ELSEVIER European Journal of Operational Research 84 (1995) 522–538

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

A new algorithm for the minimal-area convex enclosure problem

Roger B. Grinde [a,*], Tom M. Cavalier [b]

[a] *Decision Sciences Department, The Whittemore School of Business and Economics, The University of New Hampshire, Durham, NH 03824, USA*

[b] *Department of Industrial and Management Systems Engineering, The Pennsylvania State University, University Park, PA 16802, USA*

Abstract

The problem of cutting parts from a piece of material occurs in a number of settings. When the pieces to be cut are non-rectangular, the problem is called the irregular pattern layout problem (or nesting problem). This problem occurs in sheet metal and apparel industries, for example. One possible approach is to combine individual pieces into low-waste 'modules' which are then arranged by a heuristic technique. In this spirit, the Minimal-Area Convex Enclosure Problem is solved in this paper. Given two simple polygons, the algorithm finds the relative position of one with respect to the other such that the area of their convex enclosure is minimized. The technique searches along the envelope (or no-fit-polygon), dynamically maintaining the convex enclosure vertices as well as an analytic representation of the area. The algorithm runs quickly and computational experience is included.

Keywords: Optimization; Nesting problem; Computational geometry; Manufacturing industries; Algorithm

1. Introduction

The problem of cutting parts from a piece of material occurs in a number of industrial settings. Often the parts to be cut are rectangular (e.g., standard cutting stock and bin packing problems). However, there are important cases in which the parts have irregular (i.e., non-rectangular) shapes. This *irregular pattern layout problem*, or *nesting problem*, arises in sheet metal and composites industries, as well as in apparel and leather products manufacturing.

A general statement of the irregular pattern layout problem is to find a minimum-waste arrangement of the pieces to be cut from the resource material. Pieces are not allowed to overlap. Beyond this, a particular application will determine additional characteristics, such as arranging the pieces on a fixed-width bolt of cloth while minimizing the layout length or the number of resource sheets used. If the resource material has a print or pattern, there may be relative positioning constraints imposed (e.g., stripes on one piece must match those on another). There are also times in which pieces can be oriented in a finite number of ways (e.g., 180° rotations are allowed) or when pieces can be rotated to any orientation.

The irregular pattern layout problem has been receiving increased attention. Some of the more recent journal/book articles are listed for the

[*] Corresponding author.

interested reader: Oliveira and Ferreira (1993), Poshyanonda and Dagli (1992), Chung et al. (1990), Amaral et al. (1990), and Dagli (1990). All published work on this problem to date is of a heuristic nature. It is possible to formulate the problem as a mathematical program (Grinde, 1993); however, the problem is very difficult to solve to optimality due to the computational time required.

A useful endeavor would appear to be the development of *building block* algorithms which can be integrated into an overall heuristic/optimization approach to solve the pattern layout problem. This paper presents an implemented algorithm in this spirit. Given two simple (not necessarily convex) polygons, it finds the relative position of the two figures that minimizes the area of the convex enclosure of their union (not allowing overlaps). This research extends and generalizes earlier work. Having found a good *module* containing two pieces, this module can be treated as a single piece and combined with other pieces. Ultimately, all the single pieces and modules must be arranged. However, achieving good relative positions of pieces before determining their absolute positions on the resource may allow better solutions to be developed.

Simpler versions of this *Minimal-Area Convex Enclosure Problem* have been addressed. Clustering irregular shapes in rectangular modules has been examined (Adamowicz and Albano, 1976). For a single polygon, all orientations that cause a side of the polygon to be parallel to the horizontal or vertical axis are found. For each orientation, the rectangular enclosure is calculated. This approach finds the minimal-area rectangular enclosure, as proven by Freeman and Shapira (1975). Pairwise clustering is more difficult. For fixed orientations, they use the vertices of the *no-fit-polygon* (NFP: concept introduced as the *envelope* by Art, 1966, and later termed the NFP) of one shape with respect to the other as the basis of the search. An 'extended search' is conducted about each vertex to see if the enclosure can be made smaller. The process can be repeated with different orientations. Clustering more than two shapes is more difficult because the ordering of clustering impacts the quality of solution. The authors present a simplified heuristic algorithm for multistage clustering.

Nesting two convex polygons is considered by Lee and Woo (1988). The goal is to find the convex enclosure of minimal area containing the two figures. A fixed number (i.e., not dependent on the number of sides of the figures) of orientations is allowed, and with this restriction, the algorithm runs in linear time in the total number of edges of the two polygons. Basically, one figure is moved around the boundary of the other, and area added to form the convex enclosure is maintained. The position minimizing added area is optimal. However, the algorithm only works for convex figures.

The minimal-area rectangular enclosure algorithm (Adamowicz and Albano, 1976) is limited because it does not find the minimal-area convex enclosure. Especially for intermediate steps in the pattern layout process, it may not be important that the figures are nested in rectangles. The minimal-area convex enclosure will have less waste for the same pair of enclosed figures. The minimal-area convex enclosure for two convex polygons (Lee and Woo, 1988) is limited because it considers only convex polygons as input. The algorithm in this paper generalizes these results.

Other related papers are briefly mentioned. Given a convex polygon, an algorithm to circumscribe it by a polygon of fewer sides with minimal area additions is developed (Dori and Ben-Bassat, 1983). Another algorithm takes as input an arbitrary closed curve and produces the minimum-area enclosing rectangle (Freeman and Shapira, 1975). They first find the minimum-perimeter enclosing convex polygon, and given this, they find the minimum-area rectangle. The major result is that the rectangle of minimum area enclosing a convex polygon has at least one side collinear with one of the edges of the polygon.

2. Problem statement and representation

The problem addressed is termed the Minimal-Area Convex Enclosure Problem. Simple polygons are assumed. Both polygons may be non-convex, but an optimal solution is guaran-

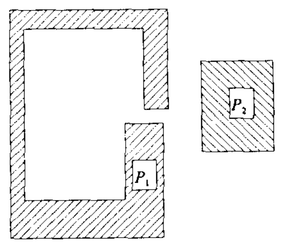


Fig. 1. Relatively multiply-connected figures.

teed only for 'relatively simply-connected' figures (terminology due to Adamowicz, 1969). Two figures can be relatively multiply-connected even if they are both simple polygons. Fig. 1 shows an example of two relatively multiply-connected figures. The main characteristic is that there is a 'channel' in P_1 which is too narrow for P_2 to fit through, but inside the channel there is an area large enough to 'contain' P_2. It is impossible to reach this area by sliding P_2 around P_1 if the polygons are never allowed to overlap.

A definition of the convex enclosure of two polygons is given, and then the problem is stated.

Definition. The *convex enclosure* (CE) of two non-overlapping polygons P_1 and P_2 with vertex sets V_1 and V_2 is the convex hull (CH) of the set of points $V_1 \cup V_2$. (The notation CE is used both as an abbreviation for 'convex enclosure' as well as the set containing the vertices of P_1 and P_2 in the convex enclosure.)

Minimal-Area Convex Enclosure Problem (MACEP). Given simple polygons P_1 and P_2 which are not relatively multiply-connected and which have fixed orientations, find the position of P_2 with respect to P_1 which minimizes the area of the convex enclosure of P_1 and P_2.

Let the vertices (in counter-clockwise order) of P_1 be given by

$$V_1 = \{v_{1j},\ j = 1,\dots,p1\}$$

and the vertices of P_2 by

$$V_2 = \{v_{2j},\ j = 1,\dots,p2\}.$$

Denote the convex hulls of P_1 and P_2 by $CH(P_1)$ and $CH(P_2)$. The convex enclosure of P_1 and P_2 can only include points from $CH(P_1)$ and $CH(P_2)$. An optimal solution will have the boundaries of P_1 and P_2 in contact, since a non-contact placement can be made no worse by bringing the figures closer together. Therefore, the search need only consider points on the 'no-fit-polygon' (NFP) or 'envelope', which is the set of non-overlapping placements of a reference point of one polygon with respect to another, where the borders of the two figures are in contact.

