International Journal of Computational Geometry & Applications Vol. 2, No. 2 (1992) 191–214

A PARALLEL ALGORITHM FOR ENCLOSED AND ENCLOSING TRIANGLES

SHARAT CHANDRAN*
NTT Data Communications Systems Corporation
66-2 Horikawa-cho, Saiwai-ku, Kawasaki
Japan 210.

and

DAVID M. MOUNT†
Department of Computer Science and Institute for Advanced Computer Studies
University of Maryland, College Park, Maryland 20742, USA.

Received 30 November 1990
Revised 25 September 1991

ABSTRACT

We consider the problems of computing the largest area triangle enclosed within a given n-sided convex polygon and the smallest area triangle which encloses a given convex polygon. We show that these problems are closely related by presenting a single sequential linear time algorithm which essentially solves both problems simultaneously. We also present a cost-optimal parallel algorithm that solves both of these problems in $O(\log\log n)$ time using $n/\log\log n$ processors on a CRCW PRAM. In order to achieve these bounds we develop new techniques for the design of parallel algorithms for computational problems involving the rotating calipers method.

Keywords: Minimum enclosures, convex polygons, parallel algorithms, rotating calipers.

1. Introduction

The problems of finding minimum area enclosing and maximum area enclosed k-gons arise from the desire to approximate many-sided convex polygons by polygons with fewer sides. One application is in the area of collision avoidance in robotics.[4,5] It is easier to test interference of objects with fewer sides, and once interference is established, a more refined analysis can be made.

Algorithms for finding maximal enclosed k-gons and minimal enclosing k-gons were presented by Boyce, et al.[3] and subsequently were improved by Aggarwal and Park.[2] In the latter case the maximum area inscribed k-gon, as well as the minimum

*Most of this work was completed when this author was with the Center for Automation Research, College Park, MD 20742-3411, USA. Author's current address is: Computer Science and Engineering Department, IIT Powai, Bombay, India 400076.

†The work of this author was supported by National Science Foundation Grant CCR–89–08901.

area circumscribed k-gon can be computed in $O(kn + n \log n)$ time where n is the number of vertices in the polygon.

These sequential algorithms are not known to be optimal for general values of k and n. The cases of finding the minimum enclosing and maximum enclosed triangles ($k = 3$) are the only cases where optimal sequential algorithms are known. An $O(n)$ algorithm for computing the largest area triangle enclosed in a given convex polygon was given by Dobkin and Snyder.[6] The problem of finding the minimum area triangle enclosing a given convex polygon was first considered by Klee and Laskowski where an $O(n \log^2 n)$ algorithm was given.[8] This was subsequently improved to $O(n)$ by O'Rourke, Aggarwal, Maddila and Baldwin.[11]

Both linear time sequential algorithms are roughly based on the method of "rotating calipers"[12] to generate a finite set of intermediate triangles from which the optimum is selected. In spite of the outward similarity between the two problems, there is quite a difference in the details of the algorithms and their proofs of correctness. As an indication of the dissimilarity, an $O(\log n)$ time and $O(n)$ processor parallel algorithm was given by Aggarwal, Chazelle, Guibas, O'Dunlaing and Yap[1] for finding the minimum enclosing triangle; however, they state that their methods do not yield a better than $O(\log^2 n)$ parallel algorithm for the problem of finding the maximum enclosed triangle. The model of parallel computation that they use is the CREW PRAM, a parallel shared memory machine with concurrent-read and exclusive-write.

In this paper, we consider the problem of computing the maximum area enclosed triangle and the minimum area enclosing triangle for a given convex polygon. We show that these problems are much more closely related than had been previously thought. Both can be solved easily once a particular set of $O(n)$ intermediate triangles, which we call *P-stable triangles*, have been computed. These intermediate triangles are a superset of the P-anchored triangles introduced by Klee and Laskowski[8] for finding minimum enclosing triangles.

We prove that given a convex polygon P, the set of P-stable triangles can be computed in $O(n)$ sequential time and $O(\log \log n)$ parallel time on a CREW PRAM with $n/\log \log n$ processors. We show that by generating these triangles in $O(\log \log n)$ time, we can determine the family of all minimum enclosing and maximum enclosed triangles for a convex polygon. On the CREW PRAM we can find the extremum of n numbers in $O(\log n)$ time. As a consequence, we have improved the parallel complexity of finding maximum enclosing triangles on this model, and matched the complexity of finding minimum enclosed triangles. If we use a stronger model, such as the CRCW (concurrent read, concurrent write) PRAM machine, we have an algorithm that runs in $O(\log \log n)$ time using $O(n/\log \log n)$ processors for both problems. Note that concurrent writes are resolved by the rule that two processors writing to the same memory cell must both write the same value, and is thus the weakest of all concurrent write models.[7] Thus, we have also substantially improved the CRCW result of Aggarwal and Park[2] which considers our problems amongst others.

The approach of generating $O(n)$ triangles and then computing the extremum

has a lower bound of $\Omega(\log n)$ on the CREW PRAM model (derived from the lower bound of $\Omega(\log n)$ for finding the extremum of n numbers). We are therefore essentially limited by this bottleneck. On the CRCW model, as simple a problem as computing the parity of n bits requires $\Omega(\log n/\log\log n)$ time with any *polynomial* number of processors. It is therefore no surprise that very few parallel algorithms achieve a sublogarithmic time bound. Our parallel algorithm, however, belongs to this category.

In the next section we introduce P-stable triangles, and present a series of lemmas that characterize the geometry of the problem. In the development of efficient parallel algorithms, it is often the case that we need a sequential algorithm for the problem as a subroutine. To this end, the results of Section 2 are exploited in the development of the sequential algorithm that we sketch in Section 3. In Section 4 we present a series of increasingly efficient parallel algorithms. Finally, we make some concluding remarks in the last section.

2. Preliminaries

Throughout this paper, when referring to maximum and minimum triangles, we mean triangles of maximum or minimum area. Throughout, P will denote a convex polygon with n vertices. When referring to P we mean the boundary of P formed from the clockwise circular sequence of its vertices and edges. For two distinct points a and b on P, we use the notation (a, b) to denote the open interval of points lying strictly between a and b on P clockwise from a to b, and $[a, b]$ is the closure of this interval. Note that a and b need not be vertices, that is, they may lie on the interior of edges of P

It is easy to show that the vertices of any maximum triangle enclosed in a convex polygon must contact the polygon's boundary, as must the edges of any minimum enclosing triangle. We will assume that any triangle that encloses P or is enclosed in P is represented so that its points of incidence with P (vertices or edges) can be determined in $O(1)$ time. We say that a triangle that encloses P is *flush* with P if some edge of P is a subsegment of one of the sides of the triangle. A line L is said to *support* a convex polygon P if P and L intersect, and P lies entirely in one of the closed halfspaces defined by L. We begin with two lemmas that summarize the local properties of maximum enclosed and minimum enclosing triangles.

Lemma 2.1 (Dobkin and Snyder, 1979)

(i) If t is a maximum enclosed triangle for a convex polygon P then for each vertex v of t, the line drawn through v and parallel to the opposite side of t is a line of support for P.

(ii) Given a convex polygon P, there exists a maximum enclosed triangle in P whose vertices are a subset of the vertices of P.

The second statement is a restatement of Theorem 2.1 from Dobkin and Snyder.[6] The first statement follows from the unimodality observation: If $x_i x_k$ is a chord of P, then the function $area(x_i x_j x_k)$ is unimodal as x_j assumes values on the perimeter of the polygon. Suppose t is a maximum enclosed triangle with $x_i = a$ and $x_k = b$.

If the line through the third vertex and parallel to the chord ab is not a line of support, from the above, the area can always be increased by moving the third vertex along the perimeter.

Lemma 2.2 (Klee and Laskowski, 1985)

(i) If T is a minimum enclosing triangle for P then the midpoint of each side of T intersects P.

(ii) If T is a minimum enclosing triangle then at least one side of T is flush with P.

With respect to part (ii) above, Depano[5] has shown that there is a minimum enclosing triangle having at least two sides flush with P. We begin by considering a class of enclosing triangles, from which we will later extract enclosed triangles. When solving a problem by rotating calipers, it is often convenient to consider a larger class of triangles, which include all local minima as a subset. For this reason we introduce the notions of P-supported and P-stable triangles.

Definition 1 *Given a convex polygon P, an enclosing triangle is* P-supported *if the midpoints of two of its sides, called the* legs, *contact P. (The third side, called the* base, *may contact P arbitrarily.)*

Observe that the condition for local minima is enforced only on two sides, rather than all three. When referring to a P-supported triangle, ABC, we will adopt the naming convention that C is the vertex opposite the base. We will assume throughout that P-supported triangles are represented so that we know which sides are its legs and which side is its base. It follows from elementary geometry that the midpoints a and b of the legs AC and BC, respectively, form a line segment that is parallel to and is half the length of the base AB.

