INFORMATION AND COMPUTATION 83, 98–110 (1989)

Finding Minimal Convex Nested Polygons*

ALOK AGGARWAL

IBM Research Division, T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598

HEATHER BOOTH,[†] JOSEPH O'ROURKE,[§] AND SUBHASH SURI[‡]

Department of Electrical Engineering and Computer Science, The Johns Hopkins University, Baltimore, Maryland 21218

AND

CHEE K. YAP

Courant Institute of Mathematical Sciences, 251 Mercer Street, New York, New York 10012

We consider the problem of finding a polygon nested between two given convex polygons that has a minimal number of vertices. Our main result is an $O(n \log k)$ algorithm for solving the problem, where n is the total number of vertices of the given polygons, and k is the number of vertices of a minimal nested polygon. We also present an $O(n)$ sub-optimal algorithm, and a simple $O(nk)$ optimal algorithm.

1. INTRODUCTION

We provide an efficient algorithm for the following problem: given two convex polygons P and Q such that Q is contained in P, determine a minimum vertex polygon K that contains Q and is contained in P. A polygon K is called *nested* between P and Q when it circumscribes Q and is inscribed in P. The problem was originally posed by Victor Klee for

* A preliminary version of this paper appeared in the first Annual Symposium on Computational Geometry, 1985, pp. 296–303, Baltimore, MD 21218.

[†] Author's present address: Dept. of Computer Science, Princeton University, Princeton, NJ 08544.

[‡] Author's present address: Bell Communications Research, 435 South St., Morristown, NJ 07960.

[§] Author's present address: Department of Computer Science, Smith College, Northampton, MA 01063.

0890-5401/89 $3.00

polytopes in arbitrary dimensions; to the authors' knowledge this remains an open problem. For the 2-dimensional case, we present an algorithm that runs in $O(n \log k)$ time, where n is the total number of vertices of P and Q and k is the number of vertices of K. The model of computation used is the usual random access machine (RAM) that allows simple arithmetic operations (like +, −, /, *) to be performed in unit time (see Aho *et al.*, 1974 for details).

The problem of determining a minimum vertex nested polygon belongs to the general area of polygonal approximations. This area has been extensively studied recently due to its numerous applications in robotics, stock cutting, collision avoidance and computer aided design problems (see References). Unlike previous investigations, our goal is to minimize the combinatorial complexity of the approximation, rather than a continuous measure as an area.

Our paper is organized in five sections. Section 2 introduces some notation and basic lemmas. Section 3 provides a straightforward linear algorithm that yields a nested polygon with at most one vertex more than the minimum. Using this algorithm, a simple $O(nk)$ algorithm is developed. This algorithm is subsequently improved to run in $O(n \log k)$ time, which is the main result of the paper. Finally, Section 5 concludes with some remarks and directions for further research.

2. Basic Lemmas

In this section we present basic lemmas that characterize minimum vertex polygons and help us discretize the search routine.

A nested polygon is called *minimal* if it has the minimum number of vertices among all polygons nested between P and Q. Henceforth, we will use P and Q to represent the boundaries of the respective polygons. The vertices of the polygons are assumed to be indexed counterclockwise and unless stated, our traversal of various polygons will also be in counterclockwise order. Let A be the closed region (the annulus) of the plane bounded by P and Q.

Lemma 1. *Every minimal polygon is convex.*

Proof. Assume to the contrary that there exists a minimal polygon $K = (v_1, ..., v_k)$ that is not convex. Then, consider the convex hull of K. Convexity of P and Q assures that Q is entirely contained in the convex hull of K which, in turn, is entirely contained in P. But, the convex hull of K has less vertices than K which contradicts the minimality of K. ∎

A *supporting line segment* is a directed segment in A that supports Q on its left and has both its endpoints on P. For any point a on P, let l_a be the unique supporting segment ab, directed from a to b. Define R_a to be the region $H_{l_a} \cap A$, where H_{l_a} is the closed right half-plane determined by the line l_a. Finally, let R'_a equal $R_a - \{a\}$ (see Fig. 1).