Polygons used for illustration throughout the paper are shown in Fig. 2. Fig. 3 displays the envelope of P_2 with respect to P_1. If the reference point of P_2 is placed inside this polygon, then the two figures overlap. The envelope is similar, but not identical, to the Minkowski Sum. This is defined for planar figures A and B as

$$A \oplus B = \{\boldsymbol{a} + \boldsymbol{b} \mid \boldsymbol{a} \in A,\ \boldsymbol{b} \in B\}.$$

See Guibas et al. (1983) for additional information.

P_1 is assumed fixed and P_2 *orbits* around P_1 along the envelope. That is, P_2 is the *orbiting polygon*, and P_1 is the *stationary polygon* (terminology due to Mahadevan, 1984). The reference point of each polygon is the vertex having the smallest y-coordinate. In case of ties, the

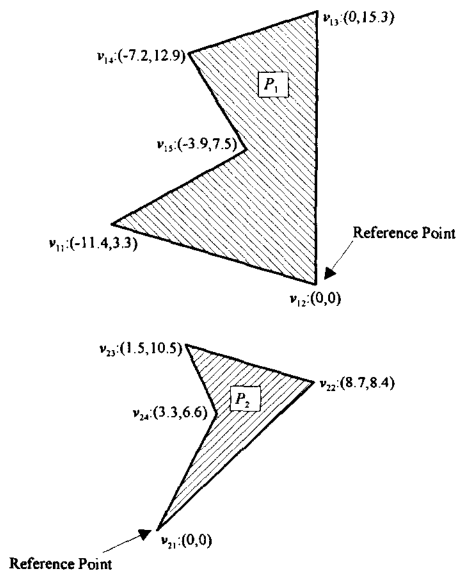


Fig. 2. Polygons used for illustration.

left-most bottom point is chosen. The envelope, denoted E, is the set of vertices (translations of the reference point of P_2) defining the no-fit-polygon,

$E = \{e_1, e_2, \ldots, e_s\}$.

As P_2 orbits around P_1, the point(s) of contact between the figures changes. As sliding occurs and a point of contact is maintained, it is necessary to detect when either a new contact point occurs, or the direction of P_2 changes. A previously developed envelope-generation code was used for this portion of the convex enclosure algorithm (Mahadevan, 1984). Changes were made to the data structures and computational improvements were made from this version, but the envelope-generation algorithm is not discussed in detail. The improvements are briefly mentioned in Section 7. For the rest of the paper, the envelope E is assumed to be an input.

The convex enclosure algorithm also uses a standard convex hull algorithm, Graham's Scan (Sedgewick, 1990). The convex hulls of P_1 and P_2 were used to improve the efficiency of the adapted envelope-generation code. Graham's Scan is also used to determine the initial vertices of CE.

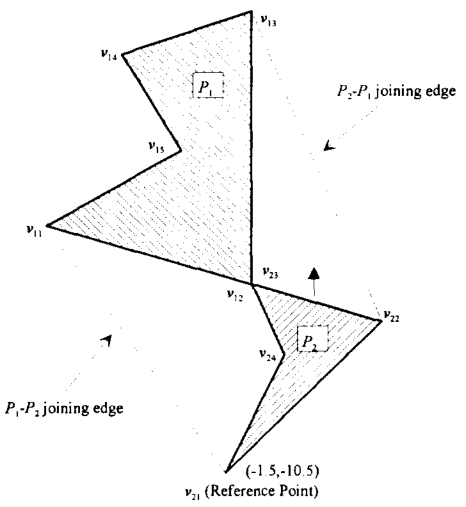


Fig. 4. Illustration of joining edges.

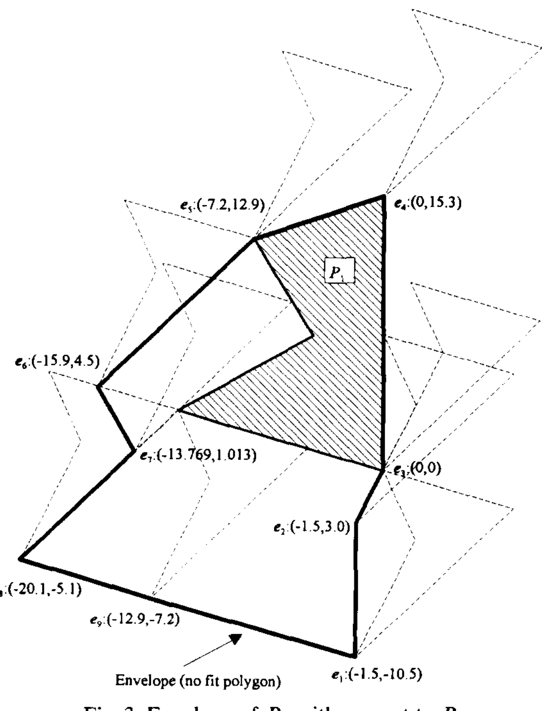


Fig. 3. Envelope of P_2 with respect to P_1.

Conceptually, the minimal-area algorithm needs to maintain an expression of the area of CE around E and find the minimal-area configuration. To do this, CE is maintained dynamically. In general, CE contains some vertices from P_1 and some from P_2 (a special case exists when P_2 fits completely inside $CH(P_1)$, or vice versa). Further, all vertices in CE from P_1 (P_2) will be 'contiguous'; that is, there will be no vertices from P_2 (P_1) interspersed throughout them. This leads to the concepts of *joining vertices* and *joining sides*.

Definition. A *joining side* is a side of CE with one endpoint a vertex of $CH(P_1)$, and the other endpoint a vertex of $CH(P_2)$.

Refer to Fig. 4. The convex enclosure is formed by the counter-clockwise (CCW) ordered vertices v_{21}, v_{22}, v_{13}, v_{14}, and v_{11}. In the CCW direction, one joining edge connects a vertex of P_2 (v_{22}) with a vertex of P_1 (v_{13}). This is the P_2-P_1 joining edge. The P_1-P_2 joining edge connects v_{11} with v_{21}. The four vertices involved in joining edges are called joining vertices.

As P_2 orbits around P_1, vertices of E will be encountered. However, the purpose of the algorithm is to find the area of CE, so any change in the vertices of CE must be detected. The vertices of CE may or may not change between envelope vertices. For example, in Fig. 4, v_{21} is a vertex of CE. However, as P_2 orbits around P_1, it will drop out of CE and later re-enter. Similarly, even though v_{12} is not currently a vertex of CE, it will enter as P_2 orbits. These intuitive notions are formalized in the next definition.

Definition. A *breakpoint* of CE is a point at which either the direction of P_2 changes, and/or a point at which the vertex set of CE changes. There are three types of breakpoints:

1. A vertex of E is encountered (direction of P_2 changes).
2. A vertex of P_1 or P_2 (not currently in CE) is inserted into CE.
3. A vertex of P_1 or P_2 (currently in CE) drops out of CE.

The next section illustrates the different breakpoint types and presents a property which limits the number of vertices that can cause breakpoints.

3. Illustration of breakpoints

The different breakpoint types are illustrated in the next four figures, beginning with Fig. 5.

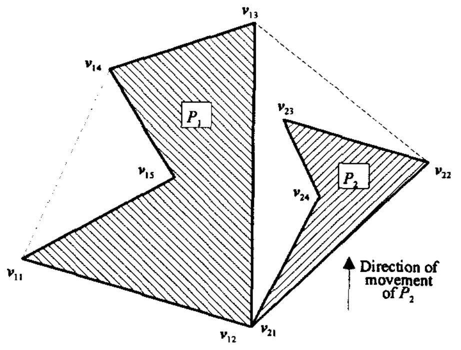


Fig. 5. Illustration of breakpoints: Envelope vertex.

