horizontal Translation with Minimal Overlap

```
Given a set 𝒮 of polygons and a polygon P ∉ 𝒮.
for each edge e from polygons 𝒮 \ {P}
   for each edge f from P
      Create breakpoints for edge pair (e,f).
Let B = breakpoints sorted.
Define area-function α(t) = Ã t² + B̃ t + C̃.
Set Ã = B̃ = C̃ = 0.
for each breakpoint b ∈ B
   Modify α(t) by changing Ã, B̃ and C̃.
   Look for minimum on the next interval of α(t).
return t with smallest α(t)
```

7. Problem variations

The solution method presented in the previous sections can also be applied to a range of variations of nesting problems. Two of the most interesting are discussed in the following subsections. More details and other variations are described by Nielsen and Odgaard (2003).

7.1. Rotation

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

It has been shown by Nielsen and Odgaard (2003) that a rotational variant of the Intersection Area Theorem is also possible. They also showed how to calculate the breakpoints needed for an iterative algorithm. It is an open question though whether an efficient iterative algorithm can be constructed.

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

7.2. Quality regions

In e.g. the leather industry the raw material can be divided into regions of quality (Heistermann

and Lengauer, 1995). Some polygons may be required to be of specific quality and should therefore be confined to these regions. This is easily dealt with by representing each region by a polygon and mark each region-polygon with a positive value describing its quality. Now if an element is required to be of a specific quality, region-polygons of poorer quality are included during overlap calculation with the element, thus disallowing placements with the element within a region with less-than-required quality. Note that the complexity of the translation algorithm is not affected by the number of quality levels.

8. Results

The solution method described in the previous sections has been implemented in C++, and we call the implementation 2DNEST. A good description of the data instances used can be found in Gomes and Oliveira (2006). These data instances are all available on the ESICUP homepage.[1] Some of their characteristics are included in Table 1. For some instances rotation is not allowed and for others 180° rotation or even 90° rotation is allowed. In 2DNEST this is handled by extending the neighborhood to include translations of rotated variants of the stencils. Note that the data instances Dighe1 and Dighe2 are jigsaw puzzles for which other solution methods would clearly be more efficient, but it is still interesting to see how well they are handled since we know the optimal solution.

Most of the data instances have frequently been used in the literature, but with regard to quality the best results are reported by Gomes and Oliveira (2006). They also report average results found when doing 20 runs for each instance. Gomes and Oliveira implemented two variations of their solution method (GLSHA and SAHA) and results for the latter can be found in Table 1 (2.4 GHz Pentium 4). Average computation times are included in the table since they vary between instances. More precisely they vary between 22 seconds and 173 minutes. When considering all instances the average computation time is more than 74 minutes. In total it would have taken more than 15 days to do all of the experiments on a single processor.

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

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

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

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

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

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

Table 1
Comparison of our implementation 2DNEST and SAHA by Gomes and Oliveira (2006)

Data instance			Average		Std. dev.		Minimum		Maximum		6 hours	Seconds
Name	Size	Deg.	2DNEST	SAHA	2DNEST	SAHA	2DNEST	SAHA	2DNEST	SAHA	2DNEST	SAHA
Albano	24	180°	86.96	84.70	0.32	1.23	86.12	83.27	87.44	87.43	87.88	2257
Dagli	30	180°	85.31	85.38	0.53	1.07	83.97	83.14	85.98	87.15	87.05	5110
Dighe1	16		93.93	82.13	5.16	3.90	86.57	74.68	99.86	100.00	99.84	83
Dighe2	10		93.11	84.17	5.42	6.84	81.81	75.73	99.95	100.00	93.02	22
Fu	12	90°	90.93	87.17	0.62	1.40	90.05	85.08	91.84	90.96	92.03	296
Jakobs1	25	90°	88.90	75.79	0.42	0.88	87.07	75.39	89.07	[b,a]78.89	89.03	332
Jakobs2	25	90°	80.28	74.66	0.18	0.89	79.53	74.23	80.41	77.28	81.07	454
Mao	20	90°	82.67	80.72	0.87	0.87	81.07	78.93	85.15	82.54	85.15	8245
Marques	24	90°	88.73	86.88	0.25	0.81	88.08	85.31	89.17	88.14	89.82	7507
Shapes0	43		65.42	63.20	0.78	0.98	64.25	61.39	67.09	66.50	66.42	3914
Shapes1	43	180°	71.74	68.63	0.79	1.41	71.12	65.41	73.84	71.25	73.23	10,314
Shapes2	28	180°	79.89	81.41	1.05	0.74	76.71	80.00	81.21	83.60	81.59	[a]2136
Shirts	99	180°	85.73	85.67	0.41	0.49	85.14	84.91	86.33	[b]86.79	87.38	10,391
Swim	48	180°	70.27	72.28	0.69	0.97	69.41	70.63	71.53	74.37	72.49	6937
Trousers	64	180°	89.29	89.02	0.28	0.57	88.77	87.74	89.84	89.96	90.46	8588

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

[a] These values have been corrected compared to those given in Gomes and Oliveira (2006).