Given the line L supporting P, there is, in general, an infinite family of P-supported triangles whose base lies on the line L. There is a sense in which all such triangles are equivalent, which was observed earlier.[11]

Lemma 2.3 *Given a convex polygon P and a supporting line L, let S denote the set of P-supported triangles whose bases lie on L.*

(i) S is nonempty, and all the elements in S have the same area, and their corresponding legs share the same points as midpoints.

(ii) If S contains more than one element then the midpoints a and b are vertices of P, and the set of apexes of S form a line segment, parallel to L, and at twice the height above L as the line segment ab. The endpoints of this segment can be found in $O(1)$ time (see Figure 1).

Since we are interested in the areas of enclosing triangles, any local transformation that preserves the area of a P-supported triangle will not affect the final output. Thus, given a P-supported triangle, in $O(1)$ time we can select one endpoint (the rightmost, say) of the set of allowable apexes to form the *canonical* P-supported triangle having this base direction (see Figure 1).

The computational difficulty with P-supported triangles (even in canonical form) is that there are an infinite number of them, corresponding to each possible line of support for P forming the base of the triangle. To limit the set to a finite collection

of triangles, but encompassing enough to contain all locally minimum enclosing triangles, Klee and Laskowski introduced the notion of a P-anchored triangle.[8,11] Such a triangle is simply a P-supported triangle whose base is flush with P. One notable feature of the definition of P-anchored triangles is its asymmetry, in that the anchoring condition applies only to the base edge and not to the legs. In the case of minimum enclosing triangles this condition is sufficient to solve that problem. To encompass the set of triangles needed for the maximum enclosed triangle problem we generalize this definition as follows.

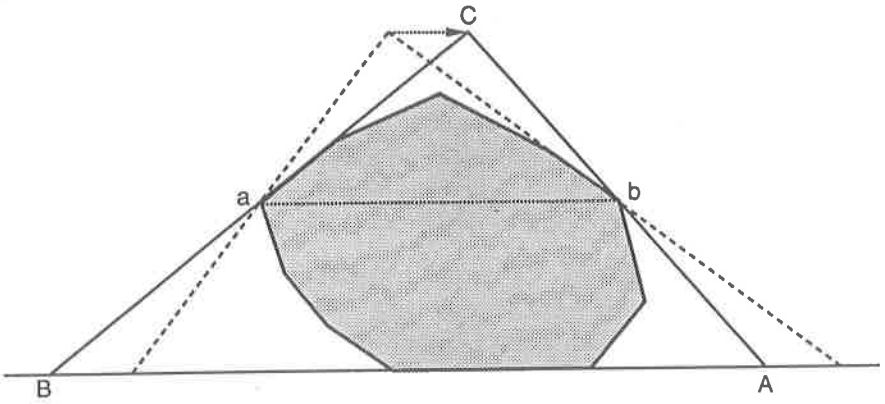

Figure 1: The canonical P-supported triangle ABC.

Definition 2 *A P-supported triangle is* P-stable *if either*

(i) the base is flush with P, or

(ii) one of the legs is flush with an edge of P and has as its midpoint a vertex of this edge.

Clearly every P-anchored triangle is P-stable and every P-stable triangle is P-supported. To draw the connection between enclosing and enclosed triangles we introduce the notion of an inner triangle.

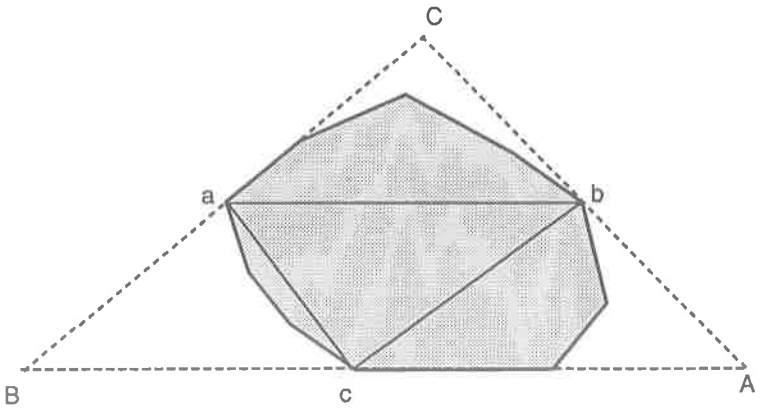

Figure 2: The inner triangle abc.

Definition 3 *Given a P-supported triangle ABC, let a and b be the midpoints of the legs BC and AC, respectively, and let c be any point on the base AB that also lies on P. The triangle abc is an* inner triangle *of ABC. The* base *of the inner triangle is the edge ab, the remaining sides are the* legs*, and c is the* apex *(see Figure 2).*

If the base of a P-supported triangle is flush with an edge of P, then there are an infinite number of inner triangles all of equal area (by sliding the apex of the inner triangle along the edge of P parallel to the base of the inner triangle) and hence we will adopt the convention of selecting the furthest eligible clockwise point for the apex of the inner triangle. Thus we can talk about "the" inner triangle for a given P-supported triangle. Note further that the operation of mapping a P-supported triangle into a canonical form does not alter its inner triangle. A number of simple observations regarding inner triangles can be made at this point.

Lemma 2.4 *Let P be a convex polygon, let T be a P-supported triangle, and let t be the inner triangle for T. Then*

(i) t is uniquely determined from the orientation of T's base,
(ii) the bases of T and t are parallel, and
(iii) the area of T is four times the area of t.

Proof. Observation (i) follows from Lemma 2.3 and the fact that the inner triangle is defined by the P-supported triangle. Observation (ii) follows from the elementary fact that the line joining the midpoints of two edges of a triangle is parallel to the other edge. Observation (iii) follows by observing that the inner triangle has a base parallel to and one half the width of the base of the outer triangle, and has height one half the height of the outer triangle. □

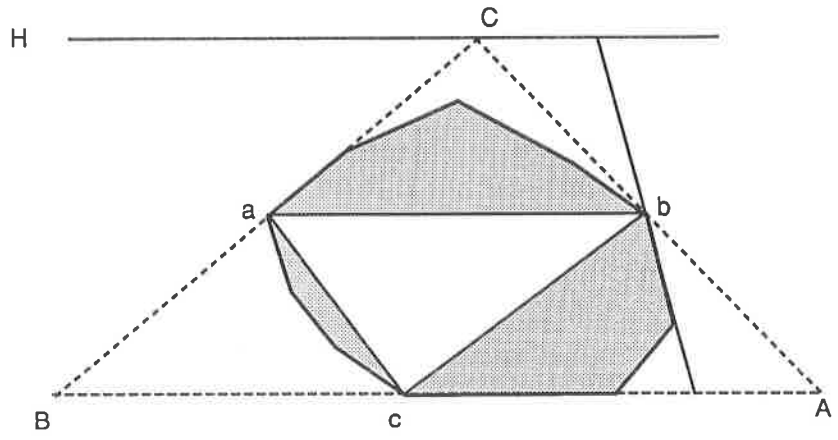

Figure 3: Generating an outer triangle ABC from an inner triangle abc.

It is obvious how to generate the unique inner triangle for a given P-supported triangle (given P). Conversely, given an inner triangle abc (of some P-supported triangle), the unique *canonical* outer triangle ABC can be found in $O(1)$ time as follows. Think of ab as being horizontal, and directed from left to right. The base

AB is parallel to ab and passes below through c. The apex C lies on a line H at twice the height of ab above this base. Extend the edges of P lying just clockwise from a and b until they intersect H. The leftmost of these two intersection points is the apex C of the outer triangle in canonical form (see Figure 3). (If the line from C through b is not a line of support for P then it follows that abc cannot be an inner triangle.)

The fundamental observation linking the two problems of finding minimum enclosing and maximum enclosed triangles is the following.

Lemma 2.5

(i) There exists a minimum enclosing triangle that is a canonical P-stable triangle.

(ii) There exists a maximum enclosed triangle that is an inner triangle of some canonical P-stable triangle.

Proof. Observation (i) is an immediate consequence of Lemma 2.2 together with the fact that all P-anchored triangles are P-stable. To prove the second observation, let abc be a maximum enclosed triangle such that a, b and c are vertices of P. Form a triangle $T = ABC$ by drawing a line through each vertex of abc that is parallel to the side opposite the vertex. By Lemma 2.1(i), the sides of triangle T support P, hence T encloses P. It is easy to derive from elementary geometry that the sides of T have their midpoints at a, b and c. (For example, to see that c is the midpoint of AB observe that abAc and abcB are parallelograms sharing the common edge ab). Now map abc into canonical form (if needed) by sliding c parallel to ab. This does not alter the area of abc. Similarly map triangle T into canonical form (if needed) by sliding the vertex C parallel to edge AB. Let us assume without loss of generality that side BC becomes flush with P. Since a is a vertex of P and since BC is flush with P, T is a canonical P-stable triangle, and abc is its inner triangle. □

Thus, to determine the minimum enclosing triangle and maximum enclosed triangle it suffices to compute the set of canonical P-stable triangles (which we show later to be a finite set) and to select the outer triangle of minimum area and the corresponding inner triangle of maximum area. In order to proceed we must first introduce the following *interspersing property*, which is basic to all rotating calipers algorithms.[1,3,6,11] This property states that as the orientation of the base edge advances in a clockwise direction, the vertices of the corresponding inner triangle move clockwise about P. This lemma is not stated in exactly this form by O'Rourke, et al.,[11] but it follows directly from the proof of their Lemma 2, together with the simple observation that as the orientation of the base turns clockwise, the point of contact with the base, c, also moves clockwise around P. We may associate any directed line segment AB with an angle of orientation by considering the angle of the vector $B - A$.