LEMMA 2. *For any $a \in P$, R'_a contains at least one vertex of any minimal polygon.*

Proof. If R'_a does not contain any vertex of K, then there must exist two adjacent vertices of K, x and y, that are joined through R'_a. As both x and y are to the left of l_a, it is impossible for the edge xy to intersect R'_a. ▮

Define a *supporting polygon* $(v_1, v_2, ..., v_k)$ as one, all of whose edges, except for perhaps the last one $v_k v_1$, are supporting segments. Let S_a be the supporting polygon with $v_1 = a$. For an edge $v_i v_{i+1}$ of the supporting polygon $S_a = (v_1, v_1, ..., v_k)$, define the *Q-contact* as the vertex of Q that supports $v_i v_{i+1}$ and the *P-contact* as the edge of P on which v_i lies. In case of degeneracies, we choose the vertex of Q and the edge of P with higher indices. Clearly, a supporting polygon of k edges has at least $k-1$ Q-contacts and k P-contacts. Also, S_a can be completely and uniquely specified by $a \in P$, the *start* vertex, and by all its contacts with P and Q.

LEMMA 3. *For any $a \in P$, S_a has at most one vertex more than the minimum.*

Proof. $S_a = (v_1, ..., v_k)$ has, by definition, at least $(k-1)$ supporting segments, namely $l_{v_1}, ..., l_{v_{k-1}}$. By Lemma 2, each R'_{v_i}, defined by l_{v_i}, where $0 \leqslant i \leqslant k-1$, must contain a vertex of the minimal polygon. Hence every minimal polygon must have at least $(k-1)$ vertices. ▮

In the following lemma, we establish that the search domain for the minimal polygon can be restricted to one R_a.

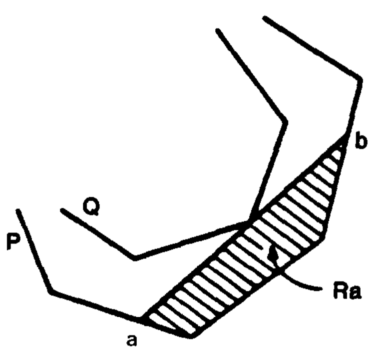

FIG. 1. A supporting segment *ab*.

LEMMA 4. *Any minimal polygon* $K = (v_1, ..., v_k)$ *can be transformed into an* S_a *for some* $a \in P$.

Proof. The transformation is accomplished in two steps:

(1) Force all vertices of K to lie on P. Call this intermediate polygon K'.

(2) Identify one vertex of K' as a and rotate all the sides of K' until all, except perhaps the last one, become supporting segments; denote the resulting polygon by K''.

Step 1. Let v_1, v_2, and v_3 be three consecutive vertices of K such that v_2 does not lie on P. See Fig. 2a. Let the directed segment $v_1 v_2$ when extended, intersect P at v_2'. Replace v_2 by v_2'. Convexity of P and Q assures that $v_2' v_3$ lies completely in A. Minimality of K assures that neither a collinearity nor a non-convexity can arise from this alteration. This procedure can be iteratively applied to move all the vertices of K onto P. The number of vertices of the resulting polygon K' is clearly the same as that of K.

Step 2. Without loss of generality, set $v_1' = v_1'' = a$. See Fig. 2b. Rotate $v_1'' v_2'$ about v_1'' until it becomes coincident with l_a. Replace v_2' with the other endpoint of l_a, say v_2''. Repeat the procedure for $v_2'' v_3'$ and so on up to $v_{k-1}'' v_k''$. Again, the minimality of K' ensures that no collinearities or non-convexities can occur as a result of executing this procedure.

