Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 35 (2008) 267 – 281
computers & operations research
www.elsevier.com/locate/cor

A comprehensive and robust procedure for obtaining the nofit polygon using Minkowski sums

Julia A Bennell*, Xiang Song

School of Management, University of Southampton, Southampton SO17 1BJ, UK

Available online 17 April 2006

Abstract

The nofit polygon is a powerful and effective tool for handling the geometric requirements of solution approaches to irregular cutting and packing problems. Although the concept was first described in 1966, it was not until the early 90s that the general trend of research moved away from direct trigonometry to favour the nofit polygon. Since then, the ability to calculate the nofit polygon has practically become a pre-requisite for researching irregular packing problems. However, realization of this concept in the form of a robust algorithm is a highly challenging task with few instructive approaches published. In this paper, a procedure using the mathematical concept of Minkowski sums for the calculation of the nofit polygon is presented. The described procedure is more robust than other approaches using Minkowski sum knowledge and includes details of the removal of internal edges to find holes, slits and lock and key positions. The procedure is tested on benchmark data sets and gives examples of complicated cases.

Scope and purpose Cutting and packing problems involving irregular shapes feature in a wide variety of manufacturing processes. Automated solution techniques that can generate packing arrangements more efficiently than current technology that employs user intervention, must be able to handle the complex geometry that arises from these problems. The nofit polygon has been demonstrated to be an effective tool in providing efficient handling of the geometric characteristics of these problems. The paper presents a new algorithmic procedure for deriving this tool.

Keywords: Cutting and packing; Nesting; Geometric algorithms; Configuration space

1. Introduction

Cutting and packing problems involving irregular shapes are common in a variety of manufacturing processes. They occur whenever a piece of irregular shape is to be cut from a sheet of stock material. Examples include dye-cutting in the engineering sector, parts nesting for shipbuilding, marker layout in the garment industry, and leather cutting for shoes, furniture and other goods. The paper specifically addresses the geometric calculations required for tackling these problems. Here we consider that shapes are irregular if they are; polygonal, i.e no arcs; simple, i.e. non-self-intersecting; and non-rectangular. Even when all the components are rectangular the problem of finding layouts that minimize waste is known to be NP-hard. Where irregular components are involved an extra dimension of complexity is generated by the geometry.

* Corresponding author. Tel.: +44 0 2380 595671; fax: +44 0 2380 593844.
E-mail addresses: j.bennell@soton.ac.uk (J.A. Bennell), x.song@soton.ac.uk (X. Song).

0305-0548/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2006.02.026

The precise requirements of a good layout will differ from industry to industry and this has led to a variety of algorithmic approaches. In spite of their differences, all the methods have a common requirement in which they need to be able to identify whether a layout is feasible or not, i.e. do any of the pieces overlap. Early research handled this problem in a number of ways. Adamowicz and Albano [1] chose to nest pieces into simpler shapes where the geometry can be more easily calculated. If the shapes are used directly then the intersection of pieces can be handled by direct trigonometric approaches such as the D function [2,3]. Alternatively the stock sheet and the pieces can be approximated as grid squares, often referred to as the raster method. Hence, if a piece occupies, fully or partially, a grid square it is coded as occupied [4,5].

Although all these approaches have merit, it is widely recognized that the nofit polygon (NFP) is more efficient, provided you have a robust and efficient NFP generator, and has become the principle approach for handling the geometry in nesting problems. Unfortunately, some researchers believe that despite the value of this tool, its introduction may have stifled research into this variant of packing problems. Wäscher et al. [6] report that there have been only 21 publications in irregular problems in the last 10 years. Researchers attribute this to the fact that the realization of the NFP as a robust algorithm is, in itself, a highly challenging task. Those considering embarking on research into irregular shaped packing may be discouraged by the significant investment of time required in first developing an NFP generator. Hence, it is essential that robust and easily realizable algorithms are available in order to facilitate new interest into this important problem.

The primary purpose of this paper is to introduce a new procedure for calculating the NFP. The method is developed from the theory of Minkowski sums [7] and builds on the principles proposed by Ghosh [8,9] and by Bennell et al. [10]. Further, the paper includes an algorithmic procedure for eliciting the true boundary of the NFP, including holes, slits and exact fits. It should be noted that the polygons considered in this paper are two-dimensional and restricted to translational motion i.e. rotations are not considered. The next section outlines the most commonly cited approaches for calculating the NFP and points out their positive features and disadvantages. Section 3 reviews in more detail the Minkowski sum approach. This is followed by a description of our new procedure based on Minkowski sums. Section 5, develops our approach for removing redundant internal points and therefore identifying the true boundary. In both cases the full algorithmic steps are provided. Finally, we develop some theoretical and empirical analysis of the approach to demonstrate its robustness with respect to being capable of handling all combinations of simple polygons.

2. Documented approaches for generating the nofit polygon

The NFP is a combination of the properties of two component polygons that, as a result, represents all the relative positions of the two polygons in which they either touch or overlap. It is well documented that the NFP can reduce the complexity of detecting overlap between two pieces from $\mathrm{O}(nm + n + m)$, where n and m are the number of edges in each polygon, obtained from direct trigonometry, to a simple point inclusion test of $\mathrm{O}(k)$, where k is the number of edges in the NFP. Full explanations of the concept can be found in Mehadevan [2], Ghosh [9], Bennell [11], Bennell et al. [10], and O'Rourke [12], where the most intuitive description is found in Cunningham-Green [13], who describes the motion of one polygon sliding around the boundary of the other; often referred to as the orbiting method. Fig. 1a and b illustrates the motion of polygon B, the orbiting polygon, sliding around A, the fixed polygon, tracing the locus of a reference point on B. He also notes that when both polygons are convex, the NFP is convex and an exact replication of the edges of both polygons, with opposite orientation, sorted into their slope order. Fig. 1c shows the edges of both polygons, where A has counterclockwise orientation and B has clockwise orientation, sorted into slope order; these can be directly mapped onto the NFP in Fig. 1b. Note that this role and orientation of polygons A and B will be adopted for the remainder of the paper. Cunninghame-Green's [13] observations underpin two of the most common approaches to generating the NFP; the orbiting method that simulates the sliding motion, and Minkowski sums that sort the edges according to the slope order and edge precedence, i.e the sequential order of edges around the polygons. A further approach commonly employed is that of decomposition. A brief description of each is provided here.