Lemma 2.6 (O'Rourke, Aggarwal, Maddila, and Baldwin,1986)
Let ABC be a P-supported triangle with base AB and inner triangle abc. (See Figure 4.) *Let $A'B'C'$ be a P-supported triangle whose base $A'B'$ lies between AB and BC in clockwise angular orientation. Further, let $a'b'c'$ be the latter's inner*

triangle. Then $c' \in [c, a]$, $b' \in [b, c']$ *and* $a' \in [a, b']$.

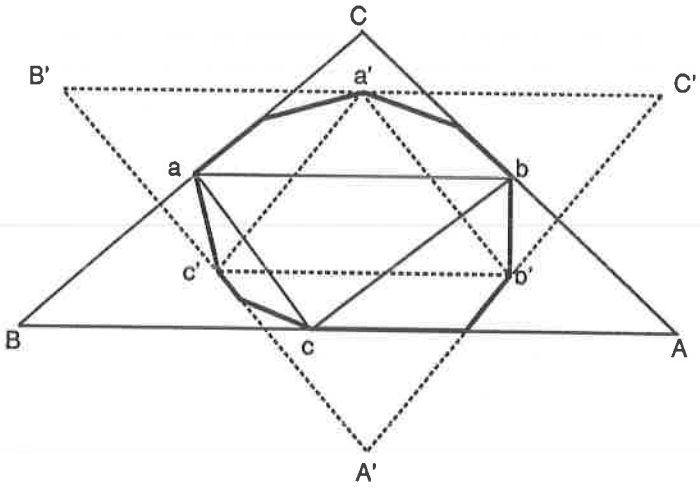

Figure 4: Interspersing property.

3. Sequential algorithm

It follows from Lemma 2.5 that, in order to find the maximum enclosed and minimum enclosing triangles, it suffices to generate all the canonical P-stable triangles. We first develop a sequential algorithm for this problem. As in O'Rourke, et al.,[11] our algorithm is an example of the rotating calipers technique, generating all P-anchored triangles in clockwise order according to the orientation of the base edge. Due to the more general class of triangles being generated, the algorithm is somewhat more involved than theirs.

The rotating caliper method by which we generate the set of canonical P-stable triangles is to imagine the following continuous process. Consider a support line "rolling around" the boundary of P clockwise through every angle through 360 degrees. For each angle of orientation, there is a unique canonical P-supported triangle having its base lying on the support line of this orientation. To simulate this process discretely we consider only those orientations at which the the triangle is P-stable. We will show that there are only $O(n)$ such *anchoring orientations* and show how to move from one anchoring orientation to the next in a constant amount of time. Finally we will select the smallest among all the P-stable triangles, and the largest among the corresponding inner triangles.

The algorithm begins by generating one P-anchored triangle ABC whose base is flush with some edge of P. This can be done in $O(\log^2 n)$ time by the algorithm given by Klee and Laskowski[8] (but see also Lemma 4.2). ABC is transformed into canonical form if it is not already in this form, and the corresponding inner triangle abc (see Figure 2) is computed in $O(1)$ time. Recall that c is a vertex of P, but a and b need not be vertices. Let e_a, e_b, and e_c denote the edges of P on which a, b and c lie, respectively (taking the next clockwise edge if they lie on vertices). Since

ABC is in canonical form, at least one of the sides of BC or CA is flush with e_a or e_b respectively (as, for example, in Figure 2). We describe how to generate the next P-stable triangle ordered clockwise by the angle of the base in $O(1)$ time. We will show that after $O(n)$ iterations we will have generated all P-stable triangles. Let a' and b' and c' be the vertices of P which (strictly) follow a, b and c in clockwise order, respectively. In other words, a', b' and c' are the clockwise endpoints of e_a, e_b and e_c (see Figure 5).

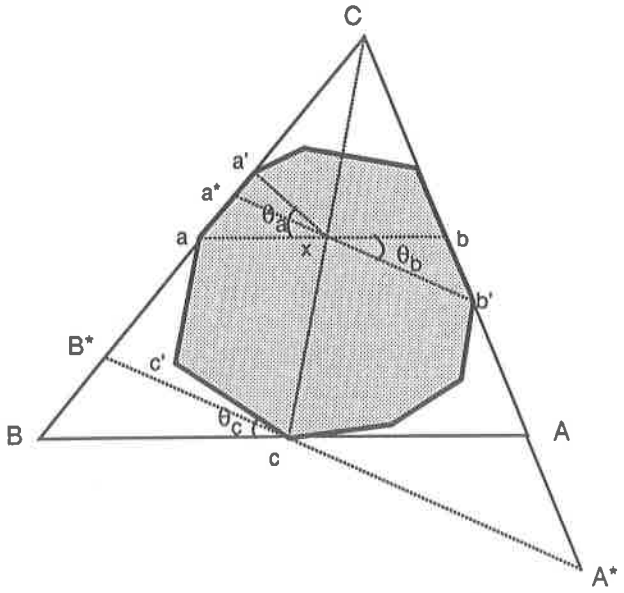

Figure 5: The case of two flush legs.

Case 1: Two flush legs. For the first case, assume that triangle leg BC is flush with the polygon edge e_a, and leg CA is flush with e_b. Recall that if the base is flush with an edge, then c is the extreme clockwise vertex of the edge, so that any small incremental clockwise rotation of the base revolves about c. Let x denote the midpoint of the segment Cc. Since a and b are midpoints of BC and CA, it follows that x is also on the base ab of the inner triangle. As the base rotates clockwise around c through a small clockwise angle θ, the legs of the P-supported triangle will remain flush with the same edges, implying that the apex of the P-supported triangle will remain fixed at C. Since c will lie on the base and the apex C is fixed, it follows that the line segment joining the midpoints of the legs will pass through x. Thus as θ rotates through a small clockwise angle, the midpoints of the resulting inner triangle, $a(\theta)$ and $b(\theta)$, will travel clockwise around the boundary of P (as predicted by the interspersing property), so that the segment joining them passes through x and is parallel to the rotating base. Observe that the angular change of θ is equal to the angular change of $a(\theta)$ and $b(\theta)$ with respect to x (see Figure 5).

The next anchoring event occurs either when the base rotates so far as to become flush with e_c, or when $a(\theta) = a'$, or when $b(\theta) = b'$. The other two events occur when

the rotating line that passes through x intersects one of the vertices a' or b'. Observe that in all three cases the resulting triangle is P-stable (although not necessarily in canonical form). Thus, we compute three clockwise angles corresponding to these events. Let $\theta_c = \angle Bcc'$, be the angle of rotation of the base of the P-supported triangle that aligns the base with e_c, let $\theta_a = \angle axa'$ be the angle of rotation of the line through x for which this line passes through a' and let $\theta_b = \angle bxb'$ be the corresponding angle for b'. Let $\theta = \min(\theta_a, \theta_b, \theta_c)$. The next P-stable triangle is found by rotating the base clockwise through the angle θ. Depending upon the value of θ, e_a and/or e_b, or c and e_c are updated.[10] If the resulting triangle is not canonical, it is converted into canonical form in constant time.

Case 2: One flush leg. The second case is if one of the legs is flush and the other is not. Let us first assume that BC is flush with $e_a = aa'$, but CA is not flush with $e_b = bb'$. This implies that b is a vertex of P. As the base rotates through a small clockwise angle θ, the leg BC will remain flush with the same edge, and as a result the other leg must rotate to maintain b as its midpoint. In order to explain this rotation, consider a line L passing through A and parallel to BC. Observe that every line segment passing through b and having its endpoints on the lines L and (the linear extension of) BC has b as its midpoint. As the base edge of the outer triangle rotates through a small clockwise angle θ about c, the point $A(\theta)$ of the outer triangle will necessarily travel along L and the point $B(\theta)$ will travel along BC (from B towards C). The location of the third vertex $C(\theta)$ of the outer triangle is determined by shooting a ray from $A(\theta)$ through b until it hits the line BC. The moving vertex of the inner triangle $a(\theta)$ lies on the edge e_a clockwise from a, such that the base of the inner triangle $a(\theta)b$ is parallel to the base of the outer triangle.

