Available online at www.sciencedirect.com
ScienceDirect
European Journal of Operational Research 179 (2007) 27–49
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

Discrete Optimization

Complete and robust no-fit polygon generation for the irregular stock cutting problem

E.K. Burke, R.S.R. Hellier, G. Kendall, G. Whitwell *

University of Nottingham, School of Computer Science & IT, Jubilee Campus, Nottingham NG8 1BB, UK

Received 22 April 2005; accepted 8 March 2006
Available online 27 April 2006

Abstract

The no-fit polygon is a construct that can be used between pairs of shapes for fast and efficient handling of geometry within irregular two-dimensional stock cutting problems. Previously, the no-fit polygon (NFP) has not been widely applied because of the perception that it is difficult to implement and because of the lack of generic approaches that can cope with all problem cases without specific case-by-case handling. This paper introduces a robust orbital method for the creation of no-fit polygons which does not suffer from the typical problem cases found in the other approaches from the literature. Furthermore, the algorithm only involves two simple geometric stages so it is easily understood and implemented. We demonstrate how the approach handles known degenerate cases such as holes, interlocking concavities and jigsaw type pieces and we give generation times for 32 irregular packing benchmark problems from the literature, including real world datasets, to allow further comparison with existing and future approaches.

Keywords: Cutting; Packing; No-fit polygon; Orbital approach

1. Introduction

The irregular two-dimensional variant of the cutting and packing problem impacts upon several important manufacturing industries such as textiles, plastics, metal cutting and others. These problems usually consist of a number of irregular pieces that are to be placed onto one or more sheets of material in the most efficient layout possible, so that all pieces are assigned and do not overlap. Additionally, there are usually rotational constraints enforced on the pieces due to the physical properties of the problem such as grain on the material, patterns on textiles and the cutting technology being employed. Sometimes rotational constraints may be used for non-physical reasons such as to restrict pieces to a finite set of rotations thus simplifying layout construction procedures and allowing for faster solutions to be obtained. The two-dimensional stock cutting problem

* Corresponding author. Tel.: +44 115 9514210.
E-mail addresses: ekb@cs.nott.ac.uk (E.K. Burke), rsh@cs.nott.ac.uk (R.S.R. Hellier), gxk@cs.nott.ac.uk (G. Kendall), gxw@cs.nott.ac.uk (G. Whitwell).

0377-2217/$ - see front matter
doi:10.1016/j.ejor.2006.03.011

has been shown to be NP hard and is therefore intrinsically difficult to solve (Garey and Johnson, 1979). There have been many different strategies for producing solutions to the irregular stock cutting problem. These include linear programming approaches, heuristic placement methods, metaheuristic guided search techniques and other novel approaches such as the iterative jostling of pieces (Dowsland et al., 1998). Survey papers can be found in (Dowsland and Dowsland, 1992; Sweeney and Paternoster, 1992; Dyckhoff, 1990). However, the feature that connects all of the approaches is that they are all required to cope with the geometry of the problem. This can be especially complicated when highly irregular shapes are used which may include holes or concavities. The implementation of robust and efficient geometry routines can be laborious and can often take considerably longer than the packing strategies themselves. In particular, the geometry must handle all of the interactions between shapes such as detecting whether two shapes are overlapping and calculating the translation distance required in a given direction so that the overlap is resolved. As we show in the following section, although these tests can be implemented using trigonometric techniques, the no-fit polygon represents a considerably more efficient solution.

Whilst the generation of the no-fit polygon is academically challenging, it is a 'tool' and not a 'solution' and this is perhaps one of the reasons why there are many publications in the literature which state that the no-fit polygon is used but which provide relatively little or no details on its implementation. In this paper we concentrate specifically on the no-fit polygon and provide an overview of the previous techniques that have been used for its creation. Furthermore, we describe and provide full implementation details for a new robust orbital approach that can cope with the traditional problem cases that many other approaches cannot handle. Hopefully this will help in the further dissemination of the benefits of using the no-fit polygon (as opposed to traditional trigonometry based approaches) within both the industrial and the academic communities.

2. The no-fit polygon – an overview

In this section we describe the functionality of the no-fit polygon and we compare it to the more traditional trigonometric based overlap and intersection tests. We also give brief overviews of the many techniques that have been used to generate no-fit polygons within the previous literature.

2.1. The no-fit polygon

The first application of no-fit polygon techniques within the field of cutting and packing was presented by Art (1966), although the term "shape envelop" was used. It was ten years later that the term "no-fit polygon" was introduced by Adamowicz and Albano who approached the irregular stock cutting problem by using no-fit polygons to pack shapes together using their minimum enclosing rectangles (Adamowicz and Albano, 1976). The term "configuration space obstacle" is often used to denote the NFP within the field of engineering and robot motion planning but the term has also been used with respect to cutting and packing in (Cuninghame-Green, 1989). The "hodograph" is often used to describe the no-fit polygon within the mathematics community (Stoyan and Ponomarenko, 1977; Scheithauer and Terno, 1993; Bennell et al., 2001). The main function of the no-fit polygon is to describe the region in which two polygons intersect. The following example gives an overview of the NFP construct.

Given two polygons, A and B, the no-fit polygon can be found by tracing one shape around the boundary of another. One of the polygons remains fixed in position and the other traverses around the fixed polygon's edges whilst ensuring that the polygons always touch but never intersect. Throughout this paper we adopt the convention of the first polygon being fixed and the second being the traversing/orbiting polygon. Therefore, when polygon B traces around the fixed polygon A, the resulting no-fit polygon is denoted by NFP_{AB}. In order to create the NFP_{AB} object we must choose a reference point from B which will be traced as B moves around A. In our implementations we use the first vertex, within the shape vertex list, as the reference point (see Fig. 1). The reference point can be any arbitrary point providing it follows the movements of the orbiting polygon. It is also important to maintain the relative position of the reference point with respect to polygon B as this is required when using the NFP to test for overlap.

In order to test whether polygon B overlaps polygon A we use NFP_{AB} and B's reference point. If polygon B is positioned so that its reference point is inside the polygon NFP_{AB} then it overlaps with polygon A. If the

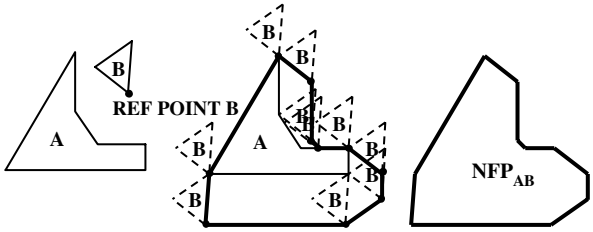


Fig. 1. The no-fit polygon of two shapes A and B.

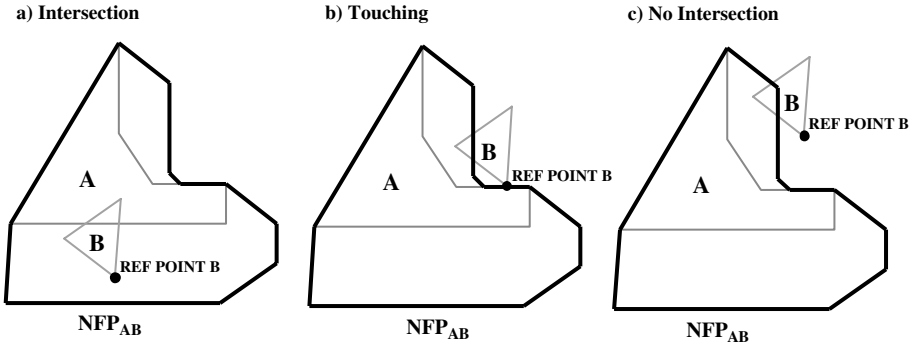


Fig. 2. Using the no-fit polygon to test for intersection between polygons A and B.

reference point is on the boundary of NFP$_{AB}$ then polygon B touches polygon A. Finally, if the reference point is outside of NFP$_{AB}$ then polygons A and B do not overlap or touch (see Fig. 2). The no-fit polygon is used within the following papers from the literature (Grinde and Cavalier, 1995; Ramkumar, 1996; Cheng and Rao, 1997; Dowsland et al., 1998, 2002; Milenkovic, 1999; Gomes and Oliveira, 2002).