2.1. Minkowski sum

Clearly, when both component polygons are convex the NFP is very simple to calculate by sorting the edges into slope order. Further, when one of the polygons is convex and the other is an arbitrary simple polygon, the NFP can still

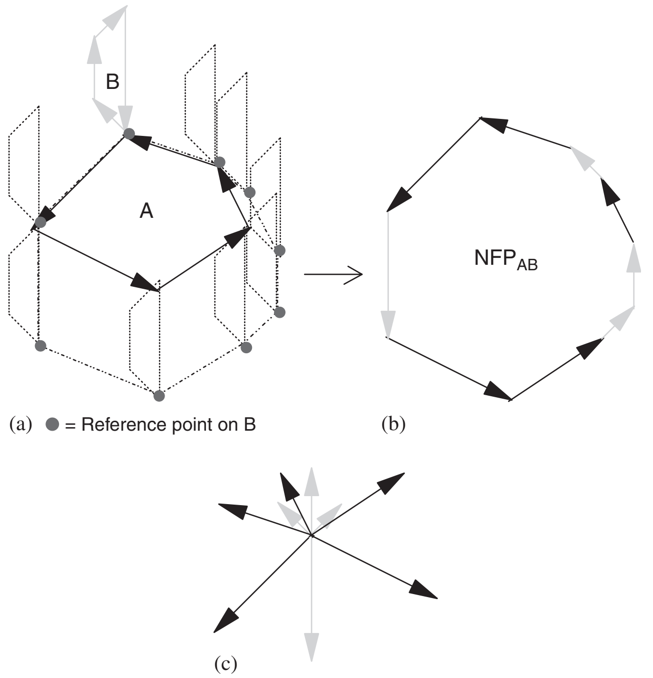

Fig. 1. The locus of the reference point on B traces the NFP as it slides around A. This is equivalent to connecting the edges in slope order.

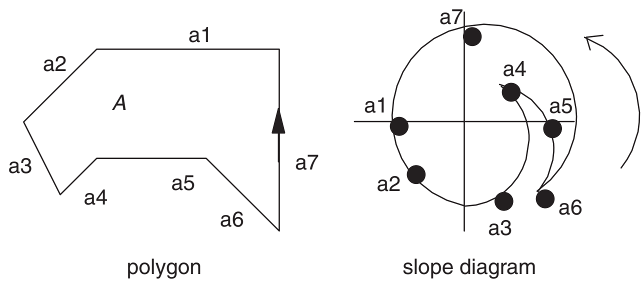

Fig. 2. A simple polygon and respective slope order.

be easily obtained from the slope order and the precedence of the edges. In this case, the NFP is obtained by forming an edge list that follows the precedence of the simple polygon, assigned as polygon A, in a counterclockwise direction, and adding the edges of the convex polygon, assigned polygon B, to the list whenever they are encountered in the slope order. Due to the concavities in A, the precedence will necessitate a clockwise turn through the slope order, if the edges of the convex polygon are encountered in the clockwise direction; they are included in the edge list with negative direction. Note that this also retains the precedence of the edges of the convex polygon. Unfortunately the resulting polygon created from this edge list is complex and further computation is required to remove edges or parts of edges that are not part of the boundary of the NFP. Fig. 2 illustrates the tracking of the precedence order of a simple polygon through the slope order in the form of a *slope diagram*. The points on the slope diagram indicate the slope or gradient of an edge, but not its length, and the arcs illustrate the path through the edges that defines the polygon. This representation will be used further in Section 3.

It is worth noting that even in the convex–simple case, the NFP may contain holes. These represent a non-overlapping placement position within a concavity that cannot be encountered through sliding. Such cases will be discussed later in the paper. When both polygons contain concavities, following the precedence of both polygons, simultaneously, becomes impossible without further modification to the approach. Since it is these principles that form the basis of the Minkowski sum approaches presented in this paper, these issues will be discussed in later sections.

Kaul et al. [14] investigate the worst case asymptotic complexity of the output size of the Minkowski sum for convex, simple–convex and simple–simple cases and report a worst case output size of $O(m+n)$, $O(mn)$ and $O(m^2n^2)$, respectively, where m and n are the number of edges in each polygon. They also develop a Minkowski sum generator, which is based on the principle of generating supporting lines of the edges of each polygon. For all edge vertex-combination between the polygons, the cone of normals can be defined. If an edge with given direction and orientation is within the cone, then it is potentially a supporting edge. Once all supporting edges are obtained, a further algorithm is required to identify the boundary. They report a time complexity of $O((m^2n^2)(m+n) \log O(m+n))$. A description of this approach can also be found in deBerg et al. [15].

2.2. Orbiting method

An alternative approach is to use the orbiting method [2]. This approach attempts to simulate the sliding motion of one polygon around the other. When both polygons are convex, this is equivalent to sorting the edges in slope order. However, when one or both of the polygons have concavities, the full extent of some edges may not be available to slide along without generating overlap. Mahadevan's [2] approach calculates the nature of the touching vertices and edges, at a given point, in order to identify the next edge to slide along; this is the translation vector. He then projects forward the vertices of the orbiting polygon and projects backward the vertices of the fixed polygon in order to identify the closest point of intersection. The orbiting polygon is then translated along the translation vector to the point of the closest intersection. The key criticism of this approach is that it can only identify the external boundary of the NFP and any holes that may exist will be missed. Burke et al. [16] have proposed some modifications to Mehadevan's approach that improves the computational efficiency and permits the identification of holes. They first find the outer face of the NFP using the principles of Mehadevan's sliding approach, while recording each edge of the polygons that have been partially or fully traversed. The edges that are not flagged are then candidates for possible holes. A process of identifying all feasible touching start positions is performed for the candidate edges. If a feasible start position is found, the sliding approach is performed again from that starting point. This continues until all edges not flagged have been investigated.

2.3. Decomposition

Given the comparative complexities of the described approaches when one or both polygons are simple, decomposing the component polygons into suitable sub-polygons is an attractive option. Examples in cutting and packing literature include convex decomposition [17] and star shaped decomposition [18]. As previously described, the NFP of two convex polygons is trivial. Li and Milenkovic selected star shaped polygons since the NFP of two star shaped polygons is also star shaped. Hence, in generating the sub-NFP, they need only be concerned with the outer boundary.

Although decomposition simplifies the core NFP operation, it also generates two further issues; efficient decomposition and robust recombining of the sub-NFPs. Agarwal et al. [19] investigated these issues for convex decomposition. They determined that although optimal decomposition in general significantly reduces the computation time for the Minkowski sum construction, the decomposition is computationally expensive and heuristics that approximate optimal decomposition are preferred. In addition they discovered that the relationship between the number of convex sub-polygons and the time to construct the final NFP was not a straight forward trade-off. Experimentation demonstrated that the relationship between the two polygons to be summed impacts on the Minkowski sum construction. Hence, they decompose the polygons simultaneously in order to minimize a cost function that takes account of this relationship. Recombination provides further challenges, since if edges from two sub-NFPs coincide or cross in and out of each other, careful analysis must be performed to detect whether these edges are part of the boundary of the NFP.

A recent development in handling the geometric properties of irregular packing problems, in both two and three dimensions, is that of the Phi-function [20,21]. Although phi-functions are not strictly NFP, they are a related concept and have proved to be both efficient and effective. The phi-function is a series of mathematical expressions that represent the mutual positions of two objects. Specifically the value of the phi-function is greater than zero if the objects are separated, equal to zero if their boundaries touch and less than zero if they overlap, and the value should represent the Euclidean distance between the two objects. Stoyan et al. [21] analytically construct phi-functions for all primary objects; rectangles, circles and other regular convex polygons. As a result, arbitrary polygons or parallelepipeds can be handled by representing them as a finite combination (union, intersection, complement) of primary objects.

All of the methods described have been somewhat successful. However, all experience difficulties when the problem instance becomes complex, for example, degenerate cases where one or more dimension fits exactly into a concavity. In such instances computational times can be large and the algorithm proposed can be difficult to realize. In this paper we will further develop the Minkowski sum approach and present a robust, efficient and simple algorithm. Although we do not dismiss the potential of the other approaches, a clear advantage of this approach is that the basic Minkowski sum can be obtained through simple rules designed to list the edges according to the precedence of both polygons while sorting in slope order. For all the described methods, the identification of holes and degenerate cases is somewhat laborious.