After the completion of Step 2, K'' is supporting, thereby, establishing the lemma. ∎

Lemmas 2, 3, and 4 are sufficient to establish that a minimal polygon can be found by examining all S_x's, where x is a point on P in R_a, for any $a \in P$.

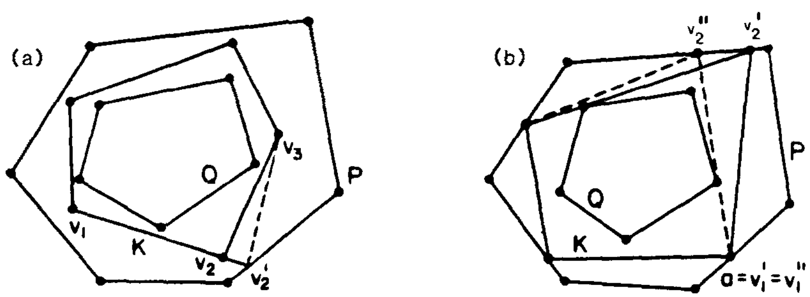

FIG. 2. (a) Transformation to a supporting polygon (step 1). (b) Transformation to a supporting polygon (step 2).

3. Algorithms for Finding S_a and Minimal Nested Polygons

In this section, we develop an algorithm to find S_a, given a point a on P, and use it to find a minimal polygon.

3.1. *Algorithm for Constructing S_a*

S_a is constructed by first identifying v_1 with a and then "wrapping" around Q with supporting segments $l_{v_1}, l_{v_2}, ..., l_{v_{k-1}}$. The intersection of l_{v_i} with P is taken to be v_{i+1}, for $1 \leqslant i \leqslant k-1$. Only the last edge may not be a supporting segment, in which case it is forced to be $v_k v_1$.

The segment l_{v_1} can be found in $O(n)$ steps by a linear search around Q for the Q-contact q_1 and around P for v_2. Subsequently, each new edge, l_{v_i}, can be found in time linear in the number of edges of P on the boundary of R_{v_i} and vertices of Q between the Q-contacts of $l_{v_{i-1}}$ and l_{v_i}, q_{i-1}, and q_i. Therefore, the time complexity of this algorithm is $O(n)$. This establishes our claim that a sub-optimal nested polygon, with at most one vertex more than the minimum, can be constructed in linear time.

In the following we describe how to obtain a minimal nested polygon by modifying an arbitrary supporting polygon. Let a point $z \in P$ be called a *contact change* point if S_z has at least one contact different from S_y, where $y \in P$ is a point in the neighborhood of z that is immediately preceding it in a clockwise traversal.

Intuitively, a minimal polygon is obtained by starting with $S_x = S_a$ and rotating S_x around in A by "sliding" x along P and checking if the non-supporting segment $v_k v_1$ ever "collapses" to a point. This search is discretized by computing S_x only for those points x along P that ar contact change points. Furthermore, starting with an arbitrary supporting polygon, S_a, we need to search through only those contact change points that lie on the outer boundary of R_a. It may be the case, however, that a minimal polygon S_x is achieved at some x that lies *between* contact change points. To locate such minimal polygons, we use a set of *projection functions* that relate the position of any vertex v_i, in particular v_k, to that of v_1. These functions, as described below, are simple polynomial quotients and have their coefficients dependent only on the current P- and Q-contacts. As the contacts of S_x do not change while x is moved between two adjacent contact change points, these functions do not change either. Therefore, two adjacent contact change points also determine the range for the applicability of these functions, within which they can be used to detect a coincidence of v_k with v_1.