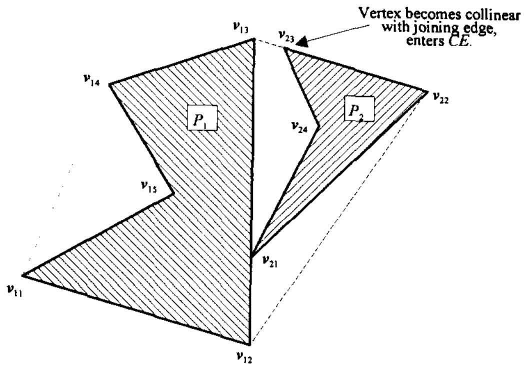


Fig. 6. Illustration of breakpoints: Type 2.

Note that

$$CH(P_1) = \{v_{11}, v_{12}, v_{13}, v_{14}\},$$

$$CH(P_2) = \{v_{21}, v_{22}, v_{23}\},$$

and that at Fig. 5 (which corresponds to a vertex of E),

$$CE = \{v_{12} \text{ (or } v_{21}), v_{22}, v_{13}, v_{14}, v_{11}\}.$$

As P_2 translates vertically, the P_1-P_2 joining edge will be (v_{12}, v_{22}) and the P_2-P_1 joining edge, (v_{22}, v_{13}). The problem is to detect when CE changes. This occurs when v_{23} becomes collinear with the P_2-P_1 joining edge (Fig. 6). At that instant, v_{23} is inserted into CE, where

$$CE = \{v_{12}, v_{22}, v_{23}, v_{13}, v_{14}, v_{11}\},$$

and the P_2-P_1 joining edge becomes (v_{23}, v_{13}). This is an example of breakpoint type 2.

P_2 continues to move vertically. The next breakpoint occurs when v_{14} becomes collinear

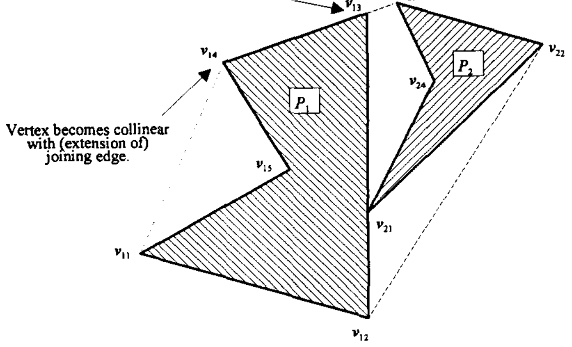


Fig. 7. Illustration of breakpoints: Type 3.

with (the extension of) the P_2-P_1 joining edge (Fig. 7). This is a type 3 breakpoint. Here, v_{13} is deleted from CE $(=\{v_{12}, v_{22}, v_{23}, v_{14}, v_{11}\})$ and the P_2-P_1 joining edge becomes (v_{23}, v_{14}). Continuing, P_2 is able to slide all the way to the next envelope vertex (Fig. 8). No changes in CE or the joining edges occur, but the direction of P_2 changes. This is a type 1 breakpoint.

Breakpoints can occur simultaneously. Normally this does not cause problems, but there are some special cases. These are mentioned in Section 5.

An important property has to do with which vertices are candidates to be 'entering' or 'leaving' vertices of CE. Only vertices of $CH(P_1)$ and $CH(P_2)$ are eligible to be in CE. However, further restrictions can be made based on a specific joining edge. For the P_1-P_2 joining edge, let the defining vertices be v_1(Join[1]) and v_2(Join[1]), where the subscript on v refers to P_1 or P_2, and the bracketed [1] refers to the P_1-P_2 joining edge. Similarly, let the vertices of the P_2-P_1 joining edge be v_2(Join[2]) and v_1(Join[2]). For example, in Fig. 6 (before the breakpoint is processed),

$$v_1(\text{Join}[1]) = v_{12}, \quad v_2(\text{Join}[1]) = v_{22},$$
$$v_2(\text{Join}[2]) = v_{22}, \quad v_1(\text{Join}[2]) = v_{13}.$$

The property that limits the search for entering and leaving CE vertices is stated here.

Property. *Given joining edges P_1-P_2 and P_2-P_1, the only vertices that can enter or leave CE are the joining vertices themselves, or the next and previous convex hull vertices relative to the joining vertices.*

For each joining vertex, the next and previous convex hull vertices could cause breakpoints. Therefore, only eight vertices can cause a breakpoint to occur on a given interval.

The breakpoints induced by these vertices do not have the same effect on CE and the joining

Table 1
Breakpoint cases

Case	Vertex producing breakpoint	effects
1	v_1(Join[1])(nextCH)	Type 2 breakpoint. v_1(Join[1]) stays in *CE*. v_1(Join[1])(nextCH) enters *CE*, becomes joining vertex.
2	v_1(Join[1])(prevCH)	Type 3 breakpoint. v_1(Join[1]) leaves *CE*. v_1(Join[1])(prevCH) becomes joining vertex (already in *CE*).
3	v_2(Join[1])(nextCH)	Type 3 breakpoint. v_2(Join[1]) leaves *CE*. v_2(Join[1])(nextCH) becomes joining vertex (already in *CE*).
4	v_2(Join[1])(prevCH)	Type 2 breakpoint. v_2(Join[1]) stays in *CE*. v_2(Join[1])(prevCH) enters *CE*, becomes joining vertex.
5	v_1(Join[2])(nextCH)	Type 3 breakpoint. v_1(Join[2]) leaves *CE*. v_1(Join[2])(nextCH) becomes joining vertex (already in *CE*).
6	v_1(Join[2])(prevCH)	Type 2 breakpoint. v_1(Join[2]) stays in *CE*. v_1(Join[2])(prevCH) enters *CE*, becomes joining vertex.
7	v_2(Join[2])(nextCH)	Type 2 breakpoint. v_2(Join[2]) stays in *CE*. v_2(Join[2])(nextCH) enters *CE*, becomes joining vertex.
8	v_2(Join[2])(prevCH)	Type 3 breakpoint. v_2(Join[2]) leaves *CE*. v_2(Join[2])(prevCH) becomes joining vertex (already in *CE*).

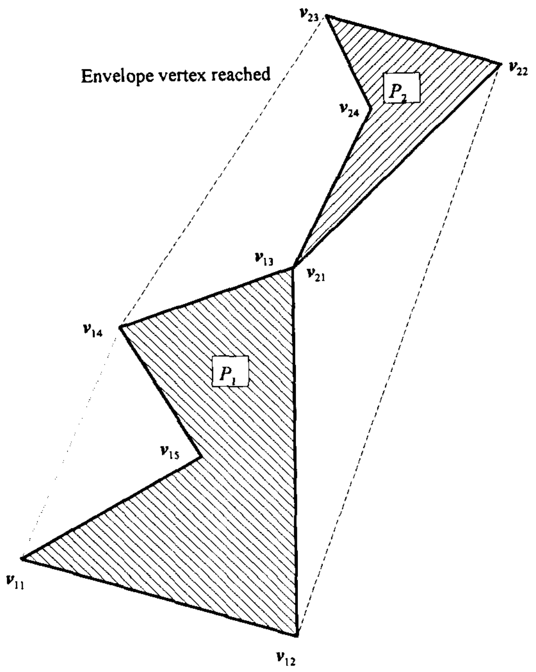


Fig. 8. Illustration of breakpoints: Type 1.

edges. The effects of each are listed in Table 1. In the table, the notation v_1(Join[1])(nextCH) refers to the next vertex of $CH(P_1)$ (in the CCW sense) relative to v_1(Join[1]). The other notation in the table has a similar interpretation. With this classification, Fig. 6 corresponds to case 7, and Fig. 7 to case 5.

The algorithm must find the distance P_2 slides before a breakpoint occurs. To do this, the joining edges are written in terms of the sliding direction of P_2, denoted $\boldsymbol{d}$, and the distance P_2 slides, denoted t. The direction vector $\boldsymbol{d}$ is normalized to be a unit vector, so t measures Euclidean distances. The distance to the next vertex of E is found, denoted t_{nfp}, where the subscript nfp is an acronym for 'no-fit-polygon'. Each case in Table 1 is analyzed to see if the breakpoint induced by it occurs before the next envelope vertex is encountered. The minimal positive distance (denoted $t_{\min}$) among all these cases determines the next breakpoint and the effects on CE, the joining edges, and the direction vector $\boldsymbol{d}$.