3. Approaches to finding the nofit polygon using Minkowski sums

As previously described, generating the NFP of both the convex–convex and simple–convex case can easily be solved using the slope and precedence order of the edges. Ghosh [8,9], developed these ideas and proposed the theory of boundary addition, which can be illustrated through the use of a slope diagram. Fig. 3a illustrates two polygons converted into their respective slope diagrams. Note that the polygons have opposite orientation, A has counter clockwise orientation; positive, and B has clockwise orientation; negative. The boundary addition theorem states that the Minkowski sum, $A \oplus -B$, which is equivalent to the NFP, can be obtained from merging the slope diagrams of A and $-B$ and is given by an edge list that follows the slope order and retains the precedence of the edges of both A and $-B$ through counter clockwise (positive) and clockwise (negative) turns. The simple–simple case is also comprehensively addressed by the boundary addition theorem. However, when concavities in the two polygons interact, it becomes impossible to define one path through the slope diagram that retains the precedence of both polygons. Ghosh overcame this problem by defining parallel paths, where the precedence of one or the other

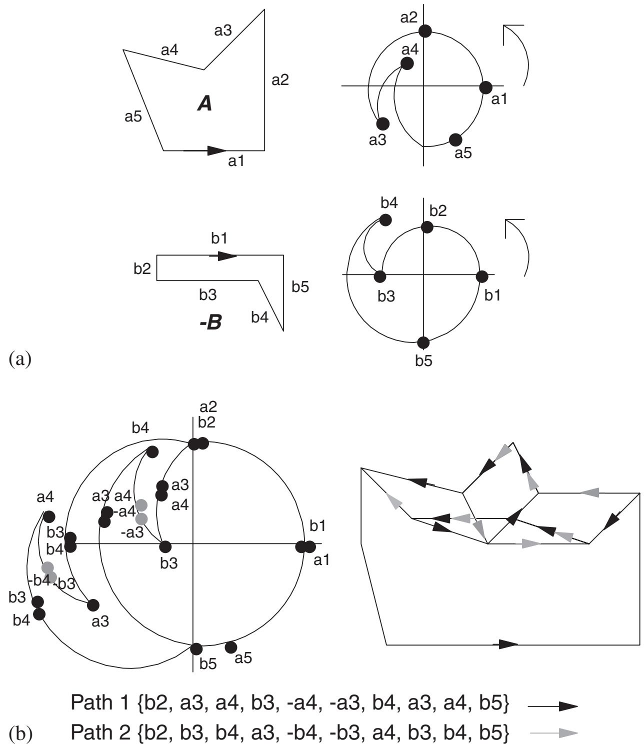

Fig. 3. Ghosh [8] approach to two simple polygons with interacting concavities.

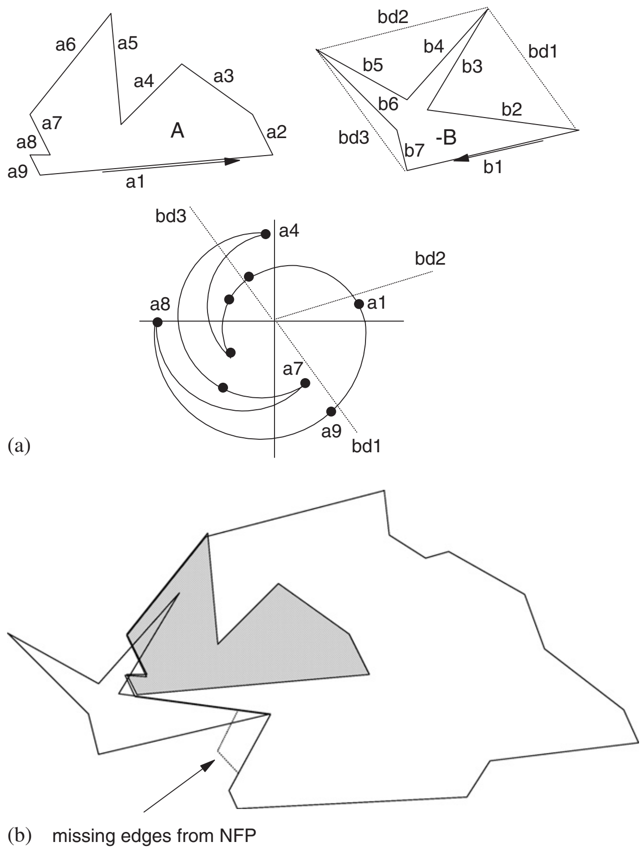

Fig. 4. An example when edges may be missed when using Bennell et al. [10].

polygon would dominate. His approach is illustrated in Fig. 3b. Note that edges b5, a5, a1, b1, a2, b2 are common to both paths. After b2 the paths split. For the B dominant path, it traverses b3, b4 to b5 and includes a3 and a4 each time they are passed, positively in counter clockwise direction and negatively in clockwise direction. Unfortunately, when multiple concavities interact, between and within the polygons, it becomes impossible to define algorithmic rules for robustly untangling the conflicting areas.

Bennell et al. [10] propose an alternative approach to the simple–simple case. Their approach exploits the knowledge that the simple–convex case is trivial and that the Minkowski sum of a simple polygon A with the convex hull of polygon B, $A \oplus -conv(B)$, will contain all the boundary and internal points of the original simple–simple case, $A \oplus -B$. In order to generate $conv(B)$, dummy edges are introduced that replace the edges that make up the concavities of B. Clearly these dummy edges appear, both positively and negatively, in $A \oplus -conv(B)$. Hence replacing the dummy edges in the edge list of $A \oplus -conv(B)$ by the real edges of B, following the precedence of the edges and including A edges when they are passed in the slope order, will result in $A \oplus -B$.

While Bennell, Dowsland and Dowsland's approach works well on the benchmark data sets [22], further investigation highlights some ambiguity in the procedure for replacing dummy edges. This is illustrated through the example in Fig. 4.

Fig. 4(a) illustrates the generation of $A \oplus -conv(B)$. It is clear that dummy edge bd_1 will slide across the vertex between edge a_9 and edge a_1. Hence appearing on the slope diagram on that traversal alone. However, we can observe in Fig. 4(b) that vertex (a_9, a_1) cannot slide along the full extent of edge b_2 due to a collision between edge b_3 and vertex (a_6, a_7). However, if when replacing the dummy edge in the slope diagram, only the A edges on the same traversal are considered, this collision will not be included in the boundary of the NFP. Fig. 4(b) illustrates, the resulting $A \oplus -B$ when only the current traversal is considered, and the true NFP. The problem can be resolved by including the additional edges. However, defining rules to determine the instances in which extra A edges should be included has proved difficult.