2.2. No-fit polygon vs. standard trigonometry overlap detection

Although the no-fit polygon is an excellent tool for conducting intersection tests between pairs of polygons, it has not been widely applied for two-dimensional packing problems in both the literature and in real world manufacturing industries. Undoubtedly this is due to the no-fit polygon's complex implementation and the lack of available robust algorithms. Instead, many intersection implementations use standard trigonometrical approaches which especially occur in the case of packing software for real-world applications where it is important that distributed software is able to handle all possible polygons without errors. However, whilst both approaches have the same overall effect, use of the no-fit polygon can be several times quicker than even the most efficient trigonometrical routines. For example, where we wish to attempt numerous iterations of the same layout problem, the pre-generation of no-fit polygons can significantly reduce the total computation time. Over numerous iterations, trigonometric approaches are likely to repeatedly detect and resolve the same overlapping shapes in repeated orientations and positions. Where a nesting overlap resolution approach requires the calculation of all intersection points between two intersecting shapes the benefits of a no-fit polygon approach are multiplied further as numerous intersection calculations are required for the full detection of collision when the no-fit polygon method is not used. Therefore, by utilising no-fit polygons, we can reduce the overlap detection problem (which is a major factor in the computational overhead of the nesting process) to a significantly less expensive *point inside polygon* test (Dowsland et al., 2002). In addition to this, when utilising the overlap resolution technique used in Burke et al. (in press), where intersecting shapes are resolved through the repeated resolution of intersecting edges in the y-axis direction, by utilising the no-fit polygon the resolution technique is more efficient, resolving the y-axis overlap in one complete movement. Furthermore, the no-fit polygon would also allow for the overlap to be resolved in *any direction* by casting out a ray from the rel-

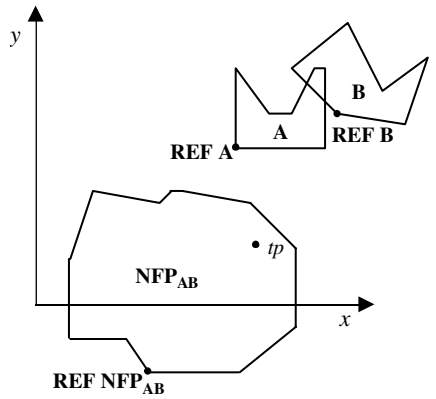


Fig. 3. Intersection testing with the no-fit polygon.

evant reference point and finding the nearest intersection with the no-fit polygon boundary. In the following section, we compare the geometrical procedures that are required to detect and resolve the overlap between two polygons using, firstly, standard trigonometry and, secondly, using the NFP. Given a nesting method where we wish to detect all points of intersection for polygons A and B (see Fig. 3), every edge is tested against every other edge leading to 42 tests to determine intersection status (given that polygon A has seven edges and polygon B has six edges). Furthermore, if we find no intersection, in order to eliminate the possibility of the polygons intersecting through their vertices, or one polygon completely containing the other, we must perform a point inside test for all points on both shapes, requiring a further 13 tests and therefore, in the worse case, a total of 55 tests.

These can be regarded as a large but unavoidable overhead for the detection of intersection between any two polygons in a layout through trigonometric methods, which can only increase as we add an increasing number of polygons to our problem. Whilst many searches can be attempted and fast intersection libraries can be developed, intersection detection remains a considerable portion of the computational overhead inherent in the generation of packing solutions. Additionally where we wish to develop numerous solutions using a search method these, often repeated, calculations will have considerable impact on the overall computation time.

Now returning to Fig. 3, the no fit polygon of the two polygons, NFP_{AB}, is also shown at some arbitrary position in the problem space. In this case, the NFP_{AB} has been generated using the proposed edge sliding technique where polygon B traversed the edges on the polygon A and the NFP was formed by tracking the reference point, REF B, of the traversing polygon. The coordinate at which the NFP has been created within the Cartesian space does not affect the use of NFP based intersection detection because we can simply generate the correct test point (tp) with respect to the NFP's position using a simple calculation based on reference points: $\mathrm{tp} = \mathrm{REF}\ \mathrm{NFP}_{AB} + \mathrm{REF}\ B - \mathrm{REF}\ A - \mathrm{Offset}$, where Offset is the vector from the reference point of polygon A (the stationary polygon) to the reference point of the no-fit polygon ($\mathrm{REF}\ \mathrm{NFP}_{AB} - \mathrm{REF}\ A$). The status of tp with respect to NFP_{AB} can be calculated using a ray-crossing algorithm described in O'Rourke (1998). If the point tp is found to be within NFP_{AB} then the polygons A and B are colliding with one another. Colliding includes both intersection and containment of one polygon by another. If tp falls on the NFP_{AB} then the polygons are known to be touching and if tp falls outside of NFP_{AB} then the shapes are neither touching nor colliding. By calculating all of the no-fit polygons for all pairs of polygons, we can save considerable computation time when performing multiple iteration nesting.

2.3. Approaches for no-fit polygon construction

Here we review the main techniques that have previously been used for construction of no-fit polygons within the literature. For each method we give a brief overview of the approach and discuss any benefits and drawbacks that may occur from its usage.

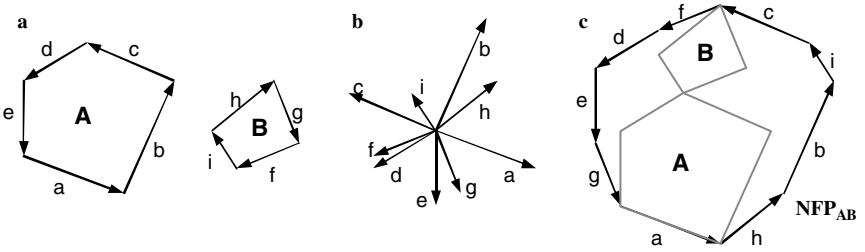


Fig. 4. No-fit polygon generation with convex shapes.

2.3.1. Convex shapes

The basic form of no-fit polygon generation occurs when both polygons are convex. Given two convex shapes, A and B, the no-fit polygon is created by the following steps: (i) Orientate shape A anticlockwise and shape B clockwise (see Fig. 4a), (ii) Translate all edges from A and B to a single point (see Fig. 4b), Concatenate these edges in anticlockwise order to yield the no-fit polygon (see Fig. 4c).

Cuninghame-Green uses this approach to produce "configuration space obstacles" between pairs of convex polygons which are then used for intersection tests during the packing of shapes (Cuninghame-Green, 1989). For instances involving non-convex pieces, Cuninghame-Green firstly calculates the convex hull of each non-convex shape (the minimal containing convex polygon) and then calculates no-fit polygons using the respective convex hulls.

The benefit of convex no-fit polygon generation is that it is simple and is extremely quick using a standard sorting algorithm in combination with edge reordering through translation. The obvious disadvantage is that no-fit polygons cannot be generated for non-convex shapes and the reduction to convex hulls results in concavity sections being unavailable in packing and non-traversable in robot motion planning. As this can adversely affect solution quality, other approaches are required.

2.3.2. Non-convex shapes

There have been many different approaches to producing no-fit polygons from non-convex shapes. These can be placed into three general categories: decomposition, Minkowski sums and orbital approaches.

2.3.2.1. Decomposition. The first approach is to decompose any non-convex shapes into sub-pieces that can be 'managed' more easily. However, decomposition usually results in several sub-pieces and therefore several no-fit polygons must be created. In order to conduct intersection testing, the no-fit polygons can remain decomposed or they can be recombined through union operations. Where possible, a single no-fit polygon offers the fastest intersection computation times but will require additional calculations to recombine constituent parts. This can be a computationally expensive and a difficult undertaking if several subparts are present and this may be further complicated if holes are present. For example, Agarwal et al. (2002) conducted an extensive investigation into different decomposition and recombination operations with respect to constructing Minkowski sums of non-convex polygons. They conclude that it is counterproductive to use optimal decompositions because the computation times to calculate them outweigh the benefits achieved during recombination. The authors report that the recombination operations are the most costly and give example execution times that range from a few seconds for shapes involving a small amount of concavities and up to twenty minutes for highly irregular shapes. Avnaim and Boissonnat (1988) discuss a decomposition recombination approach based on linear segments. These are used to produce parallelograms which are recombined to yield the NFP. The authors provide a mathematical proof to show that the approach can handle the general case and further extend this to allow rotations. The algorithm has a time complexity of $O(m^3n^3 \log mn)$.

2.3.2.1.1. Convex pieces. We have already discussed that no-fit polygon generation is trivial when convex shapes are involved. If shapes with concavities can be divided into convex pieces, fast convex no-fit polygon generation techniques can be employed. The simplification of geometrical intersection through the decomposition of non-convex shapes to convex pieces was suggested and discussed in (Avnaim and Boissonnat, 1987; Cuninghame-Green, 1992). The main difficulties with such an approach are the decomposition and

recombination algorithms. There are many well-known approaches that can be used for this including decompositions into triangular or convex pieces. Seidal suggests a fast polygon triangulation algorithm which has complexity of an O(n log n) complexity (Seidel, 1991). Further implementation details for triangulation can be found in (Amenta, 1997). However, for our purposes, the triangulation approaches produce more subpieces than is necessary and will ultimately impact upon computation time within the generation process. Unlike triangulation, which can be seen as a specialised cased of convex decomposition, generalised convex decomposition algorithms aim to represent polygons with as few convex pieces. The two key approaches are suboptimal decompositions, which has an O(n) complexity (Hertel and Mehlhorn, 1983), and optimal decomposition which has an O(n^3) complexity (Chazelle and Dobkin, 1985). Agarwal et al. (2002) state that it is generally more efficient to use suboptimal decompositions because of the computational overhead inherent with creating optimal decompositions but they also suggest that an alternative and possibly more efficient approach is to perform convex covering instead. Once any irregular polygons have been decomposed into convex pieces, the no-fit polygon may be generated by passing each convex piece of shape B around each convex piece of shape A. The disadvantage with this approach is that the no-fit polygons of the convex pieces may cross and care must be taken when recombining them to construct the no-fit polygon. Particular difficulty occurs if the original shapes contain holes as it is unclear whether intersecting no-fit polygon subsections define holes or regions that can be discarded.