[b] Better results were obtained by a more simple greedy approach (GLSHA) (Gomes and Oliveira, 2006): 81.67% for Jakobs1 and 86.80% for Shirts.

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

9. Three-dimensional nesting

Our fast translation method is not restricted to two dimensions. In this section we will describe how the method can be used for three-dimensional nesting, but we will not generalize the proofs from Section 6. Solutions for such problems have applications in the area of *Rapid Prototyping* (Yan and Gu, 1996), and Osogami (1998) has done a small survey of existing solution methods.

9.1. Generalization to three dimensions

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

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

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

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

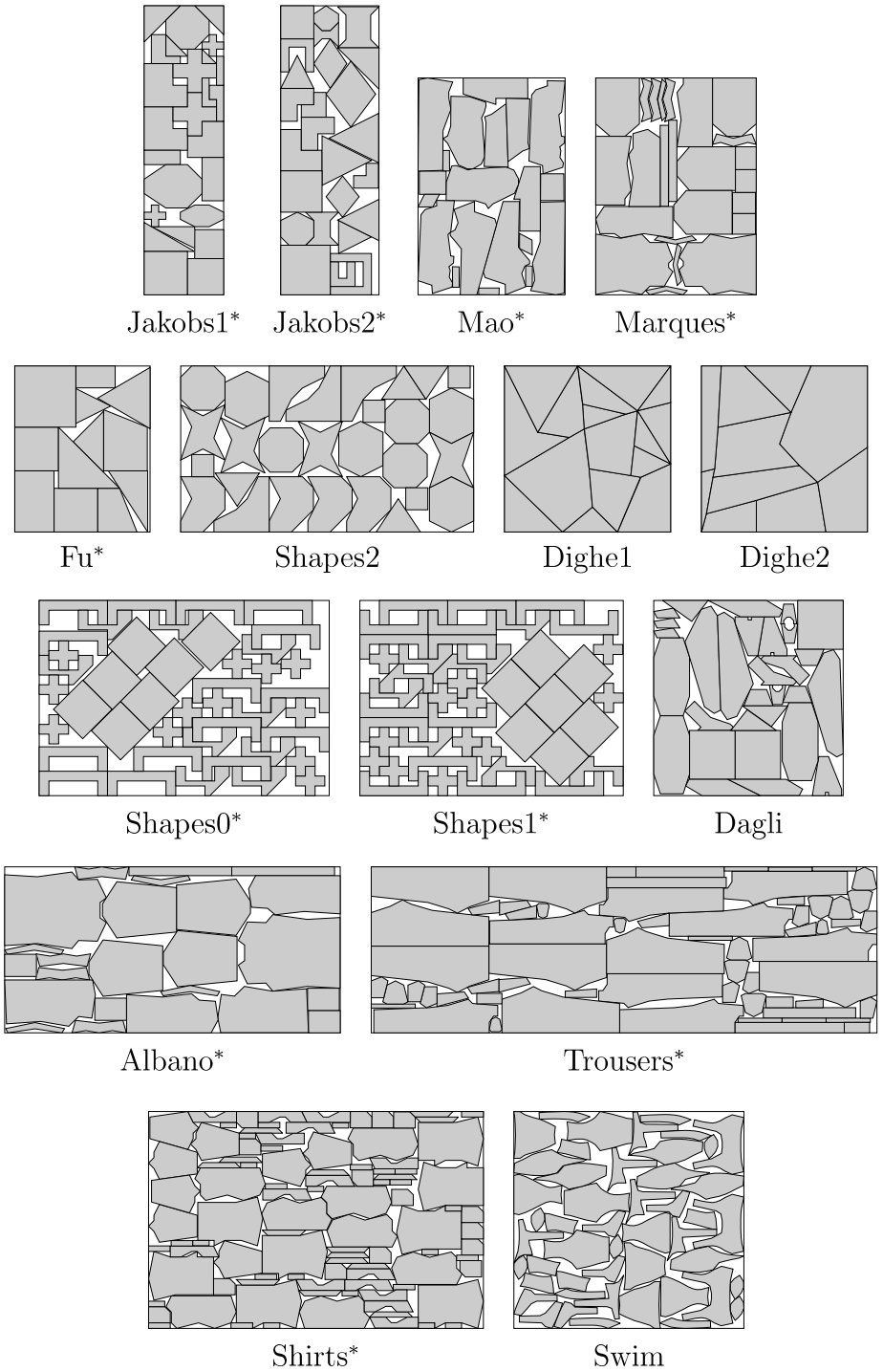

Fig. 9. The best solutions found by 2DNEST easily comparable with the ones shown in Gomes and Oliveira (2006). *These solutions are also the currently best known solutions in the literature.

We have ignored 3 special cases of pairs of corner points meeting at the same time. (1) If the faces are parallel then we can simply skip to phase 4 and use a zero constant. (2) If the two last pairs of corner points meet at the same time then we can simply skip phase 3. (3) Finally, if the first two pairs of

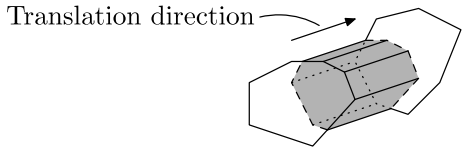

Fig. 10. An illustration of the face region between two faces. The faces are not necessarily parallel, but the sides of the face region are parallel with the translation direction. The face region would be more complicated if the two faces were intersecting.

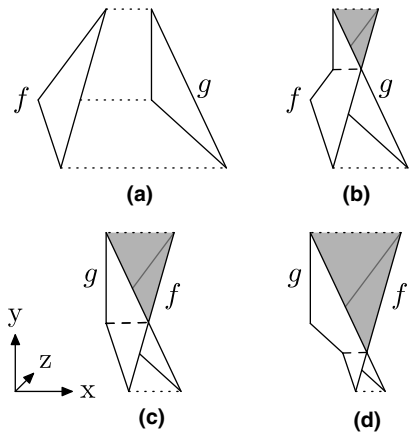

Fig. 11. Translation of a triangle f through another triangle g along the x-axis, where the triangles have the same projection onto the yz-plane. The face region $\mathbf{R}(f,g)$ changes shape each time two corner points meet.