4. A revised procedure for obtaining the boundary of the nofit polygon

The proposed new approach for finding the NFP is also based on the boundary addition theorem and inspired by the observation that the simple–convex case is trivial. Hence one polygon can remain unchanged (in this case polygon A). Further, the new approach is simple, intuitive and removes ambiguity concerning which edges should be included in the edge list. The basic idea is to break polygon B into groups that are in either continuous counter clockwise or clockwise order. Each of the groups can then be individually merged with the slope diagram of A without conflict. When combining the merged lists, linking edges need to be included in order to maintain the precedence of the edges in each polygon. As with Ghosh and Bennell, Dowsland and Dowsland, the resulting edge list is a complex polygon, where the edges represent all the boundary edges and some internal points of the Minkowski sum. In order to have successfully generated the NFP, the edges that are not part of the boundary must be removed. The approach for finding the initial Minkowski sum edge list will be first illustrated by an example and then the algorithmic procedure will be given. Removal of internal edges will be addressed in the next section.

Consider the example previously given in Fig. 3. If we follow the precedence of A, traversing from a2 to a3 we will encounter b4 before b3, yet the previous B edge encountered had been b2. The equivalent conflict would occur in A if we followed the precedence of B. However, if we break polygon B at the vertex connecting b3 and b4, and consider the edges as a list, instead of a cycle, starting from b4, then we have **b4**, **b5**, **b1**, **b2** and **b3** in a continuous counter clockwise direction. As a result, all the edge points B on the slope diagram are in the correct order, and b3, at this time, has no connection to b4. Having made this break, the procedure can be described as one of searching for the next B edge on the list through following the precedence path of A. Hence, only the next B edge is active, all others are dormant. Since the B edges may be visited more than once, it is first necessary to perform an initial counting phase. This is achieved through an exploratory cycle of the merged slope order, following the precedence of A and counting the number of times a B edge is traversed.

The approach applied to the example in Fig. 3 works as follows. Start from the first B edge, b4, we search for b5. To find this we traverse a3 and a4. From b5 we search for b1 and traverse a5. From b1 we search for b2 and traverse a1. Finally we search for b3 traversing a2. Since b3 crosses the concavity of A, it will appear three times with negative direction when it is encountered on a clockwise traversal of the slope diagram. This was established through the initial counting phase. Hence the search continues until all appearances of b3 have been found. Thus we obtain **b4**, a3, a4, **b5**, a5, **b1**, a1, **b2**, a2, **b3**, a3, −**b3**, a4, **b3**. Given that a polygon must be a complete cycle, we must now link the beginning and the end of the list. Hence from b3 we will look for b4. This requires a clockwise turn through the slope diagram traversing −a4 and −a3. Thus finally we obtain: **b4**, a3, a4, **b5**, a5, **b1**, a1, **b2**, a2, **b3**, a3, −**b3**, a4, **b3**, −a4, −a3.

In summary, the procedure to form the sequence follows the slope diagram of A positively when the series of B edges are in a counter clockwise direction and follows it negatively when the series of B edges are in a clockwise direction. With this knowledge, we consider a more complex case.

In Fig. 5, there is more than one concave point in polygon A and B. Sorting A and B into slope order, merging the lists and following the precedence of A, we discover that all B edges will be traversed three times with the exception of b7 and b11 which will be traversed five times. Further we know the direction in which they are traversed; positive or negative. Given the groups will be linked, we wish to finish a group moving forward in a counter clockwise direction, equivalent to a positive B edge. The edge points of B can be divided into the following five groups according to their appearance in consecutive counter clockwise direction or clockwise direction on the slope diagram.

1. b12, b1, b2 (counter clockwise);
2. b3, b4 (counter clockwise);
3. b5, b6, b7 (counter clockwise);
4. b8 (clockwise);
5. b9, b10, b11 (counter clockwise).

For each group we follow the precedence of A searching for the next B edge in the sequence. For example for group 1, we begin with the first B edge on the list at the occurrence that follows a1. This ensures we end on a positive B edge, i.e we have not broken the +, −, + sequence of b12. The next B edge is b1, hence we traverse b12, a2, a3, −b12, a4, b12, a5, b1, a6. Note that the admissible B edges that can be included are either −b12 or b1, if we had encountered other B edges on route to b1, they would have been ignored. This can be observed in other groups. Next, we search for

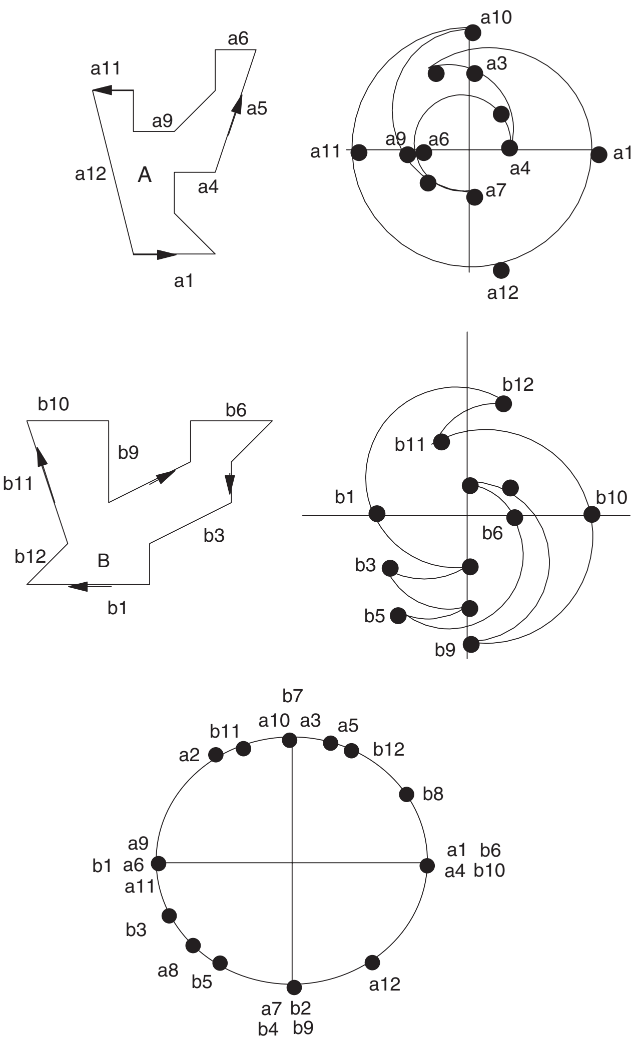

Fig. 5. Complex case of two polygons with more than one concavity in each polygon.

b2, which is encountered directly after b1. Although, b12, b1 and b2 have been found, we know we must traverse each three times, hence the search continues through a7, −b2, a8, −b1, a9, a10, b1, a11, b2. In order to link each group, some additional *A* edges need to be added. For group 1 the final *A* edge is a12 and the first *A* edge in group 2 is a7, hence we must return through −a12 to −a7 to retain the precedence order. The full edge list is detailed below, where the *A* edge that precedes the starting *B* edge is included in square brackets to indicate the starting point, but is not part of the edge list. The linking edges are underlined.