2.3.2.1.2. Star-shaped polygons. Li and Milenkovic decompose shapes into convex and star-shaped polygons. A star-shaped polygon has the property that there exists at least one internal point, or 'kernel point', that can 'see' the entire boundary of the polygon. By extending the concavity edges and elimination of invisible regions it is evident that the star-shaped polygon has a region, R_{kernel}, that can be defined within which a kernel point can be placed to see the entire polygon boundary. Conversely, this is not the case with the non-star-shaped polygon because no region can be defined in which to place a kernel. Thus, star-shaped polygons are situated somewhere between convex and non-convex shapes in terms of generality. Li and Milenkovic state that star-shaped polygons are 'closed' under Minkowski sum operations and provide a proof that the Minkowski sum of two star-shaped polygons also yields a star-shaped polygon (Li and Milenkovic, 1995). The authors do not state whether they recombine the no-fit polygon regions into one no-fit polygon entity or whether they perform multiple no-fit polygon intersection tests during the layout generation stage.

2.3.2.1.3. Phi-functions. An alternative and promising approach is presented by Stoyan et al. and is based on the use of ''phi-functions'' (Stoyan et al., 2001). Phi-functions define mathematical intersection relationships between pairs of standard or ''primary'' objects such as rectangles, circles and convex polygons. Although phi-functions are not strictly a no-fit polygon based approach, it is included because it has had very promising results. The authors further develop their work to enable the definition of mathematical intersection relationships for non-convex polygons through the union, intersection and complement of primary objects (Stoyan et al., 2002). The resultant intersection test between two shapes during layout generation is performed through comparisons of phi-functions between all pairs of the primary objects that define shape A and shape B. Phi functions have a secondary benefit in that they are based on distance functions between primary objects and therefore can easily be used to find the distance between two complex objects.

2.3.2.2. Minkowski sums. The no-fit polygon construct can be unified through the use of Minkowski sums (a form of vector addition) and was first suggested by (Stoyan and Ponomarenko, 1977). The concept is as follows: given two arbitrary point sets, A and B, the Minkowski sum of A and B is defined by the following: $A \oplus B = \{a + b : a \in A, b \in B\}$.

In order to produce no-fit polygons, we must use the Minkowski difference, $A \oplus -B$. This corresponds to two input polygons in opposing orientations and is easily shown through simple vector algebra (Bennell et al., 2001). We can verify that this is the case using our simple convex only case (Section 2.3.1) where we place shape A in its anticlockwise orientation and shape B in its clockwise orientation. We can state that the method of Cuninghame-Green, involving convex shapes only, is *also* using the Minkowski difference in its most basic form.

Whilst non-mathematical implementation details of such approaches are scarce, Ghosh and, subsequently, Bennell provide excellent explanations and implementation details for no-fit region calculation. As such, we shall refer the reader to their contributions (Ghosh, 1991, 1993; Bennell et al., 2001). The main drawbacks

of the approach is that it only works providing that the concavities of the two shapes do not interfere or interlock (Ghosh, 1991, 1993). Bennell et al. (2001) state that Ghosh's approach would cause a "cumbersome tangle of intersecting edges" which would be difficult to recombine to form the no-fit polygon. They introduce a implementation that reduces the amount of 'tangled edges' and give implementation details of the process. They also report fast generation times of around 0.3 seconds for each of five separate datasets from the literature. However, they state that their approach cannot deal with internal holes as it is difficult to detect which of the internal no-fit polygon edges can be discarded and which form the internal no-fit regions.

2.3.2.3. Orbital sliding approach. The orbital sliding approach is the main focus of this paper and involves using standard trigonometry to physically slide one polygon around another. The no-fit polygon is defined by tracing the motion of a locus point of the sliding polygon when orbiting. The first discussion and implementation of an orbiting approach for the generation of 'envelopes' is detailed in Mahadevan's Ph.D. thesis (Mahadevan, 1984). The key elements of Mahadevan's approach are: calculation of touching vertices and edges, determination of the translation vector and calculation of the translation length. The calculation of intersecting edges is performed using the notion of D-functions (Konopasek, 1981). Mahadevan modifies the D-function test to also calculate touching points which is a necessity for both Mahadevan's algorithm and our new approach. This information is then used to select a translation vector based on the touching edge. The translation vector is then projected through each vertex of the orbiting shape and then the intersecting edge testing is used to calculate the translation distance. It is also important to project the translation vector in the reverse direction of the stationary polygon. The orbiting shape is then translated along the translation vector by the smallest distance (from projection and intersection tests). This ensures the two polygons never intersect but always touch. The process continues until the orbiting polygon returns to its original starting position. We will not describe Mahadevan's approach in greater detail here because many of its features are also used within our new orbital approach (Section 3) and, where applicable, we compare against Mahadevan's approach and state the improvements within our new approach. The major disadvantage with Mahadevan's algorithm is that it cannot generate the full no-fit polygon for shapes involving holes or some concavities. The problem occurs when the orbiting polygon can be placed inside a concavity of the stationary polygon but the concavity has a narrow entrance. In this case, the orbiting polygon is too wide and will simply slide over the concavity. In Section 4 we show how our new orbital approach can generate no-fit polygons for all of the known degenerate cases.

3. The new no-fit polygon construction algorithm

We now describe our new approach for robust no-fit polygon generation through the use of orbital methods utilising standard trigonometrical techniques. For the intersection calculations within our presented implementation, it is important to use a robust geometry library. It is also beneficial to implement routines that are as fast and as close to optimal as possible in order to promote fast no-fit polygon generation and packing algorithms. We have implemented such a library. Although this can be very time-consuming and a difficult undertaking, a discussion of the required algorithms can be found within most standard geometry and computer graphics texts such as (O'Rourke, 1998) and several computational geometry forums on the internet. For practitioners who do not wish to develop such a library, the geometry modules of the LEDA and CGAL libraries may provide better alternatives. It is beyond the scope of this paper to discuss each geometry routine that is used throughout the implementation.

3.1. Overview

Our approach can be divided into two logical stages. Section 3.2 describes the procedure for the first of these where one polygon slides around another to create the outer path of the no-fit polygon of the two shapes. This part follows an approach that is logically similar to that of Mahadevan's algorithm, although a modified implementation is proposed. The second section introduces the notion and identification of starting positions that allow the algorithm to find the remaining paths of the no-fit polygon (see Section 3.3). These paths will be internal holes of the no-fit polygon (i.e. contained by the outer path) and are not found using Mahadevan's

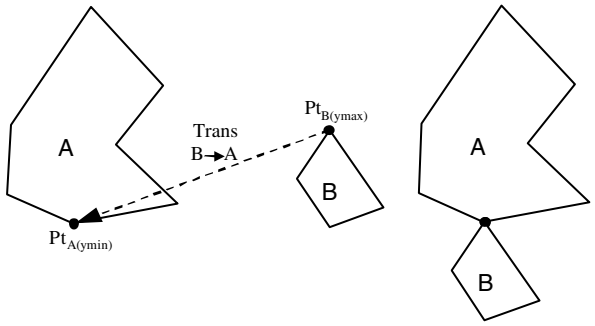


Fig. 5. Initial translation of the orbiting polygon to touch the stationary polygon.

algorithm alone. For this section we will assume that we have two anticlockwise oriented polygons, A and B, that exist somewhere within two-dimensional space and we are required to produce the no-fit polygon NFP_{AB} (orbiting polygon B around polygon A). The first operation that must be performed is to translate polygon B so that it is touching, but not intersecting, polygon A. We maintain the same approach as used by Mahadevan whereby polygon B is translated so that its largest y-coordinate is placed at the lowest y-coordinate of the polygon, A (see Fig. 5).

Using these two vertices for alignment guarantees that A and B will be non-intersecting and touching. The translation that results in polygon B touching polygon A is shown in the formula (1):

$$\text{Trans } B \rightarrow A = \mathrm{Pt}_{A(y\min)} - \mathrm{Pt}_{B(y\max)}. \tag{1}$$

In reality, any starting position may be used providing that polygons A and B are touching and non-intersecting. The orbital approach can now commence to generate the outer path of the no-fit polygon in an anticlockwise direction.

3.2. Orbiting/sliding

The main aim of the orbiting (or sliding) section of the algorithm is to detect the correct movements that B must make to traverse around A in order to return to its original position. This is an iterative procedure with each translation step creating an edge of the no-fit polygon. This can be further broken down into the following subparts which will be discussed in turn: detection of touching edges, creation of potential translation vectors, finding a feasible translation, trimming the feasible translation and, finally, applying the feasible translation.

3.2.1. Detection of touching edges

The ability to correctly detect touching and intersecting edges is paramount to the success of our approach. This is achieved by testing each edge of polygon A against each edge of polygon B. Each pair of edges that touch (one from polygon A and one from polygon B) is stored along with the position of the touching vertex. Fig. 6 shows the resulting set of touching edge pairs. This is in contrast to the approach of Mahadevan who performs calculations using all edges at a touching vertex. For example, in Fig. 6, Mahadevan would present edges $a2$, $a3$, $b1$ and $b4$ on the same diagram and, in our experience, this makes the required calculations long-winded and difficult to explain.

3.2.2. Creation of potential translation vectors

The vector with which polygon B must be translated to orbit polygon A must either be derived from an edge of polygon A or from polygon B depending on the situation. Fig. 7 shows an example of each case.