The next anchoring event occurs either when the base rotates so far as to become flush with e_c, when $a(\theta) = a'$, or when the leg passing through b becomes flush with edge e_b. Observe that in all three cases the resulting triangle is P-stable (but may not be in canonical form in the second case). The first two events occur at the angles $\theta_c = \angle Bcc'$ and $\theta_a = \angle aba'$. Let D be the line extending the edge e_b, and let x be the point at which D intersects the the extension of BC. (See Figure 6). At the moment of the third event, $C(\theta)$ coincides with x. Let y be the unique point on D such that b is the midpoint of xy. The third event will occur when the base has rotated through the angle $\theta_b = \angle Acy$.

Let $\theta = \min(\theta_a, \theta_b, \theta_c)$. The next P-stable triangle is found by rotating the base clockwise through the angle θ.[10] Depending upon the value of θ, e_a and/or e_b, or c and e_c are updated. The resulting triangle is not necessarily canonical, in which case it is converted into canonical form.

The other case of a single flush edge is if CA is flush with e_b, but BC is not flush with e_a. This case is symmetric to Case 2 with the roles of A, a and a' swapped with B, b, and b' respectively, with the only exception that θ_c is still defined to be the angle $\angle Bcc'$.

The above algorithm runs in $O(1)$ time per case. Also observe that it provides an alternative proof of the interspersing lemma, since in each case it can be observed

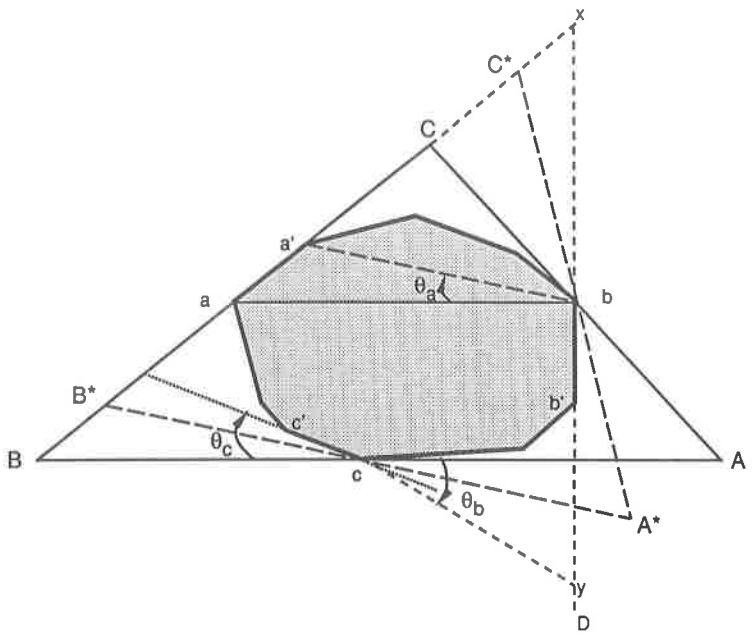

Figure 6: The case of one flush leg.

that the contact points of the inner triangle move monotonically clockwise. We can now establish the total time needed to generate all the P-stable triangles as $O(n)$, by proving that there are $O(n)$ P-stable triangles.

Lemma 3.1 *Given a convex polygon P with n vertices, there are at most $5n$ P-stable triangles and at most $3n$ canonical P-stable triangles.*

Proof. Consider a sequence of the P-stable triangles ordered by the angle of the base edge and the corresponding inner triangles. As seen from the algorithm which has just been outlined, each inner triangle differs from its predecessor in the sequence by advancing at least one of its contact points clockwise along an edge of the polygon to a vertex of the polygon, followed possibly by a canonical shift in the case that one of the legs' contact points that reaches a vertex. (It is trivial to see that in the process of advancing from one P-stable triangle to another that we do not skip over any P-stable triangles.) By the interspersing property, each of the contact points can be advanced through n vertices before returning to the initial configuration. Hence, there are at most $3n$ canonical P-stable triangles (one for each advance of a contact point) and at most $5n$ P-stable triangles (allowing for each of the $2n$ advances of a leg's contact point to be followed by a canonical shift). □

Summarizing the results of this section we have.

Theorem 3.1 *Given a convex n-sided polygon P, both the largest area enclosed triangle and the smallest enclosing triangle can be computed in $O(n)$ (sequential) time by computing the set of P-stable triangles.*

4. Parallel algorithm

In this section we present a parallel algorithm that computes all of the $O(n)$ canonical P-stable triangles. Given $n/\log\log n$ processors, our algorithm achieves a running time of $O(\log\log n)$ in generating a set of candidate triangles. Observe that it is nontrivial to generalize the sequential algorithm given in the previous section, since the rotating calipers approach seems inherently sequential. The parallel algorithm is similar in structure to the $\sqrt{n}$-divide and conquer algorithm presented by Aggarwal, et al.[1] for solving the restricted problem of finding all the P-anchored triangles, but the "back-and-forth subdivision" used by their algorithm has been replaced by simpler redundancy scheme (owing in part to the more symmetric nature of P-stable triangles over P-anchored triangles). Once all of the P-stable triangles have been computed, it is an easy matter to determine the triangles of maximum and minimum area in $O(\log n)$ time on the CREW (concurrent read, exclusive write) model. On the more powerful CRCW (concurrent read, concurrent write) model, the time to compute the maximum is $O(\log\log n)$, where concurrent writes are allowed only if the processors writing to the shared memory cell all write the same value.

Letting $T_1(n)$ denote the time taken to solve the problem using a single processor, then a parallel algorithm is *cost-optimal*, or simply *optimal*, if the time-processor product (or *cost*) $p(n)T_p(n)$ is $O(T_1(n))$. We first present a non-optimal n processor $O(\log n)$ time algorithm. Later, we improve the algorithm to make it faster, with a running time of $O(\log\log n)$ still using the same number of processors. Finally, we present a cost-optimal improvement using only $O(n/\log\log n)$ processors.

4.1. A logarithmic time algorithm

Our first parallel algorithm operates in a series of $O(\log\log n)$ stages. Intuitively, the algorithm exploits the interspersing lemma (Lemma 2.6) at each stage to compute the P-stable triangles. During successive stages we compute ever larger subsets of the P-stable triangles whose corresponding inner triangles are interspersed according to Lemma 2.6. Thus, later stages can refine their search in the ever narrowing gaps between cyclically adjacent inner triangles.

The key to the efficiency of the algorithm will be the need to guarantee that between any cyclically adjacent pair of triangles at each stage, there are at most a small number (made precise below) of vertices of P. To see the difficulty in maintaining this condition, consider the case of two inner triangles, abc and $a'b'c'$. By judiciously choosing a' to be close to a, we can guarantee that the number of vertices of P in the range $[a, a']$ is small. However, we cannot infer that the number of vertices of P within $[b, b']$ and $[c, c']$ are correspondingly small since these depend on subtle relationships between the shape of the polygon and the distribution of vertices. In order to circumvent this problem, we use a redundancy scheme, by producing three new triangles, one whose first vertex is close to a, one whose second vertex is close to b, and one whose third vertex is close to c. Irrespective of how these three triangles intersperse, this will guarantee that there are a small number

of vertices between every pair of adjacent inner triangles.

To define how these inner triangles are generated we will need to introduce the notion of a P-stable triangle that is anchored in a particular way to a side of P and investigate how to generate these triangles. We say that a P-stable triangle is *a-anchored* at a vertex v, if v is the midpoint of the leg BC (implying that the vertex a of the inner triangle coincides with v) and the leg BC is flush with one of the two edges incident to v. The notion of a b-anchored triangle is defined analogously for side CA. A P-stable triangle is c-anchored at v if its base AB is flush with the edge lying counterclockwise from v (recalling that the midpoint anchoring condition for P-stable triangles does not apply to the base). Figure 8 shows an a-anchored triangle anchored at vertex v, and Figure 3 shows a c-anchored triangle anchored at vertex c. Since these three cases encompass all the possible anchoring conditions for P-stable triangles we have, it is clear that every P-stable triangle falls into one of these three categories.

Observe that there are actually two a-anchored triangles and two b-anchored triangles at a given vertex v, owing to the fact that the triangle may be flush with either incident edge about v. To simplify the presentation we will only consider the a- and b- anchored triangles only for the edge lying counterclockwise from v, since the other case is similar. Note that these triangles may not be in canonical form.