1. [a1], **b12**, a2, a3, −**b12**, a4, **b12**, a5, **b1**, a6, **b2**, a7, −**b2**, a8, −**b1**, a9, a10, **b1**, a11, **b2**, −a11, −a10, −a9, −a7;
2. [a6], **b3**, **b4**, a7, −**b4**, a8, −**b3**, a9, a10, a11, **b3**, **b4**, −a11, −a10, −a9, −a7;
3. [a6], **b5**, a7, −**b5**, a8, a9, a10, a11, **b5**, a12, **b6**, a1, **b7**, a2, −**b7**, a3, −**b6**, a4, **b6**, a5, **b7**, a6, a7, a8, a9, −**b7**, a10, **b7**, −a10, −a9, −a7, −a6, −a5;
4. [−a5], **b8**, −a4, −**b8**, −a3, −a2, **b8**, a2, a3, a4, a5;
5. [a6], **b9**, a7, −**b9**, a8, a9, a10, a11, **b9**, a12, **b10**, a1, **b11**, a2, −**b11**, a3, −**b10**, a4, **b10**, a5, **b11**, a6, a7, a8, a9, −**b11**, a10, **b11**, −a10, −a9, −a7, −a6, −a5, −a4, −a3, −a2.

The procedure to find the Minkowski sum edge list of two polygons A and B is given below. Note that if a group of B edges have continuous counter clockwise order, it is labelled as *positive*, otherwise it is labelled as *negative*. A positive group traverses the slope diagram of A in counter clockwise order including positive A edges. A negative group follows the slope diagram in clockwise order including negative A edges. Only the positive procedure is included here. For efficiency, the procedure in Mink(Q, R, positive) traverses the slope order list following the precedence of A recording the B edges encountered along the way to provide list, s. This process allows us to know the number of times each B edge is encountered and in what direction. Further list s can be used to create the correct sequence of A and B edges without searching the slope order a second time.

Algorithm 1. Algorithm to generate the Minkowski Sum edge list

Step 1: Replace B by $-B$, i.e. replace all co-ordinates (x_B, y_B) of B by $(-x_B, -y_B)$.

Step 2: Starting at the lowest point on each polygon. Label the edges in counter clockwise order.

Calculate the angle $\theta(i)$ of each edge, i, from the horizontal in a counter clockwise direction.

For each edge i, let $\alpha(i) = \theta(i) - \theta(i-1)$.

If $\alpha(i) > 180$ then $\alpha(i) = \alpha(i) - 360$.

If $sign(\alpha(i)) \neq sign(\alpha(i-1))$ mark i as a turning point.

If any turning points have been detected then polygon is non-convex.

Sort the edges into angle order to form sort_list(P), where $P = A$ or $-B$.

Step 3: Let $(b_{k_1+1}, \ldots, b_{k_n+1})$ be the set of turning points on the slope diagram of polygon B. Then polygon B can be divided into groups $B_1 = (b_{k_1+1}, b_{k_1+2}, \ldots, b_{k_2})$, $B_2 = (b_{k_2+1}, b_{k_2+2}, \ldots, b_{k_3}), \ldots, B_n = (b_{k_n+1}, b_{k_n+2}, \ldots, b_{k_1})$ according to them being consecutive counterclockwise direction or consecutive clockwise direction.

Step 4: For each group B_j, $(j = 1, \ldots, n)$, call Mink(A, B_j, *positive*) or Mink(A, B_j, *negative*) according to group B_j being counter clockwise or clockwise, respectively. We obtain Seq_j.

Step 5: Link Seq_list(A, B_1), ..., Seq_list(A, B_n) with additional A edges one by one.

If Seq_list(A, B_j) is positive, insert negative A points to link Seq_i with Seq_{i+1}.

else Seq_list(A, B_j) is negative, insert positive A points to link Seq_i with Seq_{i+1}.

Mink(Q, R, *positive*).

Step 1: Merge sort_list(Q) and sort_list(R) to form merge_list(Q, R)

Step 2: Set $i = 1$, $k = 1$, direction $= 1$, $s_1 = q_1$

Step 3: Set $i = i + 1$

Search merge_list(Q, R) for q_i moving forward if $direction = 1$ and backwards if $direction = -1$

if R edge, r_j, set $k = k + 1$, $s_k = direction \times r_j$

When q_i is encountered, if $i = 1$, go to step 4

Otherwise set $k = k + 1$, $s_k = q_i$

If q_i is a turning point in Q, set $direction = -direction$

Repeat step 3

Step 4: Let starting edge r_1 be in position s_i in sequence

Set $j = 1$, $next = 2$, $direction = 1$, $seq_1 = s_i$

Step 5: Set $i = i + 1$, if $i > k$, set $i = 1$

If s_i is from Q, $j = j + 1$, $seq_j = s_i$

If s_i is a turning point in Q, $direction = -direction$, $next = next + direction$

If $s_i = \text{direction.r}_{\text{next}}$, $j = j + 1$, $seq_j = s_i$, $next = next + direction$

If all s_i edges have been allocated to seq_j, return seq_1 to seq_j as Seq_list(Q, R)

Otherwise, repeat step 5

The computational complexity of Algorithm 1 is O(mn) where m and n are the edges of polygon A and polygon B, respectively [8].

5. Computing the boundary of the NFP

The resulting Minkowski sum edge list forms a complex (self crossing) polygon where the edges include all the edges of the NFP and some internal points. In this section we describe a new method for identifying the true edges of the NFP.

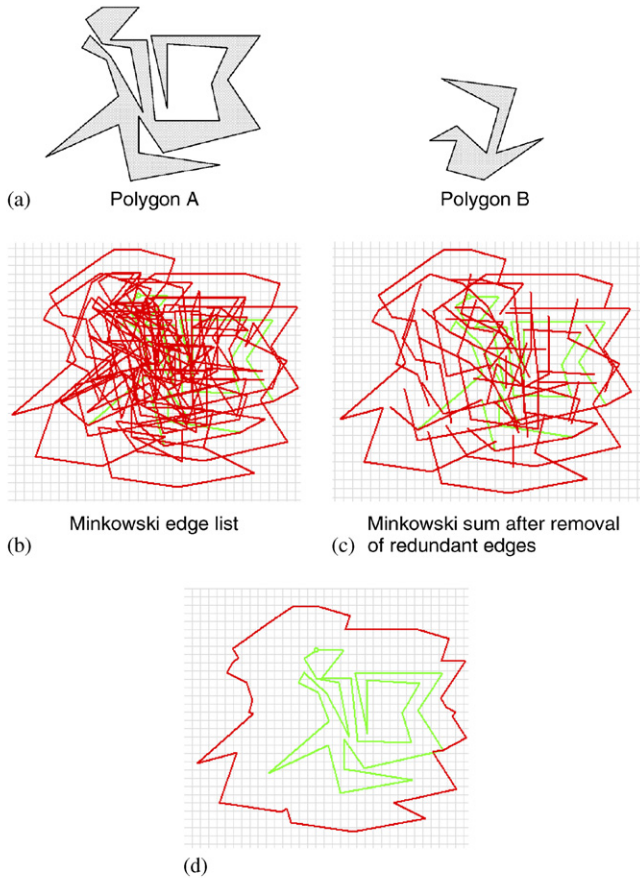

Fig. 6. A large example of the Minkowski sum edge list and track line traces.

Fig. 6b illustrates the outcome of the Minkowski sum edge list procedure described in the previous section for two simple polygons drawn in 6a. Clearly there are many edges to untangle. However, using some of the properties we know about the NFP and the boundary addition method we can quickly remove many of these edges.