3.2. *Projection Functions*

Let $S_{v_1} = (v_1, v_2, ..., v_k)$ by a supporting polygon. Let the Q-contact of $v_1 v_2$ be $q = (q_x, q_y)$. Let $\mathbf{i}$ and $\mathbf{e}$ be vectors parallel to the edges of P

containing v_1 and v_2, respectively. Assume a coordinate system with its X-axis alligned with $\mathbf{i}$. Using x_1 and x_2 as parametric representations, the positions of $\mathbf{v}_1$ and $\mathbf{v}_2$ can be represented as

$$\mathbf{v}_1 = x_1 \mathbf{i}$$

$$\mathbf{v}_2 = \mathbf{d} + x_2 \mathbf{e},$$

where $\mathbf{d}$ is some constant vector and d denotes its magnitude (see Fig. 3).

Let r and s be the points on X-axis that coincide with the perpendicular projections of q and v_2, respectively. From the similarity of the triangles $\Delta v_1 q r$ and $\Delta v_1 v_2 s$, we obtain

$$q_y/(q_x - x_1) = x_2 \sin \vartheta/(d - x_1 + x_2 \cos \vartheta).$$

Solving for x_2 gives

$$x_2 = q_y(d - x_1)/\sin \vartheta(q_x - q_y \cot \vartheta - x_1)$$

which is equivalent to

$$x_2 = (c_1 + c_2 x_1)/(c_3 + c_4 x_1) \tag{1}$$

for constants c_1, c_2, c_3, and c_4 that depend only on the contacts of $v_1 v_2$. In the same way, the position of v_3 is functionally related to that of v_2, i.e.,

$$x_3 = (d_1 + d_2 x_2)/(d_3 + d_4 x_2) \tag{2}$$

for constants d_1, d_2, d_3, and d_4 that depend only on the contacts of $v_2 v_3$. Substitution of x_2 from (1) into (2) gives, after simplification, a composed relationship between x_3 and x_1,

$$x_3 = (e_1 + e_2 x_1)/(e_3 + e_4 x_1), \tag{3}$$

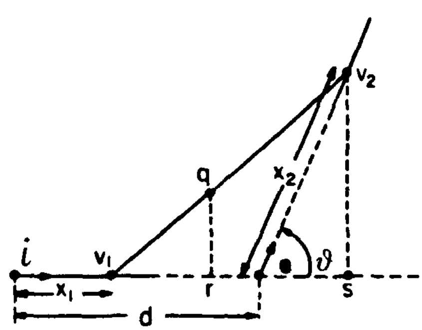

FIG. 3. Computing projection functions.

where e_1, e_2, e_3, and e_4 are constants that depend only of the contacts between v_1 and v_3.

In the same way $x_4, x_5, ..., x_k$ can also be expressed as functions of x_1. We will write these functions as $x_i = f_i(x_1)$. As the degree of f_i is independent of i, these functions can be evaluated in contant time.

We now describe how to compute the next contact change point for a given supporting polygon.

3.3. *Computation of the Next Contact Change Point*

Given a supporting polygon $S_{v_1} = (v_1, v_2, ..., v_k)$, we need to determine the edge of S_{v_1} that first changes its contact if v_1 is "slid" along P while the polygon is constrained to remain supporting. The corresponding position of v_1 is called the *next contact change* point of S_{v_1}. We first compute the edge that undergoes this contact change and then use the projection functions to compute the next contact change point.

For an edge $v_i v_{i+1}$ of S_{v_1}, let q_i be its Q-contact. Rotate $v_i v_{i+1}$ (extended to P on both sides) counterclockwise about q_i until either the edge $v_i v_{i+1}$ becomes flush with $q_i q_{i+1}$ or either v_i or v_{i+1} coincides with a vertex of P. Let the corresponding position of v_i be denoted by v_i^c (see Fig. 4).