To present the idea for the parallel algorithm more formally, for each stage s, $s \geq 1$, let $n_s = \left\lfloor n^{1/2^s} \right\rfloor$. Thus, $n_1 = \lfloor \sqrt{n} \rfloor$, $n_2 = \lfloor \sqrt[4]{n} \rfloor$, and so on. Define V_s to be the subset of vertices consisting of every n_s-th vertex in P. We compute an a-, b-, and c-anchored triangle at every vertex in V_s. We will maintain these three types of triangles in a single merged sequence. By applying the interspersing lemma, it follows that for any pair of triangles, which are consecutive in this sequence, the corresponding inner triangles abc and $a'b'c'$ will define three (possibly empty) intervals along the boundary of P, $[a, a']$, $[b, b']$ and $[c, c']$ such that each interval will contain at most $O(n_s)$ vertices. After $O(\log \log n)$ stages, we will have computed a P-stable triangle anchored (in every possible way) at every vertex, implying that we have computed all the P-stable triangles.

Since we shall be shifting our attention from P-stable triangles as defined earlier to P-stable triangles that are anchored, we need the following observations to characterize these triangles and to discuss the computational complexity our our algorithms. First we observe that for c-anchored triangles, by fixing the direction of the base edge and applying Lemma 2.3(i) we have the following.

Lemma 4.1 *Given a convex polygon P and a vertex v on P, there exists a unique (canonical) P-stable triangle T_c that is c-anchored at v.*

Likewise, there exists unique P-stable triangle T_a and T_b that are, respectively, a-anchored and b-anchored. This is not immediately obvious because of the asymmetry in the definition, but later in Lemma 4.3 we will give a constructive proof that triangles T_a and T_b also exist, and are unique.

It follows therefore that it is enough to focus on generating all anchored triangles. The next set of lemmas discuss the complexity of the generation. In previous works, it was discussed in detail how to find c-anchored triangles.[8,11] In particular, the

following result was given by Klee and Laskowski[8] in a restricted form for edges rather than points.

Lemma 4.2 *Given a convex polygon P, and given a vertex v of P, each of the endpoints of the inner triangle of the c-anchored triangle anchored at v can be found in $O(\log^2 n)$ time by a binary search that uses $O(\log n)$ time per probe.*

Proof. The vertex c of the inner triangle is just v and hence can be found in $O(1)$ time. Let e be the edge whose clockwise endpoint is v. To find the vertex a of the inner triangle, consider the line D containing e. (Let us assume that P is so oriented that D is horizontal and P is above D. See Figure 7.) For each point x on P, let $h(x)$ denote the height of x above D. The function h induces two closed clockwise intervals on P, the left interval L along which h is strictly increasing and the right interval R along which h is strictly decreasing. Clearly $a \in L$ and $b \in R$, since the directed line segment ab is horizontal and directed from left to right.

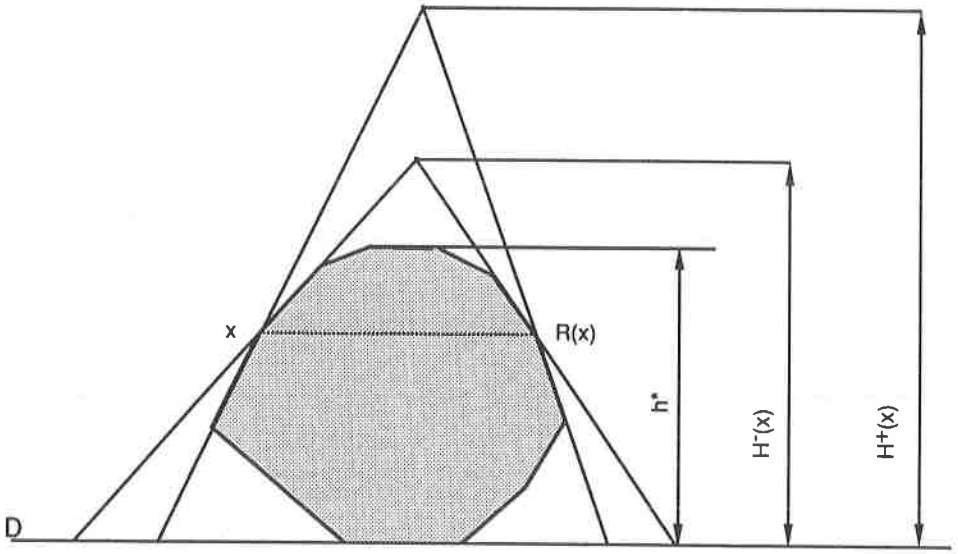

Figure 7: Height of support functions.

Let h^* be the maximum height of P, that is, h^* is the height of the first and last endpoints of R and L, respectively. For each point $x \in L$, let $R(x)$ be the unique point in R whose height above D is the same as x's. Let $H^-(x)$ denote the height of the lowest point (with respect to D) that lies above D from which a line of support can be drawn through both x and $R(x)$, and let $H^+(x)$ denote the height of the highest point from which a line of support can be drawn through both x and $R(x)$. (Observe that $H^-(x)$ and $H^+(x)$ will be identical unless either x or $R(x)$ is a vertex.) If there is no such point or if this point lies below D let $H^-(x)$ or $H^+(x)$ be positive infinity (see Figure 7). By convexity, as x varies clockwise along L, $H^-(x)$ and $H^+(x)$ decrease monotonically from $+\infty$ to h^*. It follows that there exists a unique point $a \in L$ such that $H^-(a) \leq 2h(a) \leq H^+(a)$. Let $b = R(a)$. It follows from convexity that there are two lines of support passing through a and b, respectively, that intersect at the height $2h(a)$. The triangle defined by these lines

of support and D define a c-anchored P-supported triangle, and the points a and b are the vertices of its inner triangle.

Given an arbitrary point x, we can determine $R(x)$ in $O(\log n)$ time by binary search through R, and we can determine $H^-(x)$ and $H^+(x)$ in $O(1)$ additional time by considering the lines of support formed by extending the edges lying just above and below x and $R(x)$. By applying binary search along L, we can find a and b in $O(\log^2 n)$ time. □

This Lemma provides the basis for Klee and Laskowski's $O(n \log^2 n)$ algorithm for finding the smallest enclosing triangle, since from Lemma 2.2, it suffices to determine the P-supported triangle that is c-anchored at each vertex.

We also need supplementary results on finding a-anchored and b-anchored triangles. We will state the result for a-anchored triangles, and an analogous result holds for of b-anchored triangles by a symmetric argument.

Lemma 4.3 *Given a convex polygon P and a vertex v on P, each of the endpoints of the a-anchored triangle anchored at v can be found in $O(\log^2 n)$ time by a binary search that uses $O(\log n)$ time per probe.*

Proof. Let D denote the line extending the edge e whose clockwise endpoint is v. Assume that P is oriented so that D is vertical directed upwards with P lying to its right (see Figure 8). Let e' be the edge whose counterclockwise endpoint is v. If abc is an inner P-supported triangle that is a-anchored at v, then b cannot lie on e. If the point b lies on e' then b is the other endpoint of e'. Let θ_0 denote the clockwise angle from D to e'. For each angle θ, $\theta_0 \leq \theta < \pi$ there is a unique point x_θ on P such that the clockwise angle from D to the segment vx_θ is θ. Let B_θ denote the lowest point on line D lying above v such that there is a line of support from B_θ through x_θ. (Observe that it is only when x_θ is a vertex that there is any choice for B_θ. If there is no such point then let B_θ be a point at infinity above v. For each angle θ, $0 < \theta < \pi$, there is a unique line of support S_θ whose angle is θ with respect to D such that P lies above this line. Let C_θ denote the point (necessarily on or below a) at which this line of support intersects D.

As θ increases from θ_0 to π, the length of the segment vC_θ decreases strictly and continuously from some finite value to 0, whereas the length of vB_θ increases (discontinuously) from 0 to infinity. There exists a unique minimum angle ϕ for which the length of vB_ϕ is greater than or equal to the length of vC_ϕ. Let $b = x_\phi$, and let c be the leftmost point at which S_ϕ touches P from below. It follows that there exists a unique line of support passing through b that equalizes the distances vB_θ and vC_θ. This line of support, together with D and S_ϕ define a unique P-stable triangle that is a-anchored at v, and whose inner triangle consists of $a = v$, b, and c.