Property 1. As detailed earlier in the paper, the NFP can be described as the path of a reference point on B as it slides around the edges of A. Further, the sliding motion is always in the same direction (i.e. counter clockwise) and as a result the path mapped by the reference point on B is also in one direction.

Property 2. The slope diagram representation, used in boundary addition, indicates that if edge b_j appears between edges (a_i, a_{i+1}), this corresponds to the physical condition that edge b_j slides along vertex (a_i, a_{i+1}). When the direction from a_i to a_{i+1} is counter clockwise and edge b_j is positive, the corresponding sliding condition is that the convex vertex (a_i, a_{i+1}) slides along b_j. Otherwise, the negative edge b_j corresponds to the condition that the concave vertex (a_i, a_{i+1}) slides along edge b_j. Clearly an edge cannot slide along a concave vertex without creating overlap between the polygons.

It can be deduced from these two properties that any negative edges cannot be part of the boundary of the NFP and can be removed. Further, the linking edges between sequences can also be removed since their inclusion is to define the correct starting position of the next sequence and they do not represent potential sliding between the polygons. Fig. 6c illustrates the Minkowski edge list with the redundant edges removed. With this understanding, we can develop intuitively a new method to identify the boundary of the NFP by only considering useful parts of the derived Minkowski sum edge list.

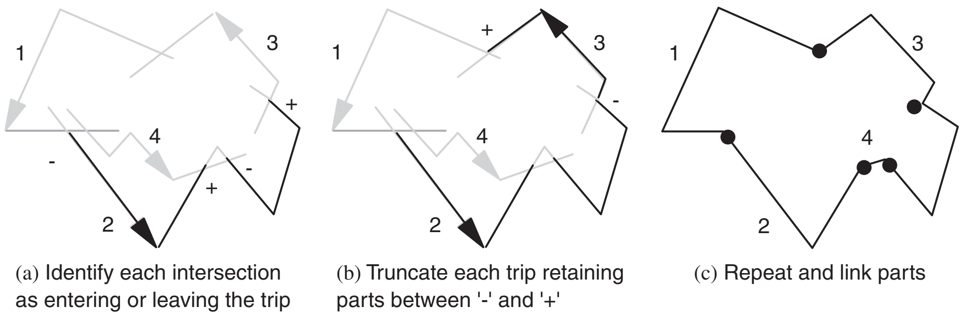

Fig. 7. Procedure for removing internal edges from the Minkowsi sum: (a) identify each intersection as entering or leaving the trip; (b) truncate each trip retaining parts between '−' and '+'; (c) repeat and linkparts.

In order to introduce this method, we recall briefly some terms introduced by Ramkumar [23]. A *state* is a pair consisting of a position $\dot{s}$ in the plane and a direction $\vec{s}$. A *move* is a set of states with constant direction and position varying along a line segment parallel to the direction. A *turn* is a set of states with constant position and direction varying along an arc of the circle of directions. A *polygonal trip* consists of a continuous sequence of moves and turns; the trip is closed if it starts and ends at the same state. A polygonal tracing is a collection of closed polygonal trips. We think of each loop of a tracing as being traversed by a car which always faces in the direction of the state it is currently following.

The first step is to break up the Minkowski sum edge list into *polygonal trips* according to the continuous sequence of moves and turns. Those that cannot be part of the boundary of the NFP, according to the properties described above, are discarded. This is equivalent to removing the negative and additional A edges.

The second step of the procedure requires the identification of all the intersection points between the polygonal trips. Each intersecting point is marked with a '−', indicating it is *entering* the trip, or with a '+', indicating it is *leaving* the trip. Consider the example in Fig. 7a, where the current trip is trip 2. Imagine standing at the beginning of the first edge of trip 2 facing along the edge, then we can consider that trip 1 intersects trip 2 from right to left. Trip 1 is said to *enter* trip 2 and is marked with '−'. Continuing along trip 2 we eventually meet an intersection with trip 4, where trip 4 intersects from left to right. Trip 4 is *leaving* trip 2 and marked with '+'. Hence, entering a trip means the beginning part of the intersecting edge is on the right side of the trip edge and leaving corresponds to the beginning part of the intersecting edge is on the left side of the trip edge. The fragments of trips that span a '−' intersection to a '+' intersection are kept to form the boundary of the NFP. All other fragments are discarded.

Finally, fragments that share end points are linked. Only those parts, which form a cycle, can be taken as the boundary of the NFP. In the case where cycles represent holes in the NFP, we implement a simple direct test of whether the two component polygons overlap when the reference point of B is located on a vertex of the hole.

The algorithm for finding the boundary of the NFP can be summarized in Algorithm 2 and 3 as below:

Algorithm 2. Algorithm for breaking Minkowski edge list into track line trips:

Step 1: Let $i = 0$ be the index number of Minkowski sums obtained. Let $j = 0$ be the number of track line trips, n_j be the total edges in track line trip j and $k = 0$ be the index of each track line trip.

Step 2: Search forward in Minkowski edge list for positive s_i, which corresponds to a track line. If s_i can be found, set $t_{jk} = s_i$, else go to *Step* 4.

Step 3: Set $i = i + 1$ and $k = k + 1$. If s_i is positive and corresponds to a track line, repeat step 3,
else set $n_j = k$, $j = j + 1$ and $k = 0$, repeat Step 2.

Step 4: Return t_{j0} to t_{jn_j} as track line trip T_j.

The computational complexity of Algorithm 2 is O(N), where N is the number of edges of Minkowski sum edge list. In the worst case $N = mn$.

Algorithm 3. Algorithm for finding the boundary of the NFP from track line trips:

Step 1: Let trip T_j contain n_j segments. Let seg_{jk_j} $(k_j = 1, 2, \ldots, n_j)$ be the k_jth segment of T_j.

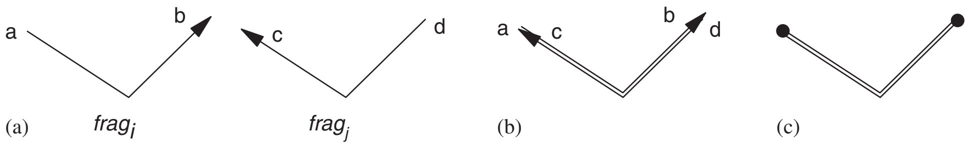

Fig. 8. Coinciding fragments to indicate exact slide in NFP.

For all i and j where $i \neq j$,

If seg_{jk_j} intersects seg_{ik_i}, let p_{jr} $(r = 1, 2, \ldots, w_j)$ be the rth intersection points on T_j , where w_j is the total number of intersection points on T_j. Set $pos_{jr} = k_j$.

If on intersection point p_{jr}, segment seg_{ik_i} crosses right to the left of seg_{jk_j}, set $sign_{jr} = -1$, else set $sign_{jr} = +1$.

Step 2: If $sign_{jr} = -1$ and $sign_{jr+1} = +1$. Store all segments between the intersection points, $\{p_{jr_j}, s_{jpos_{jr}+1}, \ldots, s_{jpos_{jr+1}}, p_{jr_j+1}\}$, into $frag_i = \{f_{i1}, f_{i2}, \ldots, f_{il_i}\}$, where l_i is the total number of segments in $frag_i$ $(i = 1, 2, \ldots I)$, where I is the total number of fragments obtained.