Note that in the second case (Fig. 7b), because polygon B slides along one of its own edges, the translation vector is found by reversing the edge. This is further examined through the relative movement of polygon A with respect to B. Polygon A is relatively moved along the edge b_1. In reality, polygon A must remain fixed and B must be translated and therefore, the translation vector is reversed in this situation. We can obtain the set of

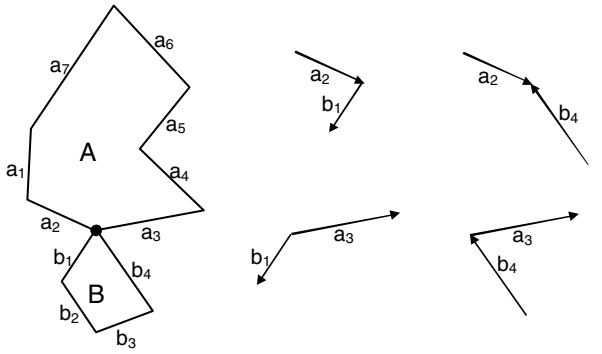


Fig. 6. Identification of touching edge pairs.

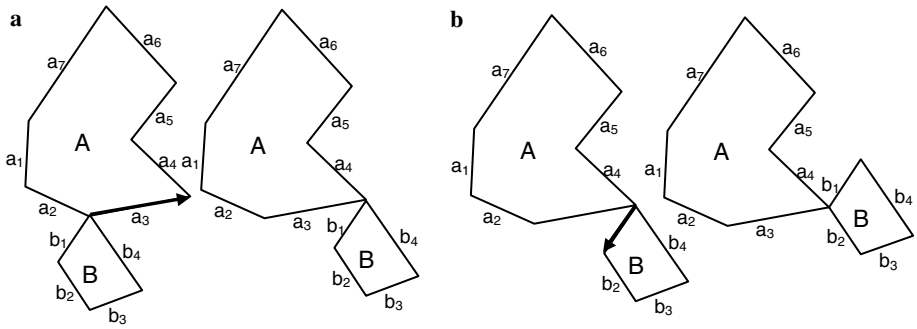


Fig. 7. Translation vector: (a) derived from edge a_3 and (b) derived from edge b_1.

potential translation vectors by using the touching edge pairs. There are three possibilities: (i) both edges touch at a vertex, (ii) a vertex of orbiting edge touches the middle of the stationary edge, or (iii) a vertex of the stationary edge touches the middle of the orbiting edge. These cases are depicted in Fig. 8.

Each pair of touching edges yields one potential translation vector. In case (ii), the translation vector is simply defined using the point at which the two edges touch and the stationary edge's end vertex. In case (iii), we use a similar process except we use the end vertex of the orbiting edge and we also reverse the vector direction. Case (i) requires the correct identification of whether the potential translation vector is derived from the stationary or orbiting edge. This can be identified by the following set of rules based on the touching vertices and a test for whether the orbiting edge, b, is left or right of the stationary edge, a. Table 1 shows the different possibilities and the edge from which a potential translation vector is derived under each circumstance.

We shall now clarify why some potential translations can be eliminated in certain cases of the table (an '–' entry in the table indicates that no translation is derived). Recall that when a translation is derived from an edge, the resultant vector is defined by touching point → end vertex (and then the vector is reversed for an orbiting edge). This allows us to eliminate case 7 of the table because both the stationary and orbiting edges touch on the end vertices and this would yield a null translation vector. In cases 3 and 4, the orbiting edge touches at its end vertex thus a translation cannot be derived from the orbiting edge. In cases 5 and 6, the

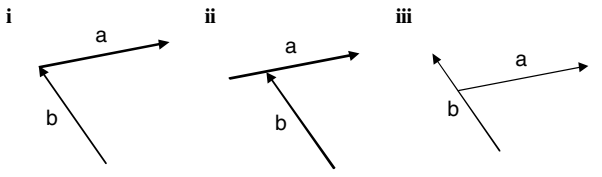


Fig. 8. Touching edge-pair types.

Table 1
Deriving the potential translation when both edges touch at vertices

Case	Touching edge vertex		Relative position of the orbiting edge to the stationary edge	Translation vector derived from
	Stationary	Orbiting		
1	Start	Start	Left	Orbiting edge
2	Start	Start	Right	Stationary edge
3	Start	End	Left	–
4	Start	End	Right	Stationary edge
5	End	Start	Left	–
6	End	Start	Right	Orbiting edge
7	End	End		–
8			Parallel	Either edge

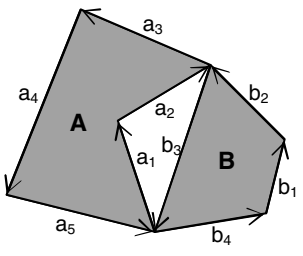
stationary edge cannot be used as it would produce a null vector. As an example, we show each of the cases using two polygons that touch at two separate places (see Fig. 9).

In cases 3 and 5, no translation can be derived because the edge of the orbiting polygon is to the left of the edge from the stationary polygon. For example, edges a_3 and b_2 touch on their start and end vertices respectively. As we produce a null translation using edge b_2 (due to touching with the end vertex), the only possibility is to derive a translation vector from the stationary edge, a_3. However, edge b_2 is left of edge a_3 and, if this translation vector were used, edge b_2 would slide along the inside of edge a_3. Therefore, the positional "left test" eliminates the translations whereby polygons would slide along the inside of an edge rather than the outside of the edge. Where edges are parallel, either the stationary or the orbiting edge may be used.

3.2.3. Finding a feasible translation vector

Once the potential translation vectors have been produced, we next select a translation vector that does not result in an immediate intersection. For example, in Fig. 9 we have generated two potential translation vectors, a_1 and $-b_3$. It can be seen that the orbiting polygon B must be translated using vector $-b_3$ in order to move anticlockwise around polygon A. If, instead, we were to translate polygon B along vector a_1, this would result in an immediate intersection between edges a_2 and b_3 (and also a_3 and b_3). Once again the set of touching edge-pairs, generated from Section 3.2.1, contains all of the information needed to determine a feasible translation vector. The process here is simplified due to our proposed representation of touching edges and involves taking each of the potential translation vectors in turn, identified in Section 3.2.2, and placing them on the touching position at each touching edge-pair. We can define positional relationships between the touching edges of the stationary and orbiting polygons based on the union of left/right regions that indicate whether a particular potential translation will be suitable for those edges. For a potential translation vector to be identified as feasible, it must be suitable for every pair of touching edges. Fig. 10 shows some examples of touching edge-pairs and the angular direction at the touch point (given by the circular arcs) for which edge b can move without intersecting with edge a.

For the sake of brevity, we have not listed all possibilities that can occur (the examples in Fig. 10 provide enough information to derive the omitted relationships). In Fig. 9, we calculated two potential translation vectors, $-b_3$ and a_1, for two touching polygons. Fig. 11 demonstrates how the approach eliminates translation



Case	Edges	Translation Vector Derived From
1	(a_3,b_3)	b_3
2	(a_1,b_4)	a_1
3	(a_3,b_2)	-
4	(a_1,b_3)	a_1
5	(a_5,b_4)	-
6	(a_2,b_3)	b_3
7	(a_2,b_2), (a_5,b_3)	-

Fig. 9. Two polygons touching at two separate positions and vertex–vertex touch cases.

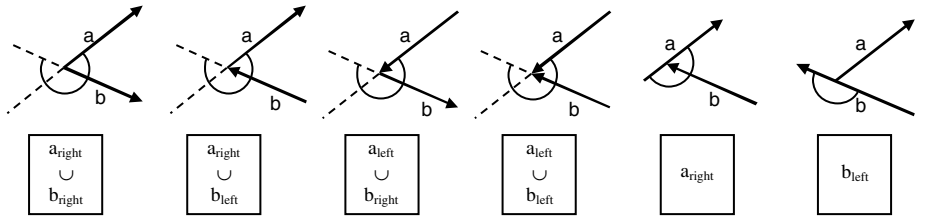


Fig. 10. Identifying the feasible angular range of translations (indicated by an arc).

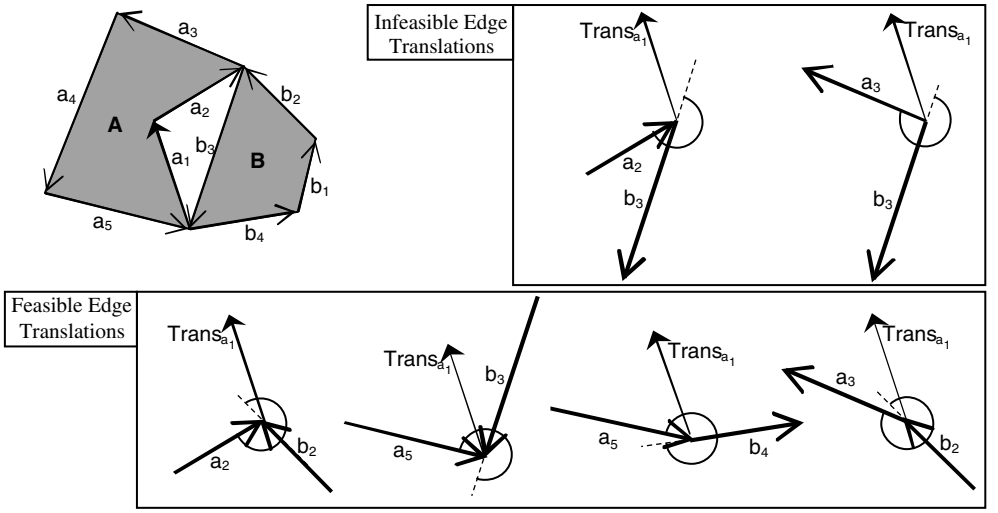


Fig. 11. Elimination of potential translation vector, a_1.

vector a_1 (we have omitted the edge-pairs involving edge a_1 for brevity but these will be feasible translations because the translation vector is also a_1). Once a potential translation fails on one such test, the translation is infeasible and it is eliminated.