For an arbitrary angle θ we can determine whether $\theta \leq \phi$ by constructing the points B_θ and C_θ and comparing their distances from v. To do this we need to locate the point x_θ and determine the support line S_θ, each of which can be performed in $O(\log n)$ time by the convexity of P. By applying binary search, we can determine the edges or vertices of P on which b and c must lie. It is a straightforward exercise to determine the exact location of these points in $O(1)$ additional time (based

on a case analysis of whether b and c are known to lie on edges or vertices or a combination of the two). □

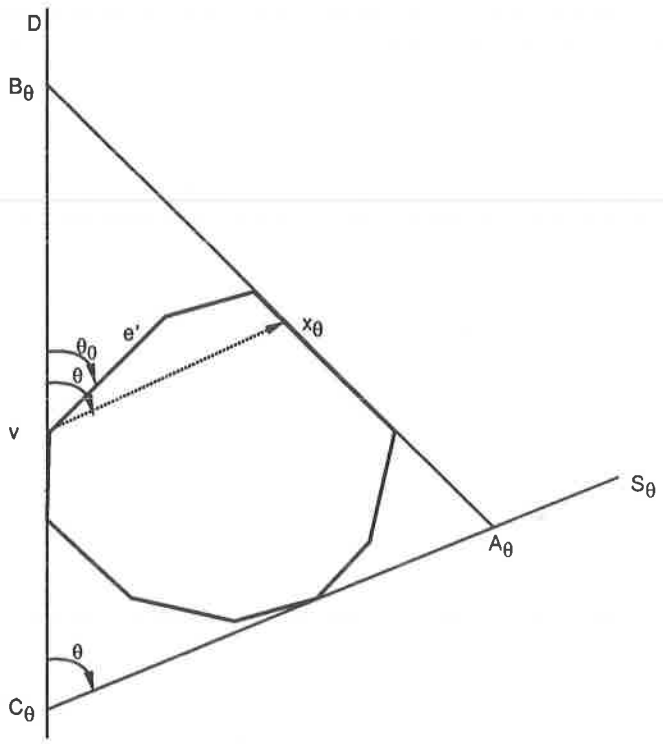

Figure 8: Generating the a-anchored P-stable triangle.

We now have the necessary machinery to describe the basic structure of the parallel algorithm. Let the current stage be $s \geq 1$, and recall that $n_s = \left\lfloor n^{1/2^s} \right\rfloor$. We maintain the invariant that at the end of stage s, for each vertex $v \in V_s$ we compute an a-anchored, b-anchored and c-anchored triangle at v. Thus there will be $3|V_s| = 3n/n_s$ triangles computed altogether by the end of stage s (not all necessarily distinct). These triangles are maintained in cyclic order according to the angles of their base edges (thus the three triangles anchored at a given vertex may be quite far apart in this ordering). For each vertex in $v \in V_s$ we allocate n_s processors with which to compute a-, b-, and c-anchored triangles at v. This implies that n processors are used altogether at any stage. As mentioned earlier, within $O(\log \log n)$ stages, we will have computed all the P-stable triangles. The largest and smallest of these triangles can be determined easily in $O(\log n)$ time using n processors on the CREW model, and in $O(\log \log n)$ time on the CRCW model.

At the beginning of stage s the cyclically ordered inner triangles generated so far define three separate partitions of the boundary of P into closed intervals. In particular, if $a_0, a_1, \ldots, a_{m-1}$ are the a-vertices of the $3|V_s|$ inner triangles generated so far, the a-partition consists of the half-closed intervals $[a_i, a_{i+1})$ (indices taken modulo m). (Initially, all three partitions are trivial, consisting of a single interval.)

There is an analogously defined b-partition and a c-partition. It follows from the interspersing lemma that the inner triangle of any subsequent P-stable triangles will lie within the closures of some three intervals corresponding to cyclically adjacent triangles $[a_i, a_{i+1}]$, $[b_i, b_{i+1}]$, and $[c_i, c_{i+1}]$. Each a-interval (an interval of the a-partition) has at most n_{s-1} vertices of P, since we explicitly constructed a-anchored triangles at the points of V_{s-1} during the previous stage. Similar bounds hold for the b- and c-intervals. Thus, by the interspersing lemma, as more and more triangles are generated, the location of subsequent inner triangles becomes more and more constrained.

The data are stored as follows. Each cyclically adjacent pair of inner triangles abc and $a'b'c'$ that have been generated so far define three corresponding intervals $[a, a')$, $[b, b')$, and $[c, c')$, one in each of the three partitions. These three intervals are linked to each other, and each vertex of P contains three pointers to the intervals in each of the three partitions containing this vertex. Vertices of inner triangles are labeled as either a-, b-, or c-vertices.

The processing at stage s is as follows. Each vertex $v \in V_s - V_{s-1}$ is allocated n_s processors with which to compute its a-, b-, and c- anchored triangles.

Let us first consider the case of finding the c-anchored triangle. We determine the c-interval $[c, c')$ that contains v and access the corresponding b- and a-intervals $[b, b')$ and $[a, a')$. There are at most $n_{s-1} = n_s^2$ vertices of P in each of the two intervals. Lemma 4.1 and Lemma 2.6 assures us that there is a unique c-anchored triangle whose b vertex lies among those n_{s-1} points.

Let V_a denote the sequence of vertices in $[a, a']$ and $[b, b']$. Each of these sequences contains at most $n_s^2 + 2$ vertices. By Lemma 4.2, for each $x \in V_a$, we can determine in $O(\log n_s)$ sequential time whether or not the a vertex of the c-anchored triangle satisfies $a \in [c, x]$. Further, we can determine two vertices of V_a bounding a in $O(\log^2 n_s)$ time. However, we can employ n_s processors, and the following result due to Kruskal[9] to reduce the time complexity for the complete operation to $O(\log n_s)$ for this step. (We set $Q(n) = \log n_s$ in the application of the result.)

Lemma 4.4

(i) Consider a sorted list a_i, $1 \le i \le n$, and a value x, $a_1 \le x \le a_n$. Let $Q(n)$ denote the sequential time needed to access an arbitrary member a_i. We can find the index i such that $a_i \le x \le a_{i+1}$ in $O(Q(n))$ time on a CREW PRAM using $\sqrt{n}$ processors.

(ii) Two sorted lists of size m each may be merged in $O(\log\log m)$ time using m processors on the CREW PRAM.

Once the triangles have been computed, we need to update the ordered set of triangles, and hence refine the three partitions. First note that the number of vertices of V_s in any interval is at most $n_{s-1}/n_s = n_s$. Let us consider the case of an a-interval, since the other cases are similar. By the interspersing lemma, there are three new sets of inner triangle endpoints that might fall into the interval $[a, a')$. They are the a-vertices of each of the following: (1) the a-anchored triangles generated in $[a, a')$, (2) the b-anchored triangles in $[b, b')$, and (3) the c-anchored triangles in $[c, c')$. We can maintain three sorted lists containing these a-vertices.

Each list is associated with the sorted list of vertices of V_s lying within each of the three intervals that was responsible for generating this triangle vertex. Each list can have at most n_s elements in it. Using the processors allocated to one of the vertices of V_s in one of the three intervals, we merge these three lists into one sorted list and replace the single interval $[a, a')$ with the refined list of intervals. (Note that at least one of the three intervals must contain a vertex of V_s, for otherwise no new triangles would be generated in any of the intervals.) This stage requires no more than $O(\log\log n_s)$ time (Lemma 4.4(ii)).

Another update of the data structures is necessary. For every vertex $v \in V_{s+1}$ we seek the appropriate partition in each of the three lists. That is, for each vertex of $P \in V_{s+1}$ in $[a, a')$ a new label is desired indicating its new interval membership. This is done by yet another merge as follows. We assign n_s processors to each vertex $v \in V_s$. These are responsible for updating the pointers of at most n_{s+1} new vertices. The partition that these vertices belong is necessarily in the refinement of $[a, a')$ and the size of this is at most $3n_s$. From Lemma 4.4, we can complete this in $O(\log\log n_s)$ time.

This leads us to

Theorem 4.1 *The largest area triangle enclosed within a given n-sided convex polygon and the smallest area triangle which encloses a given convex polygon may be computed in $O(\log n)$ time using n processors.*

Proof. The algorithm to compute these triangles has been given above. To analyze its complexity, we note that the running time of stage s is $O(\log n_{s-1})$. The total running time for all the $\log\log n$ stages is

$$\sum_{s=1}^{\log\log n} \log n_{s-1} \leq \sum_{s\geq 0} \log n_s \leq \sum_{s\geq 0} \log n^{1/2^s} \leq \sum_{s\geq 0} \frac{\log n}{2^s} \in O(\log n).$$

□

4.2. An $O(\log\log n)$ time algorithm

In the previous section, we presented a parallel algorithm that runs in logarithmic time. In order to reduce the running time, we focus on the part that dominates the running time, which is in the generation of new anchored triangles. For example, in the generation of c-anchored triangles, the point c is known, and we have to find the remaining two vertices, say, $a*$ and $b*$. We use processors in parallel in the search for $b*$, but not in the search for $a*$. The crucial observation here is that using the convexity of the polygon P, we may use more than one processor even in the search for $a*$.

Consider the development of the algorithm in Lemma 4.2. (In the following, we use the terminology in the proof of Lemma 4.2.) Given an arbitrary point $x \in L$, we can determine $R(x)$ in $O(\log n)$ time. If this point x does not satisfy the relation $H^-(x) \leq 2h(x) \leq H^+(x)$ we move the point x either close to c or farther from c. Depending upon the direction in which we choose to move from the point x, we may introduce the notion of *low* and *high* points.[8,11] In the cited work, the formulation

is in terms of edges of P but here we choose to describe it in terms of the points of (the boundary of) P.