Suppose P_2 is placed at vertex $\boldsymbol{e}_l$ of E. Denote the next envelope vertex (in the CCW direction) by $\boldsymbol{e}_m$. Then t_{nfp} is given by

$$t_{\text{nfp}} = \left[(e_{m1} - e_{l1})^2 + (e_{m2} - e_{l2})^2\right]^{1/2}.$$

The unit direction vector $\boldsymbol{d}$ is

$$\boldsymbol{d} = (d_1, d_2) = \left(\frac{e_{m1} - e_{l1}}{t_{\text{nfp}}}, \frac{e_{m2} - e_{l2}}{t_{\text{nfp}}}\right).$$

Let t be the distance P_2 is slid in direction $\boldsymbol{d}$. If at least one breakpoint is found to occur with $t < t_{\text{nfp}}$, then vertices of the convex enclosure are added/deleted before reaching the next envelope vertex, and t_{nfp} is updated to reflect the remaining distance to $\boldsymbol{e}_m$. If all breakpoints have $t > t_{\text{nfp}}$, then P_2 slides all the way to the next envelope vertex $\boldsymbol{e}_m$, and in the next iteration P_2 will change directions. If $t = t_{\text{nfp}}$, then a breakpoint(s) occurs at the same time envelope vertex $\boldsymbol{e}_m$ is reached.

To find the values of t which cause breakpoints to occur, equations of the joining edges are written in terms of $\boldsymbol{d}$ and t. One can then find the value of t at which each of the eight possible cases occurs.

Consider the P_1-P_2 joining edge, and for notational convenience, let

$$\boldsymbol{v}_1(\text{Join}[1]) = \boldsymbol{v}_{1j} = (v_{1j1}, v_{1j2})$$

and

$$\boldsymbol{v}_2(\text{Join}[1]) = \boldsymbol{v}_{2k} = (v_{2k1}, v_{2k2}).$$

Further, let (x_1, x_2) be the coordinates of a vertex from P_1 or P_2 to be tested as to whether it causes a breakpoint. Then the equation to determine the value of t at which (x_1, x_2) becomes collinear with the joining edge is

$$\begin{aligned}(v_{2k2} + d_2 t - v_{1j2})x_1 - (v_{2k1} + d_1 t - v_{1j1})x_2 \\ - \left[v_{1j1}(v_{2k2} + d_2 t - v_{1j2}) \right. \\ \left. - v_{1j2}(v_{2k1} + d_1 t - v_{1j1})\right] = 0. \qquad (1)\end{aligned}$$

This equation is valid for a vertex from P_1 or P_2, but the coordinates of a vertex from P_2 also depend on $\boldsymbol{d}$ and t. For a vertex from P_1 (either

v_{1j}(nextCH) or v_{1j}(prevCH)), the critical value of t is

$$t=\frac{-b_2a_1+b_1a_2}{d_2a_1-d_1a_2} \tag{2}$$

where $\boldsymbol{a}$ and $\boldsymbol{b}$ are vectors with the following components:

$$a_1=x_1-v_{1j1},\quad a_2=x_2-v_{1j2},$$

$$b_1=v_{2k1}-v_{1j1},\quad b_2=v_{2k2}-v_{1j2}.$$

If the denominator in (2) is zero, then the vertex never becomes collinear.

Similarly, for a vertex of P_2, with coordinates $(x_1+d_1t,\ x_2+d_2t)$, the critical value of t is

$$t=\frac{-b_2a_2+b_1a_2}{c_1d_2-c_2d_1} \tag{3}$$

where $\boldsymbol{a}$ and $\boldsymbol{b}$ are defined as before, and $\boldsymbol{c}$ is defined as

$$c_1=x_1-v_{2k1},\quad c_2=x_2-v_{2k2}.$$

The formulas for the P_2-P_1 edge are similar. Letting

$$v_1(\text{Join}[2])=v_{1j}$$

and

$$v_2(\text{Join}[2])=v_{2k},$$

points (x_1, x_2) on this line satisfy the following equation:

$$\begin{aligned}&(v_{1j2}-v_{2k2}-d_2t)x_1-(v_{1j1}-v_{2k1}-d_1t)x_2\\&\quad-\big[(v_{2k1}+d_1t)(v_{1j2}-v_{2k2}-d_2t)\\&\qquad-(v_{2k2}+d_2t)(v_{1j1}-v_{2k1}-d_1t)\big]=0.\end{aligned}$$

For a vertex on P_1, the critical value of t is

$$t=\frac{-b_2c_1+b_1c_2}{a_1d_2-a_2d_1} \tag{4}$$

where $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ have the same form as before (but note that v_{1j} and v_{2k} are really different vertices for this joining edge). Similarly, for a vertex on P_2, the value of interest is

$$t=\frac{-b_2c_1+b_1c_2}{c_1d_2-c_2d_1}. \tag{5}$$

Using (2) through (5), the roots corresponding to the eight vertices in Table 1 are calculated, compared with t_{nfp}, and the appropriate actions taken.

4. Area of the convex enclosure

The area of *CE* can be found as a function of t for a given vertex set *CE* and direction $\boldsymbol{d}$. Whereas the paper dealing with the minimal-area enclosure of two convex polygons triangulated the area *added* by the convex enclosure (Lee and Woo, 1988), the algorithm presented here minimizes the *total* area of the convex enclosure. The areas of P_1 and P_2 are constant, so the approaches are equivalent. However, the added area method appears difficult to extend to the nonconvex case.

A cross-product expression of the area of a simple polygon is used (CRC, 1975). The formula assumes the vertices are in CCW order and are given by the set

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n), (x_{n+1}, y_{n+1})\},$$

where

$$(x_{n+1}, y_{n+1})=(x_1, y_1)$$

for convenience.

$$\text{Area}=\tfrac{1}{2}\sum_{i=1}^{n}(x_iy_{i+1}-x_{i+1}y_i). \tag{6}$$

As P_2 orbits around P_1, the vertices of P_2 move in direction $\boldsymbol{d}$, so it is possible to develop an expression of the area of *CE* between each pair of breakpoints. Theorem 1 shows that the area function is linear between breakpoints. As part of the proof, analytic expressions of the area are found.

Theorem 1. *As P_2 orbits around P_1 along E, the area of CE is piecewise-linear in the total distance slid, with slope changes occurring at breakpoints.*

Proof. Between breakpoints, vertices of *CE* and $\boldsymbol{d}$ are constant. Only coordinates of the vertices of P_2 change. Let t be the distance P_2 is slid in direction $\boldsymbol{d}$. Since the end of one breakpoint interval is the beginning of the next, the area function is continuous in the total distance slid. It must be shown that each cross product term in (6) is linear between breakpoints. There are four cases to consider. In each, let $v_j = (x_j, y_j)$ be the coordinates of one vertex of CE, and let $v_k = (x_k, y_k)$ be the next vertex of CE in the CCW-sense.

Case 1. v_j and v_k are both vertices of P_1. Then v_j and v_k do not move, so

$$x_j y_k - x_k y_j$$

is constant.

Case 2. v_j is on P_1, and v_k is on P_2 (i.e., v_j and v_k define the P_1-P_2 joining edge). Then v_k has coordinates $(x_k + d_1 t, y_k + d_2 t)$, and the cross product term is

$$x_j(y_k + d_2 t) - (x_k + d_1 t) y_j$$
$$= (x_j y_k - x_k y_j) + (x_j d_2 - y_j d_1) t$$

which is linear in t.

Case 3. v_j is on P_2, and v_k is on P_1 (i.e., v_j and v_k define the P_2-P_1 joining edge). Then v_j has the coordinates $(x_j + d_1 t, y_j + d_1 t)$, and the cross product term is

$$(x_j + d_1 t) y_k - x_k(y_j + d_2 t)$$
$$= (x_j y_k - x_k y_j) + (-x_k d_2 + y_k d_1) t$$

which is linear in t.