At this stage we have found one (or more) feasible translation vectors. Usually there will only be one translation vector but several may be present under the special circumstances that are illustrated in Fig. 12 (an actual screenshot taken from our implementation). The centre of the orbiting square has been set as the reference point for clarity.

This introduces another important aspect of our algorithm; we must maintain the edge that was used to generate the previous translation vector. When several feasible translation vectors are present, we choose

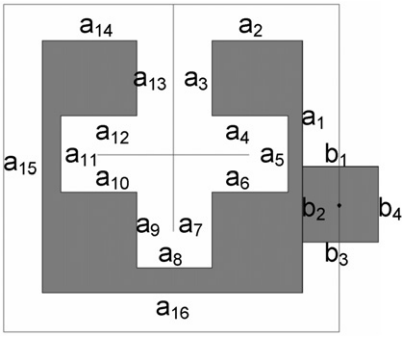


Fig. 12. Two polygons involving exactly fitting 'passageways'.

the edge that is nearest (in edge order) to the previous move. Therefore, returning to Fig. 12, the square enters the passageway using edge a_3 instead of sliding straight over the opening using edge a_{14}. Then, after sliding along into the centre of the stationary polygon, the square has four feasible translation vectors derived from edges a_4, a_7, a_{10} and a_{13}. However, the translation vector derived from edge a_4 will be used because the edge that was used previously was a_3. This is another improvement that we have made to Mahadevan's approach, significantly improving the generality of the approach.

3.2.4. Trimming the feasible translation vector

The last step before we can translate polygon B is to trim the translation vector. This is important because there may be other edges that can interfere with the translation of the orbiting polygon. Fig. 13a provides an example whereby applying the entire translation vector results in the two shapes intersecting. In order to prevent the orbiting polygon entering the body of the stationary polygon, the feasible translation, a_7, must be trimmed to edge a_1 (see Fig. 13b).

In order to find the correct non-intersecting translation we project the translation vector at each of the vertices of polygon B and test it for intersection with all edges of polygon A. This ensures that we correctly identify the vertices of polygon B that will cross into polygon A and reduce the translation distance accordingly using formula (2):

$$\text{New Translation} = \text{Intersection Pt} - \text{Translation}_{\text{Start Pt}}. \tag{2}$$

We must also project the translation vector back from all vertices of polygon A (by translating the end vertex of the translation vector onto each vertex) and perform intersection testing with all edges of polygon B to test if any will cross into polygon B. This is depicted in Fig. 14 using the same example but a different orbiting polygon.

Once again, if there is an intersection, we can reduce the translation distance to reflect this using formula (3):

$$\text{New Translation} = \text{Translation}_{\text{EndPt}} - \text{Intersection Pt}. \tag{3}$$

Our approach is fundamentally the same as used by Mahadevan except that we trim the translation vector at the time of intersection whereas Mahadevan keeps the translation vector the same but stores the minimal intersection distance and trims once after all projections have been completed. As our algorithm trims at every intersection, the translation vector becomes shorter throughout the testing process which can reduce the number of intersections that occur due to fast elimination tests through the use of bounding boxes which are much

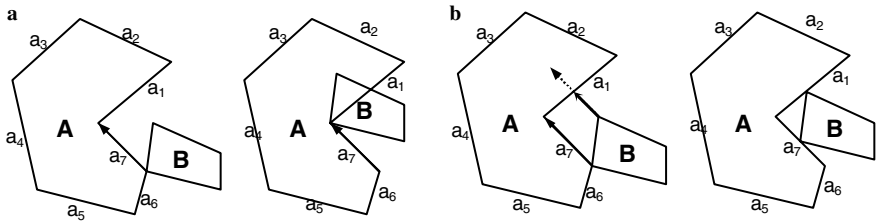


Fig. 13. A translation vector that requires 'trimming' to avoid intersection.

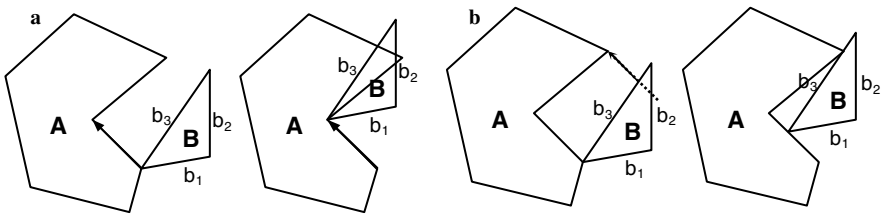


Fig. 14. Trimming with projections from polygon A.

faster than standard line intersections. The approach of Mahadevan may need to calculate several more full intersection tests because the entire translation vector is used at each projection and could potentially require more computation.

3.2.5. Applying the feasible translation vector

The final step is to append the trimmed translation vector to the end of the partial no-fit polygon created so far and polygon B is translated by the trimmed translation vector. This will move the polygon to the next 'decision' point and the process can restart from the detection of touching edges (Section 3.2.1). The only additional check to this process is that we must perform a test to detect if the reference point of polygon B has returned to its initial starting position.

3.3. Start points

In Section 3.2, we identified an approach for the orbiting of one polygon around another using edge comparisons and edge sliding. The overall effect of the approach is similar to the proposed algorithm of Mahadevan. However, some improvements have been proposed that result in the procedure requiring less computation and which enable the approach to be explained more easily. Therefore our approach thus far has some of the same limitations that previous orbital implementations have suffered such as the inability to generate complete no-fit polygons for shapes involving interlocking concavities, jigsaw pieces or holes (see Section 4). Fig. 15a shows two polygons for which the sliding algorithm alone does not produce the complete no-fit polygon. The polygons have concavities that can interlock with each other to create a further no-fit polygon region but this is never found because of the narrow entrance to the stationary polygon's concavity (see Fig. 15b). Another approach is needed to identify such possibilities.

In this section we describe an approach based on the identification of feasible touching but non-intersecting start positions with which the previously described sliding technique can be employed to generate the remaining inner loops of the no-fit polygon construct. For example, if polygon B can be placed in the location shown in Fig. 15c, then the sliding algorithm can be employed to generate the internal no-fit polygon region. Feasible starting positions are found through a modification to the approach described in Section 3.2 and once again the process involves sliding. However, we are not interested in sliding to trace the no-fit polygon but to resolve edge intersections whilst translating along a particular edge vector. In order to explain the process we will examine an example of finding the start positions along one edge of polygon A. This process can then easily be reproduced for each edge of polygon A and polygon B to create all feasible starting positions to allow polygon B to orbit around polygon A in order to create the entire no-fit polygon. Given an edge, e, of polygon A, we can detect the potential starting positions of an orbiting polygon B along e. The process involves translating polygon B such that each of its vertices, in turn, are aligned to the start vertex of e. For each position we perform the following steps. If the polygons do not intersect in this position, then this is noted as a feasible start position for the two polygons. If polygon B intersects with polygon A then we must perform further testing and ultimately employ edge sliding to traverse along edge e until a non-intersecting position is found or the end of the edge is reached. In such an instance, the first test involves examining whether the two connected

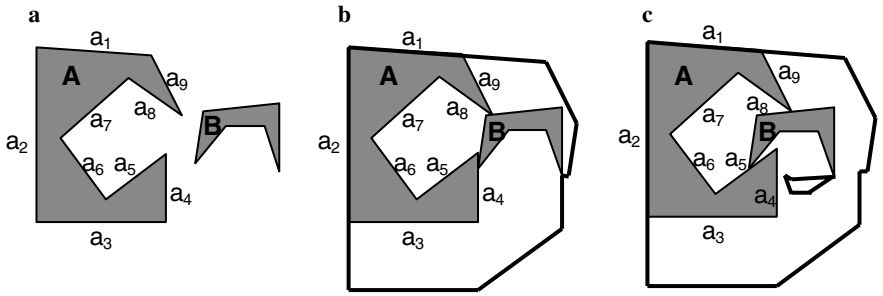


Fig. 15. Interlocking concavities: (a) polygons, (b) no-fit polygon using sliding alone and (c) the complete no-fit polygon.

edges of polygon B (joined at the touching vertex) that are both right or parallel to the edge e. If either of polygon B's connected edges are left of e then they will translate on the inside of polygon A and can be eliminated immediately as they will never yield a feasible starting position when sliding along vector e.

Fig. 16a shows an example of where the connected edge pair, b_3 and b_4, are eliminated from sliding along a_5 because b_4 is left of a_5 (note that this will not be eliminated when using edge b_4 of the polygon B and connected edges a_4 and a_5 of polygon A). Fig. 16b shows an example of where both connected edges, b_1 and b_2, are right of edge a_5.