If v_i is moved from its current position to v_i^c while the polygonal chain of vertices between v_1 and v_{i+1} is constrained to remain supporting, at least one edge changes its contacts. Let $v_i^{\diamond}$ be the first position between v_i and v_i^c such that moving v_i to $v_i^{\diamond}$ forces some edge among $v_1 v_2$, $v_2 v_3$, ..., $v_i v_{i+1}$

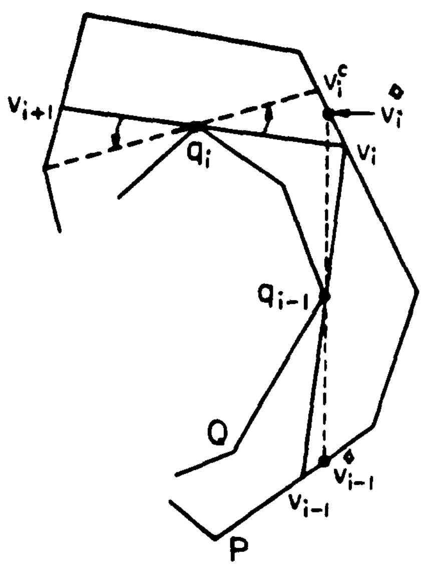

FIG. 4. Computing the next contact change point.

to change contact. Our goal is to compute $v_{k-1}^{\diamond}$ and the edge that changes its contact first. Assume inductively that we know $v_{i-1}^{\diamond}$. Then $v_i^{\diamond}$ is the more clockwise of the two points: point v_i^c and the intersection point of the extension of $v_{i-1}^{\diamond} q_{i-1}$ with P (Fig. 4). This calculation is performed by the for loop in the following pseudo-code:

3.4. *Algorithm for Computing a Minimal Polygon*

1. Choose a point $a \in P$. (Let a be a vertex of P.)
2. Compute the initial polygon S_a and set $v_1 = a$.
3. Compute $\{f_i\}$, the set of projection functions.
4. **while** $v_1 \in R_a$ **do begin**
   {Compute next contact to change.}
5.   Compute v_1^c; $i^{\diamond} \leftarrow 2$; $v_2^{\diamond} \leftarrow$ intersection of $v_1^c q_1$ extended to P;
6.   **for** $i = 2$ **to** $k-1$ **do begin**
7.   Compute v_i^c;
8.   **if** v_i^c **is clockwise of** $v_i^{\diamond}$ **then begin**
9. $v_i^{\diamond} \leftarrow v_i^c$;
10. $i^{\diamond} \leftarrow i$
        end {**if**}
11. $v_{i+1}^{\diamond} \leftarrow$ intersection of $v_i^{\diamond} q_i$ extended to P;
      end {**for**}
12. Compute *next contact change* point from $i^{\diamond} f_{i^{\diamond}}^{-1}$;
13. Move v_1 to *next contact change* point;
14. Check for the overlap of v_k with v_1 between the contact changes by solving $x_k = f_k(x_1)$;
15. Recompute $f_{i}^{\diamond}, f_{i+1}^{\diamond}, \ldots, f_k^{\diamond}$
    end {**while**}

3.5. *Complexity of the Algorithm*

Steps 1 and 2 require time $O(n)$. Step 3 takes $O(nk)$ time. The **while** loop of step 4 is executed as many times as the number of contact changes that can occur during the movement of v_1 throughout R_a. That the total number of contacts that can change during the movement of v_1 throughout R_a is $O(n)$ can be established as follows. Let the initial supporting polygon S_a define a set of disjoint regions $R_1, R_2, \ldots, R_{k-1}$, one for each supporting edge of S_a, as defined before. As v_1 is moved through R_1, the ith edge of the corresponding supporting polygon will have the following as contacts throughout this movement:

(1) the edges of P that lie in R_i, and

(2) the vertices of Q that lie between the Q-contacts of the segments defining R_i and R_{i+1}.

Hence, all the edges of S_{v_1} together can have $O(n)$ contact changes as v_1 moves throughout R_a. Each execution of the **while** loop takes $O(k)$ time to compute the next contact change, and $O(k)$ time to update functions affected by this contact change. Checking for overlap between v_k and v_1 can be done in $O(1)$ time. Hence the overall time complexity of the algorithm is $O(nk)$.