corner points meet at the same time we can skip phase 1. The reasoning for this is simple. Fig. 11c illustrates that it is possible to cut a triangle into two parts which are easier to handle than the original triangle. The upper triangle is still a growing tetrahedron, but the lower triangle is a bit different. It is a tetrahedron growing from an edge instead of a corner and it can be calculated as a constant minus the area of a shrinking tetrahedron.

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

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

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

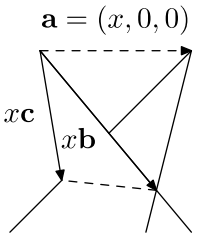

Fig. 12. The volume of the above tetrahedron can be calculated from the three vectors **a**, **b** and **c**. In our case **b** and **c** are linearly dependent on x which is the length of **a** (and the translation distance since the tetrahedron started growing).

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

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

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

$$V(x) = \frac{1}{6}|(b_yc_z - b_zc_y)(x^3 - 3tx^2 + 3t^2x - t^3)|. \tag{7}$$

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

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

9.2. Results for three dimensions

A prototype has been implemented, 3DNEST, and its performance has been compared with the very limited existing results. In the literature only one

Table 2
The Ikonen data set

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

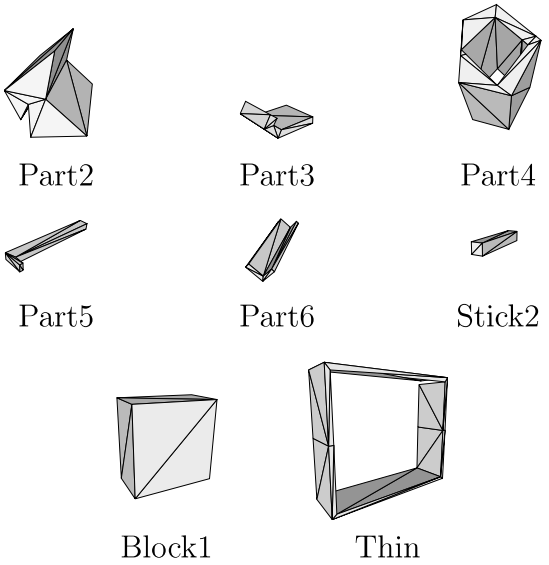

Fig. 13. The Ikonen data set.

set of simple data instances has been used. They were originally created by Ilkka Ikonen and later used by Dickinson and Knopf (1998) to compare their solution method with Ikonen et al. (1997). Eight objects are available in the set and they are presented in Table 2 and Fig. 13. Some of them have holes, but they are generally quite simple. They can all be drawn in two dimensions and then just extended in the third dimension. They have no relation to real-world data instances.