Now, assuming both connected edges are right of edge e and polygons A and B intersect we can attempt to resolve the overlap by translating the orbiting polygon along the translation vector defined by e. As with the previous section we can now employ trimming to this vector (the procedure is identical). The resultant translation vector can then be applied to slide polygon B along edge e and then another intersection test is performed. If the two polygons still intersect then the process repeats with the translation vector derived from the touching point to the end vertex of edge e. If the intersection has been resolved then the reference point of polygon B is a potential start position. If the entirety of edge e is traversed and there is still an intersection then no feasible starting position can be found along edge e and the aligned vertex/connected edges of polygon B. The next vertex of polygon B is then considered and so on until all edge vertex possibilities have been examined. Note it is also important to examine the edges of polygon B with the vertices from polygon A but as this is an identical procedure, this will not be discussed. Fig. 17 shows this process on the example presented in Fig. 17b. When aligned using a_5 and the vertex connecting edges b_1 and b_2, the polygons are initially intersecting so we need to resolve the intersection along the vector derived from edge a_5.

Fig. 17a shows the trimming process and identification of the closest intersection point, Pt. Polygon B is then translated by the trimmed translation vector. The resulting position is shown in Fig. 17b. The polygons are still intersecting so the procedure must repeat. The translation vector is calculated from the touching point to the end vertex of edge a_5 and is then trimmed as shown in Fig. 17c (only the important trim intersection is shown). After applying this translation to polygon B, the polygons are no longer intersecting and therefore a feasible start point has been found and the standard orbital approach can be employed once more to generate the no-fit polygon loop. A procedure that we include in our implementation is that we calculate the outer

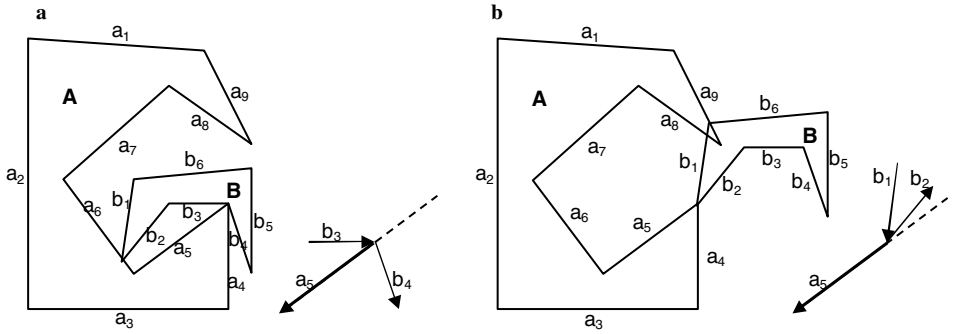


Fig. 16. Vertex alignment of polygon B to polygon A using edge a_5: (a) invalid alignment and (b) valid alignment.

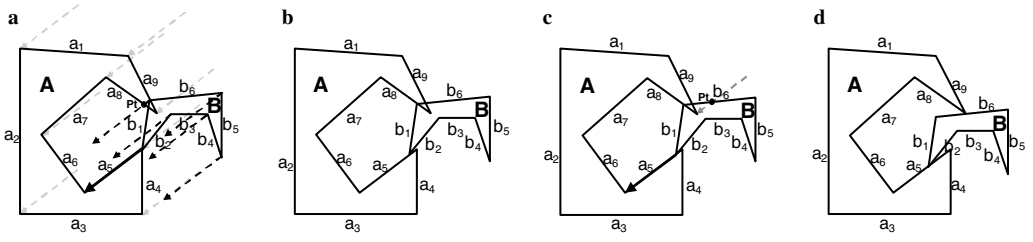


Fig. 17. Start point generation process.

no-fit polygon loop using the approach in Section 3.2 and *then* we apply the starting position procedure on the untraversed/unflagged edges from both polygon A and polygon B in turn. During the process, as soon as a feasible start position is found, we calculate the inner loop of the no-fit polygon that is derived from it (flagging edges as they are traversed as before). This ensures that the algorithm is fast because more edges become flagged as new no-fit polygon loops are produced, thus reducing the computational requirements for generating new feasible start positions. The procedure repeats until all edges have been seen or no more feasible start positions are available and the complete no-fit polygon is returned.

3.4. Summary

Given the functionality outlined in the previous sections we can describe no-fit polygon generation using the following pseudo code:

```
Input : Polygons A and B

Pt_A(ymin) = point of minimum × value of polygon A
Pt_B(ymax) = point of maximum × value of polygon B
IFP = initial feasible position
Bool bStartPointAvailable = true;
Point NFPLoopStartRefPoint;
Point PolygonB_RefPoint;
Array[Line[]] nfpEdges; // an array of arrays of lines to store NFP edges
Int loopCount = 0; // counter for number of loops in NFP

Begin

Place polygons in IFP using translation Pt_A(ymin) − Pt_B(ymax)
NFPLoopStartRefPoint = Pt_B(ymax);
PolygonB_RefPoint = Pt_B(ymax);

While(bStartPointAvailable)
{
   bStartPointAvailable = false;
   // find touching points & segments touching those points, generate touching structures
   Touchers[] toucherStructures = FindTouchers (A, B);

   // Eliminate non-feasible touchers, ones that cause immediate intersection
   Touchers[] feasibleTouchers = CanMove (A, B, toucherStructures);

   // Trim feasible translations against polygon A and B
   Touchers[] trimmedTouchers = Trim(feasibleTouchers, A, B);

   // Sort trimmed translations by length
   Touchers[] lengthSortedTouchers = Sort(trimmedTouchers);

   // Translate polygon B along longest feasible translation vector
   B. Translate(lengthSortedTouchers[0].Translation);

   // Add translation to nfpEdges & mark traversed edge on static
   nfpEdges[loopCount].Add(lengthSortedTranslations[0].Translation);
   A. MarkEdge(lengthSortedTranslations[0].StaticEdgeID);
```

```
    If(NFPLoopStartRefPoint == PolygonB_RefPoint) // completed an NFP loop
    {
       Point nextStartPoint;
       // find next feasible start point – reset PolygonB_RefPoint to relevant point
       bStartPointAvailable = FindNextStartPoint(A, B, & nextStartPoint, & PolygonB_RefPoint);

       if(bStartPointAvailable)
       {
          // Translate polygon B to nextStartPoint
          B.Translate(PolygonB_RefPoint – nextStartPoint);

          NFPLoopStartRefPoint = nextStartPoint;
          loopCount++;
       }
    }
    else
    {
       bStartPointAvailable = true; // allow edge traversal to continue
    }
}
NFP_AB = Complete(nfpEdges); // Reconstitute NFP from nfpEdges
End
```

Pseudo code for **FindNextStartPoint**:

```
Input : PolygonA, PolygonB, reference to Point nextStartPoint, reference to Point PolygonB_RefPoint

Int A_EdgeCount = PolygonA.EdgeCount;
Int B_EdgeCount = PolygonB.EdgeCount;
Edge StaticEdge;
Edge movingEdge;

Begin
for(int i = 0; i < A_EdgeCount; i++)
{
   if(PolygonA.IsEdgeMarked(i))
      continue;
   else
      staticEdge = PolygonA.GetEdge(i);
   for(int j = 0; j < B_EdgeCount; j++)
   {
      movingEdge = PolygonB.GetEdge(j);

      // translate the PolygonB so that movingEdge start is on the start of the static edge
      PolygonB.Translate(movingEdge.Start – staticEdge.Start);
      Bool bFinishedEdge = false;
      Bool bIntersects = PolygonB.IntersectsWith(PolygonA);
```

```
        while(bIntersects AND !FinishedEdge)
        {
            // Edge slide until not intersecting or end of staticEdge reached
            Toucher currentToucher = MakeToucher(staticEdge.Start);
            Toucher trimmedToucher = Trim(currentToucher, PolygonA, PolygonB);
            PolygonB.Translate(trimmedToucher.Translation);

            bIntersects = PolygonB.IntersectsWith(PolygonA);

            if(bIntersects)
            {
                If(movingEdge.Start == staticEdge.End)
                    bFinishedEdge = true;
            }
        }
        // mark the traversed edge as seen (whether edge start point found or not)
        staticEdge.Mark(true);

        if(!bIntersects)
        {
            // set the references to the points passed in to be the nextStartPoint
            // and return true;
            nextStartPoint = movingEdge.Start;
            PolygonB_RefPoint = movingEdge.Start;

            return true;
        }
    }
}
return false;
End
```

4. Problem cases

In this section we discuss each of the problem cases that cause difficulties with other methods and show why our approach is able to handle them without specific case-by-case implementations. In each figure of this section we identify the stationary and orbiting polygons in dark and light grey shading respectively and we show the reference point of the orbiting polygon by a black dot. This is used to trace the loops of the no-fit polygon (which we number). We present the data for these problem cases within the online companion for this paper.

4.1. Interlocking concavities

The interlocking of concavities is a typical problem case for the previous orbital sliding approaches in the literature such as (Mahadevan, 1984). Fig. 18 shows a screenshot of our implementation where we generate the complete no-fit polygon of identical shapes with one rotated through 180°. The shapes in these orientations involve many different interactions of the concavities and, therefore, multiple feasible start points. The no-fit polygon of these two shapes results in *six* loops within the no-fit polygon (one outer loop, 1, and five internal loops, 2–6).