Case 4. v_j and v_k are both vertices of P_2. Then the respective coordinates of v_j and v_k are $(x_j + d_1 t, y_j + d_2 t)$ and $(x_k + d_1 t, y_k + d_2 t)$. The cross product term is

$$(x_j + d_1 t)(y_k + d_2 t) - (x_k + d_1 t)(y_j + d_2 t)$$
$$= (x_j y_k - x_k y_j)$$
$$+ \left[(x_j - x_k) d_2 - (y_j - y_k) d_1\right] t$$

which is linear in t.

A special case occurs if the joining edges of CE disappear. This is a placement where, without loss of generality, P_2 is small enough to fit completely inside $CH(P_1)$ without overlapping P_1. In this case no vertices from P_2 are in CE, so the area function is constant, and the CE area is just the area of $CH(P_1)$. (The implemented algorithm detects this special case and correctly reports the convex enclosure area.) □

5. Implementation

This section presents the implementation of the algorithm. Pseudocode is first presented and discussed. Special cases and limitations are then addressed.

Data structures play an important role in the algorithm. The major ones are vertex arrays for the two polygons. Embedded in these arrays are circular, doubly-linked lists of the convex hull vertices of P_1 and P_2. Once calculated, these lists are static. Their primary use is to identify the vertices which can cause breakpoints. The other key data structure, also embedded in the vertex arrays, is a dynamic, circular, doubly-linked list of the vertices of CE.

A list of *enclosure* coordinates is generated as the code runs. Coordinates represent translations of the reference point of P_2. All of the envelope vertices will eventually be in this list, and any breakpoint encountered also generates a record. As a rule, any time the area function changes, a new enclosure record is created. Therefore, the algorithm produces a complete analytic description of the convex enclosure area.

Pseudocode for the algorithm is shown in Fig. 9. For the initial position of P_2, the vertices of CE are calculated. A standard convex hull algorithm (Graham's Scan) is used. The algorithm then performs a search along E to find the optimal placement.

For each envelope vertex, a new enclosure record is created. This record comprises coordinates of P_2's reference point, the constant and variable coefficients of the area function, the maximum distance that P_2 can be moved on the current interval, and a pointer to the next enclosure vertex. The direction $\boldsymbol{d}$ is not stored since it is implied from any two consecutive enclosure records. For the current placement, the distance remaining to the next envelope vertex, t_{nfp}, is calculated, as well as $\boldsymbol{d}$. The current vertices of CE combined with $\boldsymbol{d}$ are used to determine the

```
input/initialization
find envelope ($E$) of $P_2$ with respect to $P_1$
find beginning convex enclosure ($CE$)
for each vertex of $E$
        create enclosure vertex
        calculate sliding direction ($d$)
        calculate distance to next vertex of $E$ ($t_{nfp}$)
        find area function $A(t|d)$
        update BestArea
        while ($t_{nfp} > 0$)
                if joining edges exist
                then    calculate enclosure breakpoints ($t_{root}[1...8]$)
                        $t_{min} \leftarrow min\ \{t_{nfp}, t_{root}[1...8]\}$
                        for each breakpoint type (1...8)
                                if ($t_{root}[i] = t_{min}$)
                                then    update $CE$, joining vertices
                        if (joining edges exist and length(joining edge)=0)
                        then    modify $CE$, joining vertices
                else    find sliding distance to regain joining edges ($t_{root}$)
                        $t_{min} \leftarrow t_{nfp}$
                        if ($t_{root} > 0$ and $t_{root} \le t_{nfp}$)
                        then    $t_{min} \leftarrow t_{root}$
                                reset joining edges
                $t_{nfp} \leftarrow t_{nfp} - t_{min}$
                set distance limit field in enclosure vertex to $t_{min}$
                if ($t_{nfp} > 0$)
                then    create new enclosure vertex
                        update position of $P_2$
                        find $A(t|d)$
                        update BestArea
end
```

Fig. 9. Pseudocode for minimal-area convex enclosure algorithm.

area function $A(t\,|\,\boldsymbol{d})$ for the current interval. Between consecutive vertices of E, t_{nfp} is reduced by any amount that P_2 is slid. Thus, t_{nfp} serves as the control variable between envelope vertices.

Calculations are performed in the inner ($t_{\text{nfp}} > 0$) loop. It is normal for joining edges to exist. The only exception is if P_2 completely fits inside $CH(P_1)$. This is treated as a special case later in the discussion, and it is assumed without loss of generality that the area of $CH(P_2)$ is no greater than the area of $CH(P_1)$.

Potential breakpoint distances are calculated (see Table 1), and the minimum distance at which a breakpoint occurs is stored as $t_{\min}$. This may be the same as t_{nfp}, which implies that P_2 is able to slide all the way to the next envelope vertex. The effects of any breakpoints are implemented by updating CE and the joining edges.

However, another special case can occur. It is possible for the vertices of a joining edge to coincide; that is, the joining edge can have a length of zero. In this case, special precautions are taken to insure that the vertices in CE and the joining edges remain correct as P_2 orbits further. Detection and treatment of this case is discussed later.

After processing all breakpoints, t_{nfp} is reduced by $t_{\min}$. If t_{nfp} is still positive (i.e., have not reached the next envelope vertex yet), then a new

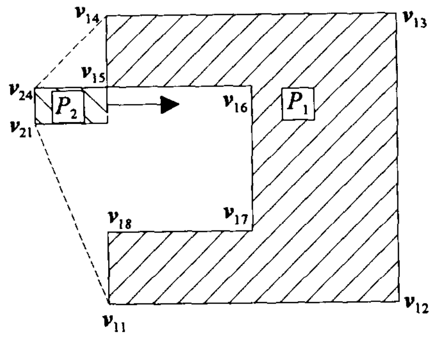


Fig. 10. Example for lost joining edges.

enclosure vertex is created, area coefficients are calculated, and the smallest area obtained so far is updated, if needed. If $t_{\text{nfp}} = 0$, then control returns to the outer loop to retrieve the next envelope vertex. The algorithm continues until all envelope vertices are exhausted.

Two special cases can occur, both involving the joining edges. The first occurs when P_2 can fit completely inside $CH(P_1)$. Refer to Fig. 10. As P_2 orbits around P_1, it moves inside $CH(P_1)$ and the joining edges disappear. As long as P_2 is inside $CH(P_1)$, the CE area remains constant, equal to the area of $CH(P_1)$.

While processing breakpoints, it must be detected when joining edges are 'lost'. Once lost, it must also be detected when joining edges are regained. In Fig. 10, the current joining edges are (v_{14}, v_{24}) and (v_{21}, v_{11}). As P_2 slides inward, P_2 moves completely inside the edge of $CH(P_1)$ defined by v_{14} and v_{11}. This is detected as follows: As P_2 slides inward and v_{21} and v_{24} line up with v_{14} and v_{11}, a breakpoint occurs (case 1 in Table 1). The fact that the next vertex of $CH(P_1)$, v_{11}, lies on the joining edge indicates that P_2 has moved completely inside $CH(P_1)$. Note that breakpoints corresponding to cases 3, 6, and 8 also occur at this instant.

The code detects these conditions by checking to see if the next convex hull vertex (for the case 1 breakpoint) is already in CE. If so, this indicates that P_2 is moving completely inside $CH(P_1)$. An important point is that the joining vertices are not changed (a flag is simply set to indicate they are not 'active'), because when P_2 'emerges' again, the joining edges will be the same as when P_2 'entered' $CH(P_1)$.

Detecting when P_2 emerges from $CH(P_1)$ is relatively easy. Since P_2 is inside an edge of $CH(P_1)$, all that has to be done is to determine when a vertex of P_2 contacts this edge. Note that the last vertex (or two vertices, in the case of Fig. 10) to enter $CH(P_1)$ is the first to emerge, since orientations are fixed. The root calculated is the same type of root calculated for the normal breakpoints. Once the root has the correct sign (i.e., positive, since by convention P_2 always moves in the positive sense of $\boldsymbol{d}$) and is less than or equal to the distance to the next envelope vertex (t_{nfp}), the joining edges which were present upon P_2's entry into $CH(P_1)$ are reinstated, and the algorithm continues normally. Until P_2 emerges, the envelope vertices are simply processed in turn, and P_2 is slid from envelope vertex to envelope vertex.