4. $O(n \log k)$ Algorithm for Computing Minimal Nested Polygons

The $O(nk)$ algorithm for minimal polygons can be improved to $O(n \log k)$ by organizing the edges of the nested polygon and the projection functions hierarchically into a complete binary tree so that the movement between two contact changes can be accomplished in $O(\log k)$ rather than $O(k)$ time. Without loss of generality, we shall assume throughout the following discussion that $k-1=2^d$ for some integer d. In Section 3, we introduced the projections functions f_i for a supporting polygon $(v_1, v_2, ..., v_k)$ which map the position of a vertex v_i, expressed as a distance along v_1's contact edge. Now we need the ability to map between any two vertex coordinate systems. Let $f_{i,j}$ be the function that maps the position of vertex v_i to a position along v_j's contact edge, where $i > j$. These functions are similar in form and properties to the function f_i, and, in fact $f_{i,1} = f_i$.

Our main data structure is a complete binary tree with $k-1$ leaves that are indexed by edges of the nested polygon. The indices are in increasing order from right to left, i.e., leaf i represents the edge $v_i v_{i+1}$ of the nested polygon $(v_1, v_2, ..., v_k)$. For any node s in the tree, let the subtree for which it is the root be denoted by T_s. A node s in the tree stores the following information:

1. $l(s)$ is the index of the lowest indexed leaf (i.e., the rightmost leaf) in T_s;

2. $f_{i,l(s)}$, where $i = 1 + i\max(T_s)$, with $i\max(T_s)$ being the largest index of a leaf of T_s (when $i=k$, the function $f_{k,l(s)}$ is never actually used, and we assume that $f_{k,k-1}$ is the identity);

3. $\alpha(s)$, index of the first contact to change among the leaves of T_s when $v_{l(s)}$ is moved counterclockwise;

4. $\delta(s)$, the distance by which $v_{l(s)}$ must move before $\alpha(s)$ changes its contact;

5. $\Delta(s)$, a correction factor that records the distance by which the leaves of T_s have moved when a lower indexed edge changes contact.

Let this tree be called a *function tree* and its root be denoted by r. Let s_L and s_R, respectively, be the left and right children of a node s. Given a supporting polygon $(v_1, v_2, ..., v_k)$ and the corresponding function tree, the next contact that is going to change can be computed on $O(1)$ time: it is determined by comparing the "winners" from the two subtrees, T_{r_L} and T_{r_R} within the coordinate system of $l(r)=1$, i.e., by computing

$$\min\{f_{i,1}(\delta(r_L)), \delta(r_R)\},$$

where $i=l(r_L)=((k-1)/2)+1$. Note that $l(r_R)=1$ so that both distances are expressed in the same coordinate system.

Hence, our main task is to construct this tree in $O(n \log k)$ time and update it appropriately after each contact change in $O(\log k)$ time.

4.1. *Initial Construction of the Function Tree*

The tree is computed recursively, bottom-up from the leaves. The information associated with a leaf node j is easily computed in constant time as follows:

1. $l(j)=j$;
2. $f_{j+1,j}$ is computed as discussed in Section 3;
3. $\alpha(j)=j$;
4. $\delta(j)=$ distance from v_j to v_j^c (see Section 3 for v_j^c);
5. $\Delta(j)=0$.

The information associated with an internal node s is, then, computed from its two children as follows:

1. $l(s)=l(s_R)$;
2. $f_{i,l(s)}=f_{i,l(s_L)}\cdot f_{l(s_l),l(s_R)}$, where $i=i\max(T_s)$;
3. $\alpha(s)$ is either $\alpha(s_L)$ or $\alpha(s_R)$ depending on which one achieves $\delta(s)$ in (4) below;
4. $\delta(s)=\min(f_{i,j}(\delta(s_L)), \delta(s_R))$, where $i=l(s_L)$ and $j=l(s_R)$;
5. $\Delta(s)=0$.