Step 3: For all $frag_i \neq 0$, we

if $f_{i1} = f_{jl_j}, frag_i = frag_j + frag_i = \{f_{j1}, \ldots, f_{jl_j} \ldots, f_{il_i}\}, frag_j = 0$.

if $f_{j1} = f_{il_i}, frag_i = frag_i + frag_j = \{f_{i1}, \ldots, f_{il_i} \ldots, f_{jl_j}\}, frag_j = 0$.

if $f_{i1} = f_{jl_j}$ and $f_{j1} = f_{il_i}$ then form $cycle_k$ $(k = 1, 2, \ldots, K)$, where K is total number of cycles obtained and set $frag_i = frag_j = 0$

Repeat step 3 until all $frag_1 = 0$.

Step 4: For each $cycle_k$.

Locate reference point of polygon B on one of the vertexes of $cycle_k$.

Discard the $cycle_k$ if the two polygons overlap

The computational complexity of Algorithm 3 is $O(X + Y \log Y)$, where X is the number of intersections of all the segments on all trips T_j $(j = 1, 2, \ldots)$ and Y is the number of all the segments on all trips T_j [24]. In the worst case $X = mn$ and $Y = m^2n^2$, where m and n are the edges of polygon A and polygon B, respectively [14].

It is important to note that the above algorithm is able to identify the outer face of the Minkowski sum, holes inside the outer face, a single point that represent an exact fit, and exact slides represented as a single line. The latter two are often referred to as degenerate cases.

5.1. *Degenerate cases*

The degenerate cases, in general, refer to combinations of polygons that can fit together like a jigsaw, resulting in a single point of fit within the NFP, or where one piece can slide into or within a concavity in one direction only, resulting in a line either extending from the edge of the NFP or within.

An exact slide can be identified when two fragments, obtained from Algorithm 3, step 2, coincide. Fig. 8a illustrate $frag_i$ and $frag_j$ with start points, a and d, and end points, b and c, respectively. If the start and end points from each fragment coincide, as shown in Fig. 8b they can be linked into a cycle (Fig. 8c) in step 3. The cycle is validated as part of the boundary of the NFP in step 4.

In the case of a single point, further calculation is required in Algorithm 3, step 1. When all the intersection points between segments of the trip are calculated, we also identify special intersection cases as shown in Fig. 9, where trips T_i and T_j intersect each other at point A. It is necessary to test all intersection points of this type to identify if it is an exact fit. In our experience, intersection points such as A seldom occurred within the intersection condition.

6. Empirical analysis

In order to evaluate the effectiveness of our new approach, we generated the NFP for all pairs of polygons for the benchmark data sets found on the ESICUP [15] (European study group on cutting and packing) website. Each data set contains a number of simple polygons and are used as the common test sets in the cutting and packing community.

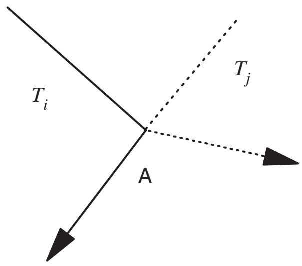

Fig. 9. Edge intersection to indicate exact nesting point in NFP.

Table 1
Generation times of all NFPs in benchmark data sets

No.	Case	No. of piece types	Ave no. of edges	Time (s)
1	Albino_hopper	8	7.25	0.07
2	Blaz_topos	7	6.28	0.04
3	Dagli_hopper	10	6.3	0.07
4	Dighe_hopper	10	4.7	0.04
5	Dighe_hopper-1	16	3.8	0.07
6	Fu_hopper	12	3.6	0.04
7	Han_hopper	20	6.95	0.33
8	Jakobs_hopper	25	5.8	0.42
9	Mao_hopper	9	9.2	0.15
10	Marques_hopper	8	7.1	0.07
11	Poly_hopper	75	4.8	2.00
12	Shapes_topos	4	8.7	0.03
13	Shirts_topos	8	6.6	0.06
14	Swim_topos	10	22.8	0.93
15	Trousers_topos	17	5.1	0.10

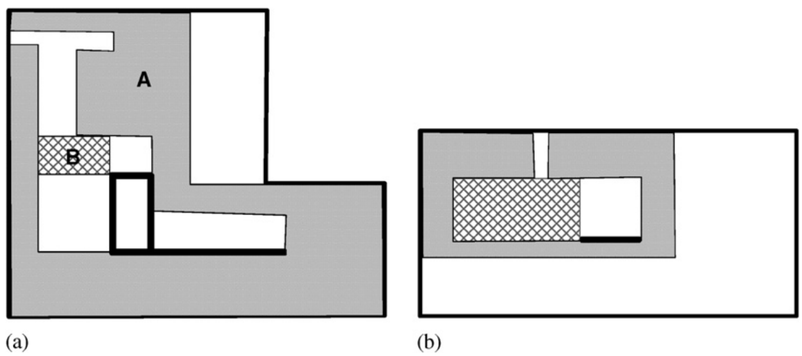

Fig. 10. (a) The nofit polygon contains a hole and an exact slide along the bottom edge of the concavity; (b) the nofit polygon contains an exact slide.

All were generated correctly and the computation times for every combination of each data set are provided in Table 1. The procedure was coded in Visual Studio C++ and the instances were run on a pc with 512 MB, 1.6 GHz.

In addition we have extensively tested our approach on new instances designed to involve characteristics such as, a large number of edges, interlocking positions, exact sliding, jigsaw type fits, and concavities the turned more than 360°. All NFPs were successfully generated, none taking more than one second to generate. The results can be found in the Figs. 10–13.

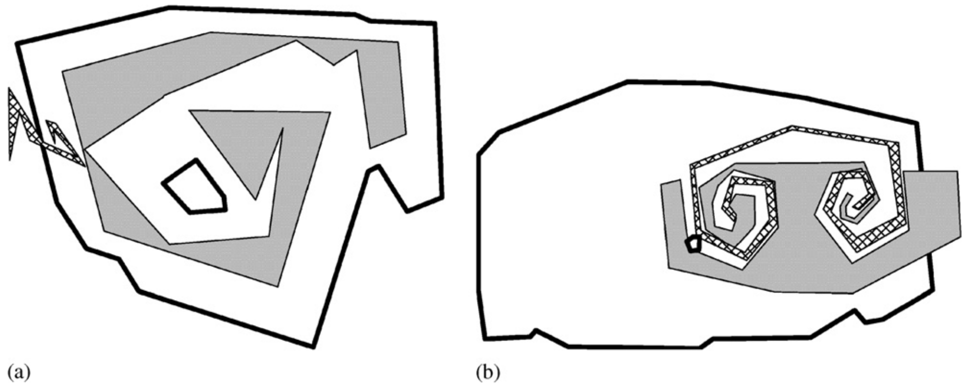

Fig. 11. (a) The nofit polygon contains a hole. The concavity in polygon A turns through more than 360°; (b) nofit polygon contains a hole. Both A and B have concavities that turn greater than 360° and interlock with each other.