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

- Case 1
  Pack 10 objects into a cylinder of radius 3.4 and height 3.0. The 10 objects were chosen as follows: $3 \times$ Part2, 1 Part4 and $2 \times$ Part3, Part5 and Part6. Total number of faces is 260 and 11.3% of the total volume is filled.
- Case 2
  Pack 15 objects into a cylinder of radius 3.5 and height 5.5. The 15 objects were chosen as in case 1, but with 5 more Part2. Total number of faces is 380 and 12.6% of the total volume is filled.

Dickinson and Knopf report execution times for both their own solution method (serial packing) and the one by Ikonen et al. (genetic algorithm) and they

Table 3
Execution times for three different heuristic approaches

Test	Ikonen et al.	Dickinson and Knopf	3DNEST
Case 1	22.13 minutes	45.55 seconds	3.2 seconds (162 translations)
Case 2	26.00 minutes	81.65 seconds	8.1 seconds (379 translations)

Note that the number of objects is doubled for 3DNEST.

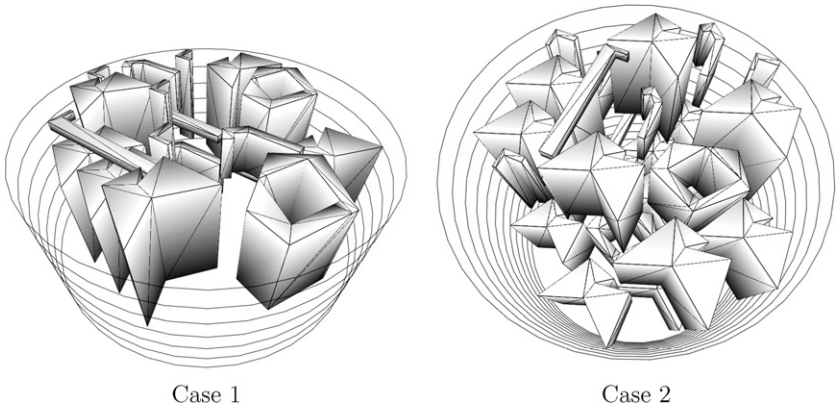

Fig. 14. The above illustrations contain twice as many objects as originally intended in Ikonens Cases 1 and 2. They only took a few seconds to find.

ran the benchmarks on a 200 MHz AMD K6 processor. The results are presented in Table 3 in which results from our algorithm are included.

Our initial placement is a random placement which could be a problem since it would quite likely contain almost no overlap and then it would not say much about our algorithm—especially the GLS part. To make the two cases a bit harder we *doubled* the number of objects. Our tests were run on a 733 MHz G4. Even considering the difference in processor speeds there is no doubt that our method is the fastest for these instances. Illustrations of the resulting placements can be seen in Fig. 14.

10. Conclusion

We have presented a new solution method for nesting problems. The solution method uses local search to reduce the amount of overlap in a greedy fashion and it uses Guided Local Search to escape local minima. To find new positions for stencils which decrease the total overlap, we have developed a new algorithm which determines a horizontal or vertical translation of a polygon with least overlap. Furthermore, our solution method can easily be extended to handle otherwise complicated requirements such as free rotation and quality regions.

The solution method has also been implemented and is in most cases able to produce better solutions than those previously published. It is also robust with very good average solutions and small standard deviations compared to previously published solutions methods, and this is within a reasonable time limit of 10 minutes per run.

Finally we have generalized the method to three dimensions which enables us to also solve three-dimensional nesting problems.

Acknowledgements

We would like to thank A. Miguel Gomes and José F. Oliveira for providing additional data on the performance of their solution method (Gomes and Oliveira, 2006). We would also like to thank Martin Zachariasen and the anonymous referees for some valuable remarks.

References

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

Albano, A., Sappupo, G., 1980. Optimal allocation of two-dimensional irregular shapes using heuristic search methods. IEEE Transactions on Systems, Man and Cybernetics 5, 242–248.

Art, Jr., R.C., September 1966. An approach to the two dimensional, irregular cutting stock problem. Tech. Rep. 36.Y08, IBM Cambridge Scientific Center.