Informally, a point x is low if the binary search procedure along L decides to move in a direction clockwise from x. More formally, given a point $x \in L$, let γ_x be the point along the edge clockwise from x such that $h(\gamma_x) = 2h(x)$. The point x is said to be *low* if $\gamma_x R(x)$ intersects P above $R(x)$, *high* if $\gamma_{x-1} R(x-1)$ intersects P below $R(x-1)$ and *critical* if it is neither low nor high. (Here $x-1$ is the first vertex counterclockwise from x. Incidentally, this notion of low and high is further exploited by O'Rourke, et al.,[11] which uses geometric characterization to determine low or high status information for an edge on the left chain L without examining points at the same height on the right chain, even though the definition is in terms of the points along R. We do not use this refined characterization in our parallel algorithms.)

By a *regular* sample of vertices of P, we refer to a subset of the vertices of P by taking every k-th vertex of P for some value $1 \leq k \leq n-1$. Suppose we are given a regular sample of points along the left chain. Let us call each sampled point as *distinguished*, and assume that we have the low or high status information for each distinguished point. The observation that we need is the following:

Lemma 4.5 *Suppose the transition between low and high occurs at two distinguished vertices x and y on the left chain. Then the true unique c-anchored triangle has its a-vertex in the interval $[x, y]$ and the b-vertex in the interval $[u, v]$ where u is the first vertex anticlockwise from and including $R(x)$ and v is the first vertex clockwise from and including $R(y)$.*

Proof. Klee and Laskowski[8] have shown that in order of increasing height from the line D, the left chain consists of a sequence of low vertices followed by at most three vertices which are critical, followed by a sequence of high vertices. From this it follows that the c-anchored triangle will have its a-vertex in the interval $[x, y]$. From the definition of $R(x)$, the third vertex is also in the claimed interval. (Note that $R(x)$ is unaffected by the sampling procedure, and that x and y need not be necessarily adjacent in the sampled set.) □

We are now in a position to reduce the running time for the construction of c-anchored triangles, but first we need the following result, paraphrased from Kruskal.[9]

Lemma 4.6 *Let $0 < \epsilon \leq 1$.*

(i) Consider a sorted list a_i, $1 \leq i \leq n$, and a value x, $a_1 \leq x \leq a_n$. We can find the index i such that $a_i \leq x \leq a_{i+1}$ in $O(1)$ time on a CREW PRAM using n^ϵ processors.

(ii) Two sorted lists of size m each may be merged in $O(1)$ time using $m^{1+\epsilon}$ processors on the CREW PRAM.

Note that these results are stronger forms of Lemma 4.4.

Lemma 4.7 *Given a convex polygon P with n vertices, and given vertex v of P, each of the endpoints of the inner triangle of the c-anchored triangle anchored at v can be found in $O(1)$ time with $\sqrt{n}$ processors.*

Proof. As before, the computation of vertex c is trivial. To find the vertex a of

the inner triangle for T, consider the line D extending the edge e whose clockwise endpoint is v. (Let us assume as before that P is so oriented that D is horizontal and P is above D). Using the ordinates of the vertices of L, the left chain of P, and the ordinates of the vertices of R, the right chain of P, as keys, we merge the two sets in parallel. This allows us to compute the point $R(x)$ for each vertex x. In $O(1)$ time we can check in parallel if the vertex x is low, high, or critical. Since there are exactly three vertices on L that are critical, one processor can report the critical point $a*$, and the point $b*$ such that the triangle with vertices c, $a*$ and $b*$ is the desired P-stable triangle (in its canonical form). The time for this computation is dominated by the merging operation which can be done in $O(\log\log n)$ time using n processors.

To reduce the running time still using n processors, we sample every $\sqrt{n}$ vertices along L to produce a chain L_1 with $\sqrt{n}$ vertices. We assign $\sqrt{n}$ processors to each vertex of L_1. For each distinguished vertex x, we obtain $R(x)$ in $O(1)$ time ($\epsilon = 0.5$ in Lemma 4.6(i) above). Next, we determine the low or high status of each of these vertices in $O(1)$ time. Lemma 4.5 assures us that the true critical points will lie in the transition from a low vertex l to a high vertex r. There are $O(\sqrt{n})$ points between two such distinguished vertices that mark the transition. We perform another search in the range $[l, r]$, once again in $O(1)$ time. At this point, we would have obtained three consecutive vertices of L that are neither low or high. The search for $a*$ can be then completed in $O(1)$ time using one processor.

To further reduce the running time, we bootstrap the n-processor, $O(1)$ time algorithm above, reducing the processor count to $\sqrt{n}$. This process is described below in more detail.

As above, we form the chain L_1. Now, with a pool of $\sqrt{n}$ processors, we can assign only one processor to each vertex x of L_1 and the search for $R(x)$ will take too long. Therefore we further sample every $n^{1/4}$ vertex of chain L_1 to produce the chain L_2 consisting of at most $n^{1/4}$ vertices. We assign $n^{1/4}$ processors to each vertex x of L_2 and obtain $R(x)$ for each of them. This takes $O(1)$ time using Lemma 4.6(i) with $\epsilon = 0.25$. At this point the transition between a low vertex and a high vertex contains $O(n^{1/4})$ vertices of L_1, and so we can apply this procedure again to find two consecutive points of L_1 to find the transition from a low vertex to a high vertex. This whole procedure takes $O(1)$ time using $\sqrt{n}$ processors.

Recall that between two consecutive points of L_1 that mark the transition, there are $O(\sqrt{n})$ points of L. We have just finished describing an algorithm that determines the transition between low and high with $\sqrt{n}$ points of L using $\sqrt{n}$ processors that takes $O(1)$ time. We apply this algorithm again to find two vertices that mark the transition from low to high such that the interval consists of $O(1)$ vertices of P. It is then easy to determine the unique points $a*$ and $b*$ that determine the canonical c-anchored triangle using a single processor. □

In order to complete the improvement of the algorithm, we need a notion of low and high for the case of a-anchored triangles as well, and we need to be able to compute these fast. The general strategy we follow in the computation is similar to one adopted for computing the c-anchored triangles. However, due to the asymmetrical

nature of the two kinds of triangles (the line segment bc of the inner triangle need not be parallel to the anchored edge), the restatement of this problem as a merge problem is slightly more complicated.

Consider, as in the proof of Lemma 4.3, the point x_θ. We can now specify it as *low* if the binary search procedure moves in the clockwise direction. More formally, x_θ is low if the length of segment vB_θ is less than the length of the segment vC_θ and is high if the length is strictly greater than the length of the segment vC_θ.

Suppose that in the process of computing the a-anchored triangles we are given a regular sample of points which constitute points for the b-vertices. If we were given the two vertices x and y where a transition occurs between low and high, then the unique b-vertex of the a-anchored triangle must lie in the interval $[x, y]$. Furthermore, the c-vertex of the a-anchored triangle must lie in the interval spanned by the c-vertices of the triangles generated for the points x and y. These statements follow from convexity and arguments presented in the proof of Lemma 4.5.

We can now state the result for a- and b-anchored triangles.

Lemma 4.8 *Given a convex polygon P with n vertices, and given vertex v of P, each of the endpoints of the inner triangle of the a- and b-anchored triangle anchored at v can be found in $O(1)$ time with $\sqrt{n}$ processors.*

Proof. Consider Figure 8. We are interested in determining whether the candidate point x_θ is a low point or a high point. We need this information without doing a binary search. Given the third vertex (c), we can draw a line of support and then compare the distances vB_θ and vC_θ. If $vB_\theta < vC_\theta$ then the point is low. If $vB_\theta > vC_\theta$ the point is high. If they are exactly equal, we have the desired triangle. Next, we use the same idea as in Lemma 4.7, but first we need to know how to compare two vertices of P. We used the vertex closest (in the sense of ordinate distance, assuming as in the construction that the base is horizontal) in the case of c-anchored triangles. Here, this is decided as follows. The point x_θ defines B_θ. We compare distances along a line perpendicular to the line segment vx_θ and passing through B_θ. Thus, following the sampling procedure, and using Lemma 4.6, we can compute the a-anchored triangle in constant time. The b-anchored triangle can be computed likewise. □

We now describe stage s of the algorithm. At this point the data consists of three separate partitions of a, b and c-vertices. They satisfy the following conditions: (i) Each cyclically adjacent pair of inner triangles abc and $a'b'c'$ define three corresponding intervals, which are linked together, (ii) Each of these intervals consists of at most $3n_{s-1}$ points, (iii) Each vertex of P contains three pointers to the intervals in each of the three partitions containing this vertex.

We perform the following three steps. For illustrative purposes, we describe c-anchored triangles and c-vertices.