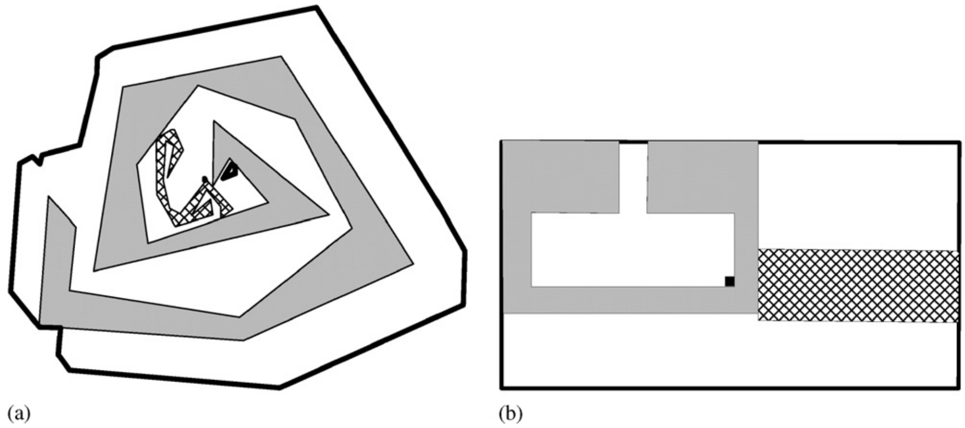

Fig. 12. (a) The nofit polygon contains a hole and an exact fit indicated by the illustrated position of polygon B; (b) Polygon B fits exactly in the concavity of polygon A.

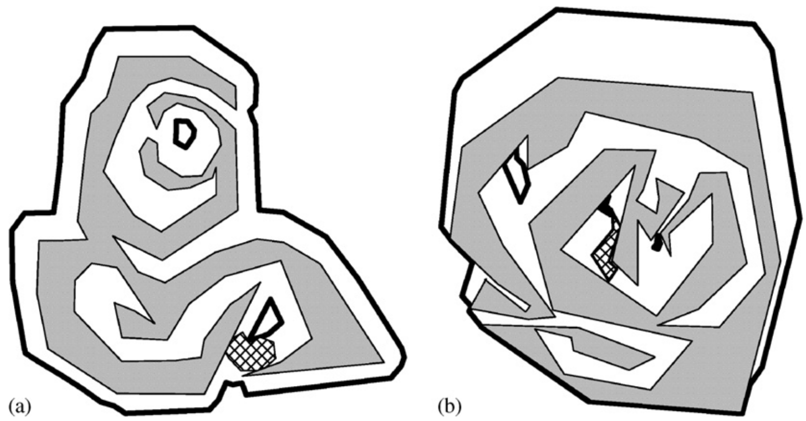

Fig. 13. Both nofit polygons contain multiple holes. Polygon A has concavities within concavities.

7. Conclusions

In this paper we have described a new approach of finding the nofit polygon. Empirical analysis demonstrates that computational times are realistic and that the approach is robust in dealing with known degenerate cases and new difficult cases such as spiraling concavities. The method is theoretically underpinned by the concept of Minkowski sums and builds on the work of Ghosh by adapting his boundary addition theorem into an algorithm procedure. It improves on the work of Bennell, Dowsland and Dowsland by finding the Minkowski sum in a single procedure and removing any ambiguity over which edges should be included in the repair procedure. Finally the paper provides a new, simple and robust procedure for the removal of internal edges and identification of holes. All approaches are described in detail, illustrated by example and the summary code is provided.

Acknowledgments

The authors would like to acknowledge the financial support of the Engineering and Physical Sciences Research Council.

References

[1] Adamowicz M, Albano A. Nesting two dimensional shapes in rectangular modules. Computer Aided Design 1976;8(1):27–33.
[2] Mehadevan A. Optimization in computer aided pattern packing. PhD dissertation, North Carolina State University; 1984.
[3] Konopasek M. Mathematical treatments of some apparel marking and cutting problems. US Department of Commerce Report; 1981. 99-26-90857-10.
[4] Oliveira JF, Ferreira JS. Algorithms for nesting problems. In: Vidal RVV, editor. Applied simulated annealing. Lecture notes in economics and maths systems, vol. 396. Berlin: Springer, p. 255–74.
[5] Babu RA, Babu RN. A genetic approach for nesting of 2D parts in 2D sheets using genetic and heuristic algorithms. Computer-Aided Design 2001;33:879–91.
[6] Wäscher G, Haußner H, Schumann H. An improved typology of cutting and packing problems. Working Paper. Faculty of Economics and Management, Otto von Guericke University, Magdeburg; 2005.
[7] Minkowski H. Volumen und Oberflache. Mathematische Annalen 1903;57:447–95.
[8] Ghosh PK. A unified computational framework for Minkowski operations. Computers and Graphics 1993;17(4):357–78.
[9] Ghosh PK. An algebra of polygons through the notion of negative shapes. CVGIP: Image Understanding 1991;54(1):119–44.
[10] Bennell JA, Dowsland KA, Dowsland WB. The irregular cutting-stock problem—a new procedure for deriving the nofit polygon. Computers and OR 2001;28:271–87.
[11] Bennell JA. Incorporating problem specific knowledge into a local search framework for the irregular shape packing problem. PhD dissertation, EBMS. University of Wales, Swansea, UK; 1998.
[12] O'Rourke J. Computational geometry in C. 2nd ed., Cambridge: Cambridge University Press; 1998.
[13] Cunninghame-Green R. Geometry. Shoemaking and the milk tray problem. New Scientist 1989, 1677: (12th August); 50–3.
[14] Kaul A, O'Connor MA, Srinivasan V. Computing Minkowski sums of regular polygons. Proceedings of the third canadian conference on computational geometry, British Columbia: Vancouver; 1991. p. 74–7.
[15] deBerg M, van Kreveld M, Overmars M, Schwartzopf O. Computational geometry: algorithms and applications. Berlin: Springer; 2000.
[16] Burke EK, Hellier RSR, Kendall G, Whitwell G. Complete and robust no-fit polygon generation for the irregular stock cutting problem. Working paper, ASAP, School of Computer Science, University of Nottingham, UK; 2005.
[17] Watson PD, Tobias AM. An efficient algorithm for the regular W1 packing of polygons in the infinite plane. Journal of the Operational Research Society 1999;50:1054–62.
[18] Li Z, Milenkovic VJ. Compaction and separation algorithms for non-convex polygons and their applications. European Journal of Operational Research 1995;84:539–61.
[19] Agarwal PK, Flato E, Halperin D. Polygon decomposition for efficient construction of Minkowski sums. Computational Geometry Theory and Applications 2002;21:39–61.
[20] Stoyan Y, Terno J, Scheithauer G, Gil N, Romanova T. Phi functions for primary 2D-objects. Studia Informatica Universalis 2001;2(1):1–32.
[21] Stoyan Y, Scheithauer G, Gil N, Ramanova T. Phi-functions for complex 2D-objects. Technical Report, MATH-NM-2-2002; April 2002, Technische Universitat Dresden.
[22] ESICUP. European working group on cutting and packing. ⟨http:/www.apdio.pt/sicup⟩, 2005.
[23] Ramkumar GD. An algorithm to compute the Minkowski sum outer face of two simple polygons. Proceedings of the 12th annual symposium on computational geometry. 1996. p. 234–41.
[24] Chazelle B, Edelsbrunner H. An optimal algorithm for intersecting line segments in the plane. Journal of Association in Computing 1992;39: 1–54.