Asano, T., Hernández-Barrera, A., Nandy, S.C., 2002. Translating a convex polyhedron over monotone polyhedra. Computational Geometry 23, 257–269.

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

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

Blazewicz, J., Walkowiak, R., 1995. A local search approach for two-dimensional irregular cutting. OR Spektrum 17, 93–98.

Blazewicz, J., Hawryluk, P., Walkowiak, R., 1993. Using a tabu search approach for solving the two-dimensional irregular cutting problem. Annals of Operations Research 41, 313–325.

Burke, E.K., Kendall, G., 1999a. Applying ant algorithms and the no fit polygon to the nesting problem. In: Proceedings of the 12th Australian Joint Conference on Artificial Intelligence (AI'99). Springer Lecture Notes in Artificial Intelligence, vol. 1747, pp. 454–464.

Burke, E.K., Kendall, G., 1999b. Applying evolutionary algorithms and the no fit polygon to the nesting problem. In: Proceedings of the 1999 International Conference on Artificial Intelligence (IC-AI'99), vol. 1. CSREA Press, pp. 51–57.

Burke, E.K., Kendall, G., 1999c. Applying simulated annealing and the no fit polygon to the nesting problem. In: Proceedings of the World Manufacturing Congress. ICSC Academic Press, pp. 70–76.

Chen, P., Fu, Z., Lim, A., Rodrigues, B., 2004. The two dimensional packing problem for irregular objects. International Journal on Artificial Intelligent Tools.

Dickinson, J.K., Knopf, G.K., 1998. Serial packing of arbitrary 3d objects for optimizing layered manufacturing. In: Intelligent Robots and Computer Vision XVII, vol. 3522. pp. 130–138.

Dobkin, D., Hershberger, J., Kirkpatrick, D., Suri, S., 1993. Computing the intersection-depth of polyhedra. Algorithmica 9, 518–533.

Dowsland, K.A., Dowsland, W.B., 1995. Solution approaches to irregular nesting problems. European Journal of Operational Research 84, 506–521.

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

Dowsland, K.A., Vaid, S., Dowsland, W.B., 2002. An algorithm for polygon placement using a bottom-left strategy. European Journal of Operational Research 141, 371–381.

Faroe, O., Pisinger, D., Zachariasen, M., 2003. Guided local search for the three-dimensional bin packing problem. INFORMS Journal on Computing 15 (3), 267–283.

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

Gomes, A.M., Oliveira, J.F., 2006. Solving irregular strip packing problems by hybridising simulated annealing and linear

programming. European Journal of Operational Research 171, 811–829.

Heckmann, R., Lengauer, T., 1995. A simulated annealing approach to the nesting problem in the textile manufacturing industry. Annals of Operations Research 57, 103–133.

Heistermann, J., Lengauer, T., 1995. The nesting problem in the leather manufacturing industry. Annals of Operations Research 57, 147–173.

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

Jain, P., Fenyes, P., Richter, R., 1992. Optimal blank nesting using simulated annealing. Journal of Mechanical Design 114, 160–165.

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

Li, Z., Milenkovic, V., 1995. Compaction and separation algorithms for non-convex polygons and their applications. European Journal of Operational Research 84, 539–561.

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

Nielsen, B.K., Odgaard, A., 2003. Fast neighborhood search for the nesting problem. Tech. Rep. 03/03, DIKU, Department of Computer Science, University of Copenhagen.

Oliveira, J.F., Ferreira, J.S., 1993. Algorithms for nesting problems. Applied Simulated Annealing, 255–273.

Oliveira, J.F., Gomes, A.M., Ferreira, J.S., 2000. TOPOS—a new constructive algorithm for nesting problems. OR Spektrum 22, 263–284.

Osogami, T., 1998. Approaches to 3D free-form cutting and packing problems and their applications: A survey. Tech. Rep. RT0287, IBM Research, Tokyo Research Laboratory.

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

Theodoracatos, V.E., Grimsley, J.L., 1995. The optimal packing of arbitrarily-shaped polygons using simulated annealing and polynomial-time cooling schedules. Computer Methods in Applied Mechanics and Engineering 125, 53–70.

Voudouris, C., Tsang, E., August 1995. Guided local search. Tech. Rep. CSM-147, Department of Computer Science, University of Essex, Colchester, C04 3SQ, UK.

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

Wäscher, G., Haussner, H., Schumann, H., 2006. An improved typology of cutting and packing problems. European Journal of Operational Research, this issue, doi:10.1016/j.ejor.2005.12.047.

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