Since the computation at the leaf level can be performed in $O(k)$ time, the entire tree can be constructed in $O(k \log k)$ time.

4.2. *Updating the Tree after a Contact Change*

We show that only $O(\log k)$ nodes in the tree need to be updated after a contact change has been performed. Let the last contact to change be indexed u. The leaf information associated with u can be recomputed in $O(1)$ time following the initialization step discussed previously. Since the relative ordering of the contacts in a subtree T_s is not affected by the change of a contact that is not a leaf in T_s, correction of the information stored at s would implicitly update all of T_s. The tree is updated by recomputing the associated function and adjusting the correction factor Δ of all the sibling nodes along the path from u to the root r. There can be $O(\log k)$ such nodes. We now show that each node can be updated $O(1)$ time. The correction of $f_{i,l(s)}$ and $\alpha(s)$ can be done by the same computation as in the initial construction of the tree. The remaining updates are done as follows:

$$\delta(s) = \min\{f_{i,j}(\delta(s_L) + \Delta(s_L)), \delta(s_R) + \Delta(s_R)\},$$

where

$$i = l(s_L) \text{ and } j = l(s_R).$$

In order to compute $\Delta(s)$, we need to determine the distance by which $l(s)$ has moved due to the change of contact at u. The cases when $u > l(s)$, $l(s) > u$, and $u = l(s)$ are updated as follows:

$$\Delta(s) = f_{u,l(s)}(\delta(u)) \quad \text{when} \quad u > l(s),$$
$$\Delta(s) = f^{-1}_{l(s),u}(\delta(u)) \quad \text{when} \quad l(s) > u,$$
$$\Delta(s) = \delta(u) \quad \text{when} \quad u = l(s).$$

These functions can be computed by appropriate composition of the functions stored at the nodes on the path from u to r. Functions at the right sibling nodes along this path are composed to compute $f_{i,j}$ if $i < u$, and those at the left sibling nodes if $i > u$. For instance, let $u = 7$ in Fig. 5. The function $f_{7,1}$ that is needed at the node B can be computed by composing $f_{7,5}$ and $f_{5,1}$. This establishes that the function tree can be appropriately updated in $O(\log k)$ time after each contact change. Since there can be $O(n)$ contact changes during the rotation of the supporting polygon, the total time complexity of this algorithm is $O(n \log k)$.

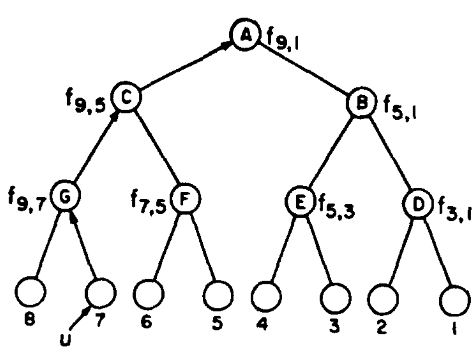

FIG. 5. Function tree (path of updates shown by arrowheads when $u = 7$).

5. Conclusions

We have considered the problem of finding a minimal vertex polygon nested between two convex polygons, and have offered three algorithms, which find:

(1) a polygon with at most one vertex more than the minimum in $O(n)$ time;

(2) a minimal polygon in $O(nk)$ time;

(3) a minimal polygon in $O(n \log k)$ time.

We suspect that the last algorithm is optimal, but have not been able to prove this. It is worth noting that the technique of detecting contact changes and computing the minimum between changes has been successfully applied to several other geometric extremal problems (De Pano and Aggarwal, 1983; O'Rourke, 1984; Toussaint, 1983).