Step 1: Compute anchored triangles. We allocate n_s processors to each vertex $v \in V_s - V_{s-1}$. Consider, for example the computation of the c-anchored triangle at a vertex v. Since each vertex points to the c-interval, we know the interval $[c, c']$ that contains v. Using the links between intervals, we know the bounding intervals $[b, b']$ and $[c, c']$. Each contain at most n_s^2 points of P. We use Lemma 4.7 to compute

the c-anchored triangle in $O(1)$ time. Note that when we compute the triangles, the links necessary in the invariant (i) above are satisfied.

Step 2: Refine partitions. We now have freshly computed n_s new c-anchored triangles, and hence generated, e.g. n_s c-vertices of P. As before, we had n_s processors standing by each of these vertices, for a total of n_s^2 vertices. We have also generated n_s c-vertices from the c-points of a-anchored triangles, and c-vertices from b-anchored triangles. The processors are used to merge these three lists in $O(1)$ time. (Lemma 4.6(ii), with $\epsilon = 1$).

Step 3: Update membership. Each vertex of $P \in V_{s+1}$ needs to be informed which c interval it belongs to. We have now computed the c interval for the n_s-th vertex that bounds this set. To extend this information for other vertices in V_{s+1} we again merge this set of vertices with the at most n_s vertices in $O(1)$ time, as described before.

At the end of these three steps, we would have maintained the invariant needed for stage $s+1$. To analyze the running time of stage s, we note that we have essentially reduced the problem to merging. Since there are $O(\log\log n)$ stages, and each stage takes constant time, the procedure we have described runs in $O(\log\log n)$ time.

4.3. A cost-optimal $O(\log\log n)$ time algorithm

In the previous section we described an algorithm that runs in $O(\log\log n)$ time using n processors. Such an algorithm does not have a processor-time product equal to the time for the sequential algorithm and therefore is not cost-optimal.

Cost-optimality is achieved by noting that we have essentially reduced the problem to problems of parallel searching, and parallel merging. Although the sorted lists that are being merged have different significance, the essential point to be noted is that we are merging two lists of size n_s using the processors available at stage s, which is n_s^2. Suppose we try to adopt our previous algorithm and allocate at stage s, $n_{s-1}/\log\log n$ processors instead of n_{s-1} processors to perform the merge. The total number of processors used is $O(n/\log\log n)$ since the number of such lists to be merged is n/n_{s-1}. We note from Kruskal's result[9] that the time for each stage is asymptotically $O(\log\log\log n)$ and since the total number of stages is $\log\log n$, we do not have a cost-optimal result. Thus, we need a better processor allocation strategy.

We first note from Lemma 4.6 that to achieve a running time of $O(1)$ per stage requires us to have a superlinear number of processors. In our algorithm in the previous section, we achieved this superlinearity very easily; in particular, we had quadratic number of processors. However, we note that with our pool of $n/\log\log n$ processors, we will be unable to sustain this superlinearity for $O(\log\log n)$ stages. In later stages we are forced to deal with an increasingly large number of points.

Instead, we run the above parallel algorithm for a certain number of stages. After that time, we will have generated many P-stable triangles. We switch over to a different algorithm to generate all remaining P-stable triangles.

Specifically, for any fixed δ, $0 < \delta < 1$, we will run the parallel algorithm

described above for S stages, where S is the greatest integer less than $\delta(\log\log n - \log\log\log\log n)$. Note that after S stages, every n_S-th vertex has its anchored triangles generated. Between any two vertices, the number of vertices n_S that do not have their anchored triangles generated is at most $n^{1/2^S}$. It can be verified that this number is $O(\log\log n)$. Thus there are n/n_S groups each of size n_S whose triangles have not been generated. We allocate one processor to each group and this processor uses the sequential algorithm of Section 3 to generate the triangles for its group. Clearly this stage runs in $O(\log\log n)$ time.

It remains to describe the process for the first S stages. At the start of each stage s we note that we have n/n_{s-1} groups and there are n_s points within each group that we seek to merge. We assign $n_s^{2-\delta}$ processors to each group. From Lemma 4.6(ii), the time to perform the merge is $O(1)$. The computation of the anchored triangles may also be done in $O(1)$ time using $n^{1-\delta}$ processors working in parallel. At each stage s we utilize no more than n/n_s^{δ} processors. Since we run the algorithm for S stages, this quantity is less than $n/\log\log n$.

Since each stage $s \leq S$ of the algorithm runs in $O(1)$ time, the time for the entire algorithm is $O(\log\log n)$. The optimal algorithm for the CREW PRAM is very similar, and it runs in $O(\log n)$ time. We therefore have the following theorem:

Theorem 4.2 *It is possible to generate all P-stable triangles for an n-sided convex polygon in $O(\log\log n)$ time using $n/\log\log n$ processors on a CREW PRAM.*

As a consequence of Theorem 4.2 and Lemma 2.5 and existing results in the literature, we report the following results:

Corollary 1 *The largest area triangle enclosed within a given n-sided convex polygon, and the smallest area triangle which encloses the given polygon may be computed in $O(\log n)$ time using $n/\log n$ processors on a CREW PRAM.*

Corollary 2 *The largest area triangle enclosed within a given n-sided convex polygon, and the smallest area triangle which encloses the given polygon may be computed in $O(\log\log n)$ time using $n/\log\log n$ processors on a CRCW PRAM.*

5. Concluding remarks

Given a convex n-sided polygon, we have shown that we can compute either an inscribed triangle (whose area is the maximum among all enclosed triangles), or an enclosing triangle (whose area is the minimum among all enclosing triangles) using a single algorithm. Although these problems are "duals" in some sense, earlier algorithms for the two problems were quite different. We are able to provide a new geometric characterization that enables us to show that these two problems are more similar than was thought earlier. Our sequential algorithm runs in optimal $O(n)$ time

Sublogarithmic time parallel algorithms are difficult to achieve. We provide a parallel algorithm for the two problems that runs in $O(\log\log n)$ time using $n/\log\log n$ processors on a CRCW model. We hope that our treatment for the parallel algorithm will provide a systematic mechanism to solve other problems which use the "rotating calipers" technique.

It is interesting to contrast our treatment of the problem with the presentation described in Aggarwal and Park[2] and Aggarwal, et al.[1] In the former we have a general strategy of converting the geometrical problem to the combinatorial problem of searching in monotone three-dimensional arrays. The problem is solved in $O(\log^2 n)$ time using $O(n)$ processors. The latter uses a somewhat involved "back-and-forth" subdivision method and results from geometry to obtain a $O(\log n)$ time algorithm (but are unable to extend this to the case of the maximum enclosed triangle). Our results follow from geometrical considerations. However, the algorithm does not extend directly to searching in totally monotone arrays because of the fact that a totally monotone two-dimensional array is not made up of many totally monotone one-dimensional sets.

References

1. A. Aggarwal, B. Chazelle, L. Guibas, C. O'Dunlaing, and C. Yap, "Parallel computational geometry", *Algorithmica*, **3** (1988) 293–327.
2. A. Aggarwal and J. K. Park, "Sequential searching in multidimensional monotone arrays", *29th Annual IEEE Symposium on Foundations of Computer Science*, White Plains, New York, Oct. 1988, pp. 497–512.
3. J. E. Boyce, D. P. Dobkin, R. L. Drysdale, and L. J. Guibas, "Finding extremal polygons", *SIAM Journal on Computing* **14** (1985) 134–147.
4. J. S. Chang and C. K. Yap, "A polynomial solution for potato-peeling and other polygon inclusion and enclosure problems", *25th Annual IEEE Symposium on Foundations of Computer Science*, Singer Island, Florida, Oct. 1984, pp. 408–416.
5. N. A. N. DePano, "Polygonal Approximations with Optimal Polygonal Enclosures", PhD thesis, Department of Computer Science, The Johns Hopkins University, 1987.
6. D. P. Dobkin and L. Snyder, "On a general method for maximizing and minimizing among certain geometric problems", *20th Annual IEEE Symposium on Foundations of Computer Science*, Oct. 1979, pp. 9–17.
7. F. Fich, P. Ragde, and A. Wigderson, "Relationships between concurrent models of parallel computations", *SIAM Journal on Computing* **17** (1988) 606–627.
8. V. Klee and M. L. Laskowski, "Finding the smallest triangles containing a given convex polygon", *Journal of Algorithms* **6** (1985) 359–375.
9. C. Kruskal, "Searching, merging and sorting in parallel computation", *IEEE Trans. Comput.* **C-32** (1983) 942–946.
10. D. M. Mount and S. Chandran, "A unified approach to finding enclosing and enclosed triangles", *Proceedings of the 26th Allerton Conference on Communications, Control, and Computing*, 1988, pp. 87–96.
11. J. O'Rourke, A. Aggarwal, S. Maddila, and M. Baldwin, "An optimal algorithm for finding minimal enclosing triangles", *Journal of Algorithms* **7** (1986) 258–269.
12. G. Toussaint, "Solving geometric problems with the 'rotating calipers' ", *Proceedings of the IEEE MELECON 83*, 1983.