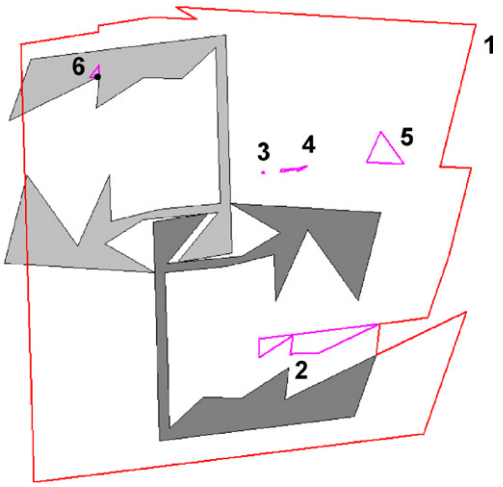


Fig. 18. Multiple interlocking positions.

4.2. Exact fit: Sliding

The next case involves sliding through exactly fitting "passageways" which cannot be handled by either the Minkowski sum approach of (Bennell et al., 2001) or the orbital approach given in (Mahadevan, 1984). We propose an extension to Mahadevan's algorithm that can deal with such problems through the maintenance of the previously traversed edge and, thus, enabling consecutive edges of the stationary polygon to be used in the next translation iteration (see Fig. 19). We previously discussed another example of this case in Fig. 12.

4.3. Exact fit: Jigsaw pieces

The problem case involving jigsaw pieces (also called "lock-and-key") that link exactly together is another form of the interlocking concavity case outlined in Section 4.1. However, the distinguishing feature of jigsaw pieces is that they fit together with no movement, thus creating a singular feasible point within the no-fit polygon (rather than an internal loop in the interlocking concavity case). In our approach, this position is found by the generation of feasible starting positions. Of course, the algorithm will still try to slide the orbiting polygon along edges but, in this case, there is no feasible translation vector therefore distinguishing that the position is

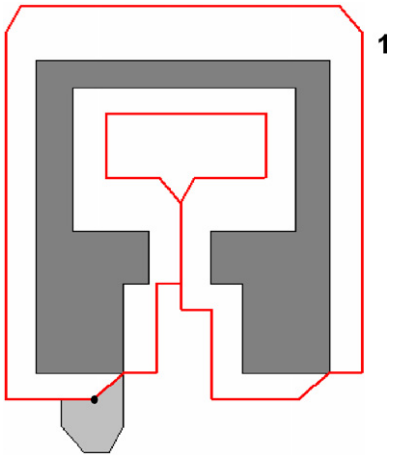


Fig. 19. Exact fit sliding through a "passageway".

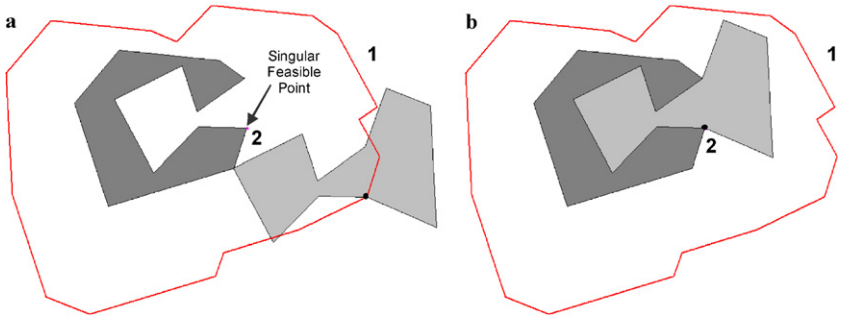


Fig. 20. Jigsaw pieces: (a) outer no-fit polygon loop and (b) singular feasible internal position.

just a singular feasible point location (see Fig. 20). Most of the previous approaches have suffered from such degeneracy including: the Minkowski sum approach where no boundary exists to identify lock-and-key positions within the no-fit polygon, convex decomposition whereby it is not possible to find such positions through recombination alone and the sliding algorithm of Mahadevan which suffers from the same difficulties as with interlocking concavities (see Section 4.1).

4.4. Holes

There have been few approaches within the literature that can operate effectively on shapes containing holes. This could be due to the difficulty of the generation process for no-fit polygons involving holes. However, through the detection of feasible starting positions, the no-fit polygon can be generated easily and, more importantly, completely.

Fig. 21 shows an example involving a shape with two holes and a shape that can be placed within several distinct regions of the holes. The figure shows a mixture cases involving holes and interlocking concavities: loop 1 is the outer no-fit polygon loop, loops 2, 3, 5, 6, 7 are all hole cases, and loops 4 and 8 are cases whereby the concavity of the smaller orbiting shape interacts with the narrow gaps within the larger hole of the stationary shape. Also, convex decompositions can result in a larger number of pieces and become more difficult with the introduction of holes. Therefore union or recombination is also compromised due to the larger number of

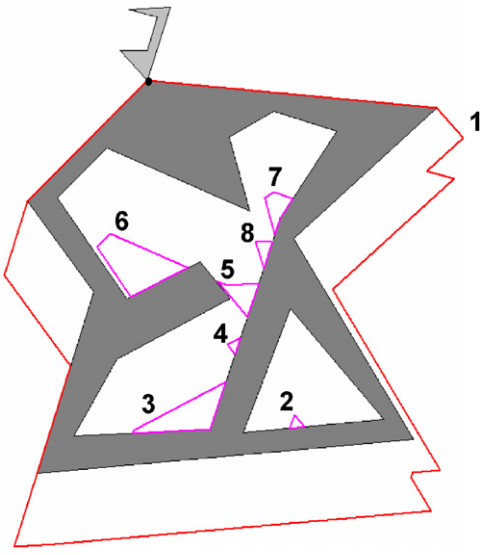


Fig. 21. No-fit polygon of two polygons, one of which involves multiple holes.

intersecting sub-no-fit polygon elements that need to be untangled. The phi-function approach of Stoyan could deal with many of the degeneracies through the resolution of primary objects. However, this approach will ultimately impact on the efficiency of intersection testing as many separate tests must be performed (especially with very complex shapes). Previous orbital approaches have only produced the outer no-fit polygon loop and Minkowski sum approaches can become difficult due to the untangling of edges. The authors are unaware of any previous no-fit polygon generation methods that have specifically presented no-fit polygon constructs for shapes involving such degeneracies.

5. Generation times on the benchmark problems

In order to demonstrate the speed and capabilities of our new no-fit polygon procedure, we report the generation times for 32 benchmark problems which we have gathered from the literature (this set of benchmarks is all of those of which we are aware). These datasets can be found on the EURO Special Interest Group on Cutting and Packing (ESICUP) website.[1] For each problem we report the total generation time and number of no-fit polygons generated per second (see Table 2). The ''Logical Total Number of Shapes'' (column E) is found by $E = B^*D$ and the ''Total Number of NFPs'' (column F) is calculated by $F = E^2$ (i.e. we calculate the NFP for every logical shape against every other logical shape). All experiments have been conducted on a Pentium 4 2 GHz processor with 256MB RAM.

Table 2 shows that the no-fit polygon constructs can be generated within reasonable time frames on most of the benchmark data. Typically the algorithm can generate about 1000 no-fit polygons per second although this can vary drastically depending on the number of line segments in each problem. For example, the two problems with the lowest no-fit polygons generated per second, ''Swim'' and ''Profiles9'', involve pieces with many line segments (up to 67) due to approximation of arcs and, further to this, ''Profiles9'' contains several shapes with holes (see Fig. 22).

The slowest overall generation times occur within the Poly4b and Poly5b problems and are simply due to the large number of no-fit polygons to generate. Whilst the majority of the literature datasets do not contain any degenerate cases, the ''Profiles6'' and ''Profiles9'' datasets do contain several of the degenerate cases that we have identified (holes and interlocking cavities). As mentioned in Section 2.3.2, other reported generation times for available benchmark problems of the literature can be found in (Bennell et al., 2001). Table 3 shows a comparison of the generation times of our new orbital approach to the Minkowski based approach of Bennell, Dowsland and Dowsland using a DEC Alpha 3000/400. We set each problem to use 0° rotations for comparative purposes (polygons in their drawn orientations) as these were the rotational constraints used in (Bennell et al., 2001).

Table 3 shows that the new approach generates the no-fit polygons quicker than the Minkowski sum approach for all of the five datasets. Whilst this comparison is unfair due to the difference in computing power available to the two approaches, it does show that the approach proposed in this paper can generate no-fit polygons very quickly with execution times that are consistent with other published approaches.

We have demonstrated that, for the problems taken from the literature (including several datasets from the textiles and metal cutting industries), we are able to quickly and robustly generate all of the no-fit polygons. Whilst very large problems may require a few minutes to generate all of no-fit polygons, such as the Poly4b and Poly5b test problems, for repeated automatic nesting it is likely that we will more than recover this overhead by being able to use the faster no-fit polygon based intersection testing.