A limitation of the algorithm should be elaborated upon. The algorithm works for relatively simply connected figures, as stated previously. Further, the extreme case of this would be if P_2 could barely fit inside $CH(P_1)$ through some channel (i.e., the channel width is exactly the same as the width of P_2). In this case the algorithm fails, because the envelope-generation portion of the algorithm (adopted from Mahadevan, 1984) essentially assumes that the envelope itself is a simple polygon. In the special case noted above, the envelope would have two sides coinciding (but in opposite directions). In addition, the envelope-calculation routine uses the concept of sliding edges and sliding vertices. While it is possible to have two sides sliding against one another, it is assumed that there are never two distinct pairs of sliding edges/vertices, as would be true for the limiting case.

The other special case has to do with a zero-length joining edge. This is different from lost joining edges, in that a zero-length joining edge is an instantaneous occurrence. It involves an additional change in the CE vertex list and the joining edges than normal breakpoints require. Consider Fig. 5. Prior to this placement, the P_1-P_2 joining edge was (v_{11}, v_{21}), and the P_2-P_1 joining edge, (v_{22}, v_{13}). As P_2 is slid, the next envelope vertex is encountered (Fig. 5). However, a case 2 breakpoint occurs as well, since the next convex hull vertex of P_1 (with respect to v_1(Join[1]) = v_{11}), or vertex v_{12}, becomes collinear with the joining side (v_{11}, v_{21}). Therefore, v_{12} is inserted into CE, and the P_1-P_2 joining edge becomes (v_{12}, v_{21}). Unfortunately, this joining edge has zero length, and it is not the correct joining edge as P_2 slides beyond the bottom placement. The action required depends on which joining edge has length zero. If the P_1-P_2 joining edge has zero length (this case), then v_2(Join[1]) should be deleted from CE (as v_{21} will be moving inside the convex enclosure), and the next vertex of $CH(P_2)$ (in this case, v_{22}) should be the new joining vertex (resulting in the correct joining side, (v_{12}, v_{22})).

A similar adjustment is made if the P_2-P_1 joining edge has zero length. Essentially what is happening in this special case is that, with a zero-length joining edge, the next convex hull vertex of P_2 or P_1 is meeting the conditions for a breakpoint immediately. A normal iteration of the algorithm requires P_2 to move some positive distance, so a root of zero is not considered.

6. Example

Two iterations of an example are presented. Consider the polygons shown in Fig. 2. The reference point of each figure is defined as the point having the minimal y-coordinate, with ties broken by choosing the smaller x-coordinate. The polygons are translated so that the reference points have coordinates (0, 0). To start the envelope-generation code, the vertex of P_2 with the maximal y-coordinate is made to coincide with the reference point of P_1. The envelope is generated and is shown in Fig. 3 with vertex set $E = \{e_1, \dots, e_9\}$. The vertices of E are passed to the minimal-area algorithm. This code processes the envelope vertices in turn, looking for breakpoints between each pair. The initial placement is shown in Fig. 4, where the coordinates $e_1 = (-1.5, -10.5)$ denote the translation of the reference point of P_2. From Fig. 3, $e_1 = (-1.5, -10.5)$ is the first envelope vertex, and $e_2 = (-1.5, -3.0)$ is the next. Therefore, $t_{\text{nfp}} =$

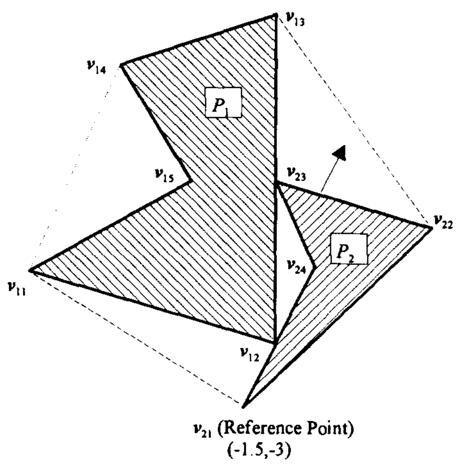


Fig. 11. Second placement for example problem.

7.5, and $\boldsymbol{d} = (0, 1)$. Referring to Fig. 4, the initial convex enclosure is calculated:

$$CE = \{v_{21}, v_{22}, v_{13}, v_{14}, v_{11}\}.$$

The joining sides are P_1-P_2: (v_{11}, v_{21}), and P_2-P_1: (v_{22}, v_{13}). The convex enclosure area function is found for the current CE and $\boldsymbol{d}$, in terms of the sliding distance t. Using the relationships developed in the proof of Theorem 1 and expression (6), the area function is

$$\text{Area}(t) = 273.51 - 10.05t.$$

The joining edges are used to find roots of the breakpoint equations for each case in Table 1. For this slide, no breakpoint occurs prior to reaching the next envelope vertex, so the vertices of CE remain the same. The reference point of P_2 is moved to the coordinates $e_2 = (-1.5, -3.0)$, shown in Fig. 11.

From the new position, $\boldsymbol{d}$ and t_{nfp} are recalculated. The next envelope vertex is accessed; its coordinates are $e_3 = (0, 0)$ (see Fig. 3). Therefore, $t_{\text{nfp}} = 3.354$ and $\boldsymbol{d} = (0.447, 0.894)$. The area function is calculated and is given by

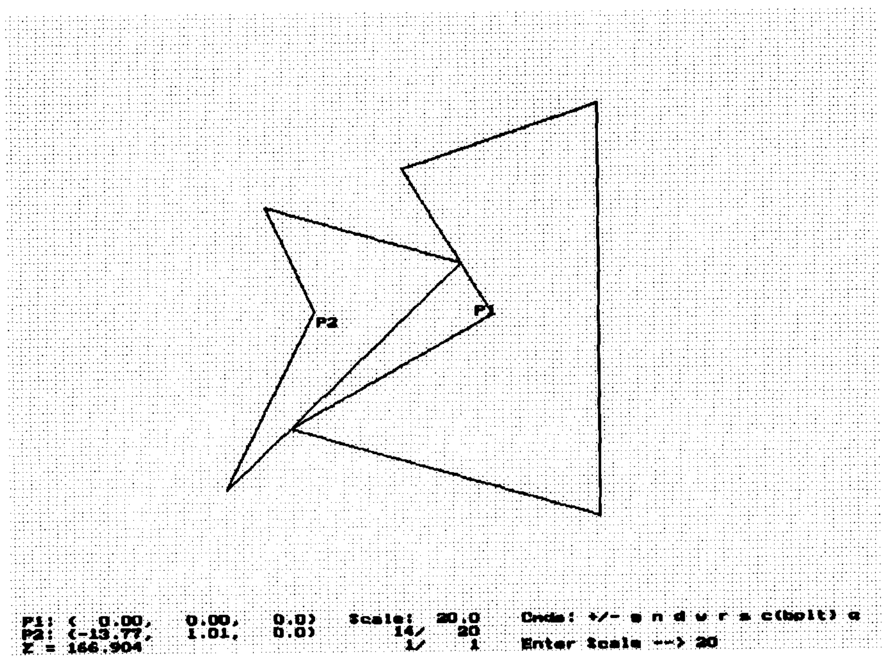
$$\text{Area}(t) = 198.14 - 4.43t.$$

Note that the previous area function evaluated at its limit of t (7.5), is equal to

$$273.51 - 10.05(7.5) = 198.14,$$

or the beginning value of the area on this interval. Calculating the breakpoint roots for each possible case results in a case 1 breakpoint when $t = 3.354$, or simultaneous with reaching the next envelope vertex. This does not cause difficulties. In accordance with case 1, v_{12} enters CE as the P_1-P_2 joining vertex. However, now the P_1-P_2 joining edge has zero length (see the special case discussion in the previous section), so v_{21} is deleted from CE, and v_{22} is set as the joining vertex; that is, the P_1-P_2 joining edge is now (v_{12}, v_{22}). Since the envelope vertex has been reached, the new placement of P_2 is (0, 0), shown in Fig. 5. If there had been a breakpoint root value less than 3.354, then the placement of P_2 would have been calculated based on the placement at $(-1.5, -3.0)$, $\boldsymbol{d} = (0.447, 0.894)$, and the value of t corresponding to the new breakpoint. The value of t_{nfp} would have been reduced by the root value to indicate the remaining distance to the next envelope vertex. A change in $\boldsymbol{d}$ would not be required until the next envelope vertex was reached.