A natural extension of this problem is to remove the convexity assumption. If the enclosing polygon P is convex then it can be readily seen that the minimal nested polygon must lie in between the enclosing polygon, P, and the convex hull of the enclosed polygon, i.e., $CH(Q)$. Now, the convex hull of Q, $CH(Q)$, can be computed in linear time; consequently, using the algorithm given in Section 4, we can find a minimal nested polygon (with k links) in $O(n \log k)$ time. For the case when both P and Q are simple, Suri and O'Rourke (1985) have presented an $O(n^2)$ algorithm for finding a minimal vertex polygon; Chan and Wang (1986) have improved the algorithm in Suri and O'Rourke (1985), to an $O(n \log n)$ algorithm. No algorithm with time complexity $o(n \log n)$ is known for the case when Q is convex but P is not. Also, the original problem as posed by Victor Klee, for polytopes in arbitrary dimensions, remains open even in the case of three dimensions.

Finally, we mention the following related result due to Edelsbrunner and Preparata (1987): given a set of p blue points and a set of q red points in the plane, Edelsbrunner and Preparata show that a convex polygon with minimum number of vertices that separates the blue points from the red, can be found in $O((p+q)\log(p+q))$ time. Although it seems that the problem considered in there is somewhat related to ours, the techniques and algorithms presented are entirely different.

Acknowledgments

The work of the first four authors was partially supported by NSF Grants MCS-83-04780 and DCR-83-51468, and that of the last author by NSF Grant DCR-84-01633.

Received May 31, 1988; accepted December 30, 1988

REFERENCES

AGGARWAL, A., CHANG, J. S., AND YAP, C. K. (1985), Minimum area circumscribing polygons, *Visual Comput.* **1**, 112–117.

AGGARWAL, A., KLAWE, M. M., MORAN, S., SHOR, P. W., AND WILBER, R. (1987), Geometric applications of a matrix searching algorithm, *Algorithmica* **2**, 209–233.

WANG, C. A., AND CHAN, E. P. F. (1986), Finding the minimum visible vertex distance between two non-intersecting simple polygons, *in* "Proceedings, 2nd Annu. ACM Symp. on Computational Geometry," pp. 34–42.

CHANG, J. S., AND YAP, C. K. (1984), A polynomial solution for potato-peeling and other polygon inclusion and enclosure problems, *in* "Proceedings, 25th IEEE Conference on Foundations of Computer Science," pp. 408–417.

DE PANO, N. A. A., AND AGGARWAL, A. (1983), Finding restricted k-envelopes for convex polygons, *in* "Proceedings 22nd Allerton Conference on Communications, Control and Computing," pp. 81–90.

DORI, B., AND BEN-BASSAT, M. (1983), Circumscribing a convex polygon by a polygon of fewer sides with minimum area addition, *Comput. Vision Graphics Image Process.* **24**, 131–159.

DEPANO, A. (1987), "Polygon Approximation with Optimized Polygonal Enclosures: Applications and Algorithms," Ph.D. thesis, The John Hopkins University.

EDELSBRUNNER, H., AND PREPARATA F. P. (1987), Minimum polygonal separation, *Inform. and Comput.*, **77**, 218–232; Technical Report, Univ. if Illinois at Urbana-Champaign.

KLEE, V., AND LASKOWSKI, M. C. (1985), Finding the smallest triangle containing a given convex polygon, *J. Algorithms* **6**, 359–375.

O'ROURKE, J. (1984), "Finding Minimal Enclosing Boxes," Tech. Report, The Johns Hopkins University.

SURI, S., AND O'ROURKE, J. (1985), "Finding Minimal Nested Polygons," Tech. Report, The Johns Hopkins University.

TOUSSAINT, G., (1983), Solving geometric problems with rotating calipers, *in* "Proceedings, IEEE MELECON '83."

O'ROURKE, J., AGGARWAL, A., MADILLO, S., AND BALDWIN, M (1986), An optimal algorithm for finding minimum enclosing triangles, *J. Algorithms* **7**, 258–268.

AHO, A. Y., HOPCROFT, J., AND ULLMAN, J. D. (1974), "Design on Analysis of Algorithms," pp. 5–12, Addison-Wesley, Reading, MA.