6. Summary

In this paper we have reviewed the main techniques that have been used for no-fit polygon generation within the literature and shown that it is an important construct that can be used for the development of faster automated packing algorithms as opposed to the traditional trigonometric based approaches. A complete and robust implementation of the orbital method of the no-fit polygon has been presented. This approach can deal

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

Table 2
No-fit polygon generation times for 32 datasets of the literature

A	B	C	D	E	F	G	H
Dataset	Number of different shapes	Rotational constraints	Number of rotations per shape	Logical total number of shapes	Total number of NFPs	Total generation time (seconds)	NFPs per second
Albano180	8	180	2	16	256	0.32	800
Albano90	8	90	4	32	1024	0.71	1442
Blaszl	7	180	2	14	196	0.21	933
Blasz2	4	90	4	16	256	0.19	1347
Dagli	10	90	4	40	1600	0.93	1720
Dighe1	16	90	4	64	4096	1.28	3200
Dighe2	10	90	4	40	1600	0.62	2581
Fu	12	90	4	48	2304	0.52	4431
Jakobs1	25	90	4	100	10,000	5.57	1795
Jakobs2	25	90	4	100	10,000	5.07	1972
Mao	9	90	4	36	1296	1.41	919
Marques	8	90	4	32	1024	0.79	1296
Poly1a	15	90	4	60	3600	1.37	2628
Poly2a	15	90	4	60	3600	1.37	2628
Poly3a	15	90	4	60	3600	1.37	2628
Poly4a	15	90	4	60	3600	1.37	2628
Poly5a	15	90	4	60	3600	1.37	2628
Poly2b	30	90	4	120	14,400	7.54	1910
Poly3b	45	90	4	180	32,400	27.14	1194
Poly4b	60	90	4	240	57,600	68.45	841
Poly5b	75	90	4	300	90,000	141.90	634
Shapes	4	90	4	16	256	0.38	674
Shapes0	4	0	1	4	16	0.11	145
Shapes1	4	180	2	8	64	0.19	337
Shirts	8	180	2	16	256	0.33	776
Swim	10	180	2	20	400	6.08	66
Trousers	17	180	2	34	1156	0.73	1584
Profiles6	9	90	4	36	1296	0.86	1507
Profiles7	9	90	4	36	1296	0.58	2234
Profiles8	9	90	4	36	1296	0.56	2314
Profiles9	16	90	4	64	4096	44.30	92
Profiles10	13	0	1	13	169	0.55	307

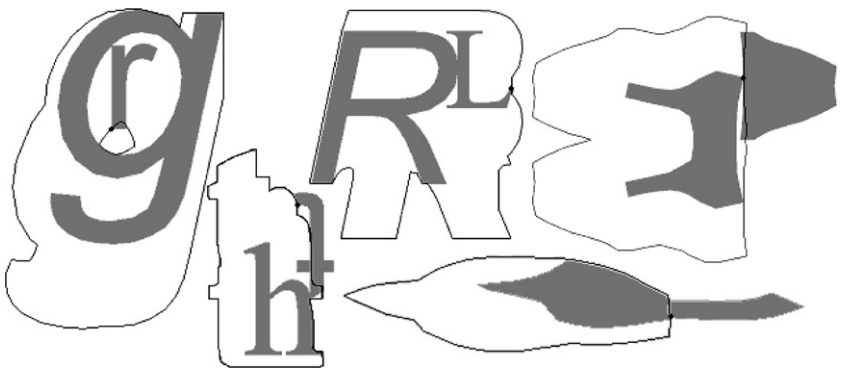


Fig. 22. A selection of pieces and no-fit polygons from the "Profiles 9" (letters) and "Swim" datasets.

with degenerate cases that the previous methods cannot resolve easily. Furthermore, we show that the execution times required to produce the no-fit polygons on the available benchmark problems from the literature are reasonable and that they are consistent with the approach of Bennell et al. (2001). As far as the authors

Table 3
Comparison of generation times of the Minkowski approach of Bennell et al. (2001) with the presented orbital approach on five literature datasets

Dataset (0° rotation only)	Total generation times (seconds)	
	Minkowski approach (Bennell et al., 2001) (DEC Alpha 3000/400)	Orbital approach (presented) (Pentium 4 2 GHz)
DS1	0.32	0.23
DS2	0.22	0.04
DS3 (Shirts)	0.23	0.17
DS4 (Shapes0)	0.38	0.11
DS5 (Blasz1)	0.36	0.13

are aware, not only is this the first time that a full implementation of an orbital no-fit polygon generation procedure has been proposed for publication within the literature but also the first time that such an approach has been rigorously investigated for robustness with the known degenerate cases and has provided computational results on such a wide range of existing benchmark data.

References

Adamowicz, M., Albano, A., 1976. Nesting two dimensional shapes in rectangular modules. Computer Aided Design 8 (1), 27–33.
Agarwal, P.K., Flato, E., Halperin, D., 2002. Polygon decomposition for efficient construction of Minkowski sums. Computational Geometry Theory and Applications 21, 39–61.
Amenta, N., 1997. Computational geometry software. In: Goodman, J.E., O'Rouke, J. (Eds.), Handbook of Discrete and Computational Geometry. CRC Press LLC, Boca Raton, pp. 951–960 (Chapter 52).
Art, R.C., 1966. An approach to the two dimensional irregular cutting stock problem. IBM Cambridge Scientific Centre, Report 36-Y08.
Avnaim, F., Boissonnat, J.D., 1987. Simultaneous containment of several polygons. In: Proceedings of the 3rd Annual ACM Symposium on Computational Geometry, pp. 242–250.
Avnaim, F., Boissonnat, J.D., 1988. Polygon placement under translation and rotation. Lecture Notes in Computer Science 294, 323–333.
Bennell, J.A., Dowsland, K.A., Dowsland, W.B., 2001. The irregular cutting-stock problem – a new procedure for deriving the no-fit polygon. Computers and Operations Research 28, 271–287.
Burke, E.K., Hellier, R.S.R., Kendall, G., Whitwell, G., in press. A new bottom-left-fill algorithm for the two-dimensional irregular packing problem, Operations Research.
Chazelle, B., Dobkin, D.P., 1985. Optimal convex decompositions. In: Toussaint, G.T. (Ed.), Computational Geometry. North-Holland, Amsterdam, pp. 63–133.
Cheng, S.K., Rao, K.P., 1997. Quick and precise clustering of arbitrarily shaped flat patterns based on stringy effect. Computers and Industrial Engineering 33 (3–4), 485–488.
Cuninghame-Green, R., 1989. Geometry, shoemaking and the milk tray problem. New Scientist 12th August 1989, no. 1677, pp. 50–53.
Cuninghame-Green, R., 1992. Cut out waste!. O.R. Insight 5 (3) 4–7.
Dowsland, K.A., Dowsland, W.B., 1992. Packing problems. European Journal of Operations Research 56, 2–14.
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.
Dyckhoff, H., 1990. A typology of cutting and packing problems. European Journal of Operational Research 44, 145–159.
Garey, M.R., Johnson, D.S., 1979. Computers and Intractability: A guide to the Theory of NP-Completeness. W.H. Freeman and Company, San Francisco.
Ghosh, P.K., 1991. An algebra of polygons through the notion of negative shapes. CVGIP: Image Understanding 54 (1), 119–144.
Ghosh, P.K., 1993. A unified computational framework for Minkowski operations. Computers and Graphics 17 (4), 357–378.
Gomes, A.M., Oliveira, J.F., 2002. A 2-exchange heuristic for nesting problems. European Journal of Operational Research 141, 359–370.
Grinde, R.B., Cavalier, T.M., 1995. A new algorithm for the minimal-area convex enclosure problem. European Journal of Operations Research 84, 522–538.
Hertel, S., Mehlhorn, K., 1983. Fast triangulation of simple polygons. In: Proceedings of the 4th International Conference, in Foundations of Computational TheoryLecture Notes in Computer Science, 158. Springer-Verlag, Berlin, pp. 207–218.
Konopasek, M., 1981. Mathematical treatment of some apparel marking and cutting problems. US Department of Commerce Report, 99-26-90857-10.
Li, Z., Milenkovic, V.J., 1995. Compaction and separation algorithms for non-convex polygons and their applications. European Journal of Operational Research 84, 539–561.

Mahadevan, A., 1984. Optimisation in computer aided pattern packing, Ph.D. thesis, North Carolina State University.
Milenkovic, V.J., 1999. Rotational polygon containment and minimum enclosure using only robust 2D constructions. Computational Geometry 13, 3–19.
O'Rourke, J., 1998. Computational Geometry in C, second ed. Cambridge University Press, ISBN 0-521-64976-5.
Ramkumar, G.D., 1996. An algorithm to compute the Minkowski sum outer-face of two simple polygons. In: Proceedings of the 12th Annual Symposium on Computational Geometry, pp. 234–241.
Scheithauer, G., Terno, J., 1993. Modeling of packing problems. Optimization 28, 63–84.
Seidel, R., 1991. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. Computational Geometry Theory and Applications 1, 51–64.
Stoyan, Y., Ponomarenko, L.D., 1977. Minkowski sum and hodograph of the dense placement vector function, SER. A10, Technical Report, SER. A10, Reports of the SSR Academy of Science.
Stoyan, Y., Terno, J., Scheithauer, G., Gil, N., Romanova, T., 2001. Phi-functions for primary 2D-objects. Studia Informatica Universalis 2 (1), 1–32.
Stoyan, Y., Scheithauer, G., Gil, N., Romanova, T., 2002. Phi-functions for complex 2D-objects, Technical Report, MATH-NM-2-2002. April 2002, Technische Universitat Dresden.
Sweeney, P.E., Paternoster, E.R., 1992. Cutting and packing problems: A categorized application-orientated research bibliography. Journal of the Operational Research Society 43 (7), 691–706.