The algorithm continues in this fashion until the starting envelope vertex is reached. The problem was run on an 80386-25MHz machine under MS-DOS. Total execution time (measured with calls to the system clock within the Turbo Pascal program) was 0.72 second, of which 0.29 second was input/initialization, 0.38 second was for generation of the envelope, and 0.05 second was for computing the convex enclosure breakpoints and the area functions, and finding the placement with minimal area. In this example, there are 10 envelope vertices (actually 9, since the starting one is counted twice for 'closure'), and 20 enclosure breakpoints (10 of these being the envelope vertices). A summary of these positions of P_2 and the area functions on the intervals is given Table 2. The 'Envelope Vertex' column indicates whether the placement corresponds to an envelope vertex, or to a breakpoint occurring between envelope vertices. The lower limit on t for each area function is 0, and the upper limit is given in the 'tmax' column. By inspection, the best placement is the 14th listed, at $(-13.769, 1.013)$ with an area of 166.9 square units. This configuration is shown in Fig. 12. The area as a function of total sliding distance is shown in Fig. 13. Points corresponding to envelope vertices are labeled



```
P1: (  0.00,    0.00,    0.0)   Scale:  20.0    Cmds: +/- e n d v r s c(bolt) q
P2: (-13.77,    1.01,    0.0)          14/   20
Z = 166.904                             1/    1    Enter Scale --> 20
```

Fig. 12. Optimal placement for example problem.

Table 2
Summary of results for example

#	Position of P_2	Envelope vertex	Area function	tmax
1	(−1.500, −10.500)	Y	273.510 − 10.050t	7.50000
2	(−1.500, −3.000)	Y	198.135 − 4.427t	3.35410
3	(0.000, 0.000)	Y	183.285 + 0.000t	4.36250
4	(0.000, 4.363)	N	183.285 + 3.600t	0.93750
5	(0.000, 5.300)	N	186.660 + 7.200t	10.00000
6	(0.000, 15.300)	Y	258.660 − 9.392t	3.77159
7	(−3.578, 14.107)	N	223.237 − 5.502t	3.81787
8	(−7.200, 12.900)	Y	202.230 − 8.402t	2.08507
9	(−8.700, 11.452)	N	184.711 − 2.899t	2.47587
10	(−10.481, 9.732)	N	177.534 + 0.357t	2.55689
11	(−12.321, 7.956)	N	178.448 + 3.613t	4.86189
12	(−15.818, 4.579)	N	196.014 + 8.759t	0.11368
13	(−15.900, 4.500)	Y	197.010 − 7.367t	4.08684
14	(−13.769, 1.013)	Y	166.904 + 8.759t	0.66494
15	(−14.247, 0.551)	N	172.728 + 10.397t	8.13555
16	(−20.100, −5.100)	Y	257.310 − 10.434t	7.50000
17	(−12.900, −7.200)	Y	179.055 − 0.000t	4.35933
18	(−8.713, −8.412)	N	179.055 + 5.244t	0.01310
19	(−8.700, −8.416)	N	179.124 + 12.592t	7.49559
20	(−1.500, −10.500)	Y	(same as position 1)	

‘env’. In this case, the global minimum occurs at an envelope vertex; however, there is no guarantee of this in general.

7. Computational experience

Brief computational experience is presented in this section. The problems developed here give a good indication of how the algorithm performs as problem size increases. Some of the problems were specifically designed to test special cases. Computational complexity is addressed first.

Envelope generation of P_2 with respect to P_1 is the most computationally-intensive aspect. Once the envelope has been found, finding the convex enclosure breakpoints and computing the area is very fast. The computational complexity of the envelope-generation portion is

$$O((p1 + p2) \cdot p1 \cdot p2)$$

(Mahadevan, 1984). The $(p1 + p2)$ factor reflects the observation that the number of envelope vertices is of this order. The $p1 \cdot p2$ factor represents the time required to find each envelope vertex. While implementing the envelope algorithm for this research, improvements were made in finding each envelope vertex. These improvements had to do with limiting the number of possible intersecting edges/vertices during a slide operation (using convex hull information from P_1 and P_2). Basically, in searching for the next envelope vertex, the improved algorithm calculates the distance until the next contact for all vertices up to the next convex hull vertex. P_2 is ‘tentatively’ moved the shortest of these distances, and an overlap check is performed. If the polygons are not overlapping, the search for the next vertex of E begins. Otherwise, some other vertex must cause the next contact, so the distance to the next contact is found for all vertices, as in Mahadevan’s original code. While approximately halving running times, the worst-case complexity does not change.

Given the envelope of P_2 with respect to P_1 with $O(p1 + p2)$ vertices, the complexity of the convex enclosure portion of the algorithm depends only on the number of breakpoints (types 1, 2, and 3) encountered, since detection and processing of an individual breakpoint is done in constant time. Clearly the number of type 1 breakpoints (envelope vertices) is $O(p1 + p2)$. Turning to the type 2 and 3 breakpoints, each convex hull vertex of P_1 (P_2) becomes a joining vertex at most once as P_2 orbits around P_1. When a particular vertex enters the CE, it enters as a joining vertex. Similarly, when it leaves the

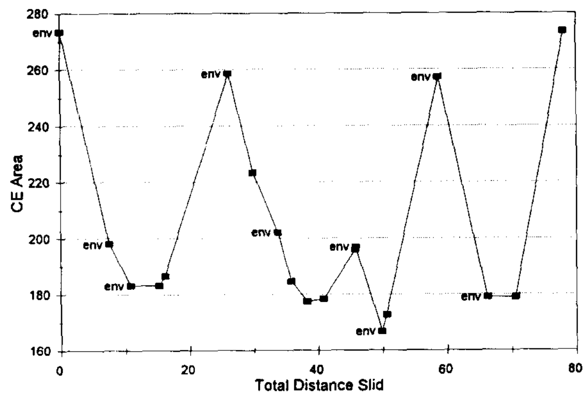


Fig. 13. Area of convex enclosure as a function of total sliding distance.

CE, it leaves as a joining vertex. This leads to an upper bound on the number of type 2 and 3 breakpoints of $O(c1+c2)$, where $c1$ is the number of convex hull vertices of P_1, and similarly for $c2$. Hence, the complexity of the convex enclosure portion of the algorithm is

$$O((p1+p2)+(c1+c2)).$$

Assuming that $c1=O(p1)$ and $c2=O(p2)$, this upper bound reduces to $O(p1+p2)$. The complexity of the envelope-generation portion is of higher order than the convex enclosure portion. This dominance is reflected in the numerical results.

Results from eight problems are presented here. To provide a measure of performance for the algorithm, a ratio was used to indicate the degree of 'convexity' of a polygon. The P_1 area ratio is calculated as

$$P_1 \text{ ratio} = \frac{\text{Area}(CH(P_1))}{\text{Area}(P_1)}$$

and the P_2 ratio is analogous. The minimum value is one, implying that P_1 is perfectly convex. The amount by which the ratio exceeds one is the percentage of area added to form the convex hull of P_1 with respect to the area of P_1. Similarly, the convex enclosure ratio is calculated as

$$CE \text{ ratio} = \frac{\text{Area}(CE)}{\text{Area}(P_1)+\text{Area}(P_2)}.$$

The *CE* ratio measures the convexity of the convex enclosure. The meaning of this ratio is similar to the P_1 and P_2 ratios, but requires an additional consideration. By comparing the *CE* ratio with the P_1 and P_2 ratios, one can determine if the convexity of the composite figure is better or worse than the individual convexities. For example, if P_1 and P_2 are highly nonconvex, but fit together and form a perfectly convex enclosure, then the individual convexity ratios would be greater than 1, but the *CE* ratio would equal 1. A ratio such as this may provide a useful way to judge the acceptability of a module for a general pattern layout problem.

Information about the problems, the number of envelope and enclosure vertices encountered, and ratios of certain areas, is presented in Table 3. All calculations were done on an 80386-25MHz computer under MS-DOS. A math coprocessor was used, as was a disk cache. Problem 1 is the same as the example used in previous sections. Problems 2–5 use convex polygons. Problems 6–8 use non-convex polygons, in general. Looking at the ratios for the convex problems, the *CE* ratio could never be better than the individual ratios. For problem 1 and problems 6–8, the *CE* ratio beats both of the individual ratios in one case (problem 7) and improves on the convexity of one of the figures in the other cases.

Execution times are reported for separate components of the algorithm, as well as total time in Table 4. The 'SE' column refers to finding the simple enclosure of the minimal-area placement.

Table 3
Computational results: Area ratios

#	Size $p1/p2$	# Vertices [a] Env/Enc	Area ratios P_1	P_2	CE
1	5/4	9/19	1.304	1.458	1.432
2	4/3	7/11	1.0	1.0	1.0
3	3/3	6/6	1.0	1.0	1.0
4	6/6	13/37	1.0	1.0	1.125
5	8/8	17/48	1.0	1.0	1.072
6	8/4	20/20	1.333	1.0	1.280
7	14/15	28/51	1.631	1.629	1.620
8	23/9	30/52	1.894	1.142	1.526

[a] Env: Number of vertices in envelope.
Enc: Number of breakpoints in convex enclosure algorithm.

Table 4
Computational results: Execution times

#	Execution times (seconds) I/O [a]	Envelope [b]	Enclosure [c]	SE [d]	Total
1	0.29	0.38	0.05	0.00	0.72
2	0.21	0.17	0.06	0.00	0.44
3	0.11	0.16	0.06	0.00	0.33
4	0.49	0.77	0.17	0.05	1.48
5	0.60	1.54	0.22	0.06	2.47
6	0.33	1.05	0.11	0.00	1.54
7	0.88	8.08	0.21	0.11	9.28
8	0.93	8.73	0.22	0.06	9.94

[a] Input/Output, Initialization.
[b] Finding envelope of $P2$ with respect to $P1$.
[c] Finding minimum-area convex enclosure.
[d] Finding coordinates of simple polygon enclosing minimal-area placement.

Essentially, this is the amount of time required to combine the two pieces into one, as might be done in a larger pattern layout setting. As problem size grows, a greater percentage of the total time is spent finding the envelope. This is consistent with the earlier complexity discussions. For example, in problem 1, 53% of the time is spend in the envelope routine. In problem 8, which is larger, the corresponding percentage is 88%. Even for the larger problems, the times seem fairly reasonable, considering that faster processors are commonplace.

8. Conclusions

An important nesting problem has been solved in this paper. Given two simple polygons, the algorithm finds the configuration which minimizes the area of their convex enclosure. The problem has obvious application in nesting irregular shapes. The algorithm relies on an existing envelope-generation code, but the logic for dynamically maintaining the vertices of the convex enclosure and the area function are new contributions. The algorithm has been implemented and execution times are quite reasonable.

Two features of this algorithm are noteworthy. First, a complete analytic representation of the convex enclosure area as a function of sliding distance is an output of the algorithm. This function is piecewise-linear and continuous, but not convex. Second, the algorithm produces a complete description of the dynamic set CE as P_2 orbits around P_1. The objective in this research was to minimize area, but any measure which uses CE vertices could be analyzed, after determining the properties of the objective between breakpoints.

There are several interesting avenues worth pursuing. Obviously the integration of this algorithm with other 'building-block' methods into a heuristic technique for the irregular pattern layout problem would be interesting. Specifically, integration with placement routines and computational geometry algorithms (e.g., polygon containment) seems promising. Meta-strategies such as tabu search and/or simulated annealing may prove useful. In the past, many placement routines have been sequential with no possibility of backtracking. The current algorithm seems well-suited for integration with a meta-heuristic such as tabu search, which tries to escape from local optima encountered when placing pieces. Assuming that composite pieces are formed out of two or more original pieces, the backtracking and/or tabu attributes could revolve around placing the composite pieces in a different order, or in reforming the composites themselves.

Other possible extensions of the algorithm include analyzing different objectives, the incorporation of constraints on the configuration of P_1 and P_2, different orientations of the pieces, and the generation of composites with 'desirable' (from a pattern layout perspective) shapes.

Acknowledgements

The first author acknowledges support received from the Office of Naval Research in the form of a National Defense Science and Engineering Graduate Fellowship. Both authors gratefully acknowledge the helpful comments from two anonymous referees and the editors, which improved an earlier version of this paper.

References

Adamowicz, M. (1969), "The optimum two-dimensional allocation of irregular, multiply-connected shapes with linear, logical, and geometric constraints", Ph.D. Dissertation, Electrical Engineering Department, New York University.

Adamowicz, M., and Albano, A. (1976), "Nesting two dimensional shapes in rectangular modules", *Computer Aided Design* 8, 27–33.

Amaral, C., Bernardo, J., and Jorge, J. (1990), "Marker making using automatic placement of irregular shapes for the garment industry", *Computers & Graphics* 14/1, 41–46.

Art, R.C. (1966), "An approach to the two-dimensional irregular Cutting Stock problem", Report No. 320-2006, IBM Cambridge Scientific Center, Cambridge, MA.

Chung, J., Scott, D., and Hillman, D.J. (1990), "An intelligent nesting system on 2-D highly irregular resources", *Applications of Artificial Intelligence VIII (SPIE)* 1293/1, 472–483.

CRC (1975), *Standard Mathematical Tables*, 23rd ed., CRC Press, Cleveland, OH.

Dagli, C.H. (1990), "Neural networks in manufacturing: Possible impacts on Cutting Stock Problems", in: *Proceedings of the Rensselaer 2nd International Conference on CIM*, May 21–23, 1990, 531–537.

Dori, D., and Ben-Bassat, M. (1983), "Circumscribing a convex polygon by a polygon of fewer sides with minimal area additions", *Computer Vision, Graphics, and Image Processing* 24, 131–159.

Freeman, H., and Shapira, R. (1975), "Determining the minimum-area encasing rectangle for an arbitrary closed curve", *Communications of the ACM* 18/7, 409–413.

Grinde, R.B. (1993), "Polygon containment and nesting: An optimization-based approach", Ph.D. Thesis, Department of Industrial and Management Systems Engineering, Pennsylvania State University.

Guibas, L., Ramshaw, L., and Stolfi, J. (1983), "A kinetic framework for computational geometry", in *24th Annual Symposium on Foundations of Computer Science*, November 7–9, 1983, 100–111.

Lee, H.C., and Woo, T.C. (1988), "Determining in linear time the minimum area convex hull of two polygons", *IIE Transactions* 20/4, 338–345.

Mahadevan, A. (1984), "Optimization in computer aided pattern packing", Ph.D. Thesis, North Carolina State University.

Oliveira, J.F.C., and Ferreira, J.A.S. (1993), "Algorithms for nesting problems", in: R.V. Vidal (ed.), *Applied Simulated Annealing*, Lecture Notes in Economics and Mathematical Systems 396, Springer-Verlag, Berlin, 256–273.

Poshyanonda, P., and Dagli, C.H. (1992), "A hybrid approach to composite stock cutting: Neural networks and genetic algorithms", in M. Jamshidi, R. Lumia, J. Mullins and M. Shahinpoor (eds.), *Robotics and Manufacturing: Recent Trends in Research, Education, and Applications*, 4.

Sedgewick, R. (1990), *Algorithms in C*, Addison-Wesley, Reading, MA.