CS81017-R

ON THE COMPUTATION OF MINIMUM ENCASING RECTANGLES AND SET DIAMETERS

D.C.S. Allison
M.T. Noga

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

Key Words: Computational geometry, convex hull, applied computational complexity.

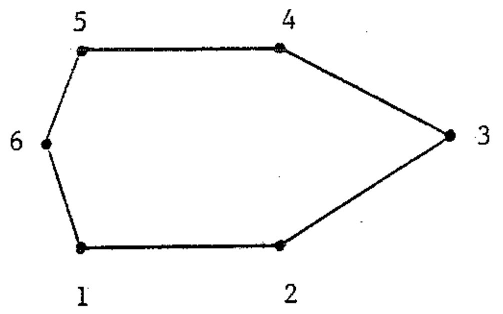
Abstract: Two new algorithms are described for the following problems: given a set of N points in the plane determine (i) the rectangle of minimum area which will completely cover (or encase) the set and (ii) the two points that are farthest apart (diameter of the set). Both algorithms have O(NlogN) time complexity and are based upon a similar strategy.

1. *INTRODUCTION*

In the field of computational geometry two problems of recent interest are: given a set of N points in the plane determine (i) the rectangle of minimum area which will completely cover (or encase) the set and (ii) the diameter of the set (the two points that are farthest apart). We show here that these two problems may be solved by algorithms which employ a common technique. Throughout, we will refer to this technique as the HIGHPOINT strategy. As we shall see, the HIGHPOINT strategy leads to efficient algorithms for both of these problems because all highest points can be found in O(N) time. We formally prove this result in the next section.

2. *HIGHPOINT STRATEGY*

The highest points problem is the following: given a convex polygon determine the vertex point (points) which has (have) the greatest perpendicular distance above each edge; Fig. 1.



edge	highpoint(s)
12	4,5
23	5
34	1
45	1,2
56	3
61	3

Fig. 1.

Assuming that no three vertex points on the polygon are collinear, each edge can have no more than two highpoints. We define these points to be the left and right highpoints. The left highpoint is the counterclockwise successor of the right highpoint along the polygonal boundary. It is also possible for several adjacent edges to have the same highpoint. An algorithm to determine all highest points follows in which we make use of the scalar product [13,14]:

$$S_p = x_p(y_i - y_j) + y_p(x_j - x_i) + y_j x_i - y_i x_j, \qquad (1)$$

where the magnitude of S_p is directly proportional to the height of a point (x_p, y_p) above the line ij with coordinates (x_i, y_i) and (x_j, y_j).

HIGHEST_POINTS

Input: A doubly linked-list containing a convex polygon in standard form. (A polygon is in standard form if its vertices occur in counterclockwise order beginning with the vertex that has least y-coordinate. All vertices must be distinct with no three consecutive vertices collinear [1].)

Output: All edges and their highest point(s).

Step 1: Locate the highest point(s) above an initial edge ij of the polygon. This can be carried out by scanning counterclockwise examining each pair of successive vertices A and B until the condition $S_B \le S_A$ holds, Fig. 2. The scan starts with A = cclock(j), B = cclock(A). If $S_A = S_B$ then output both A and B as the highpoints (A is the right highpoint, B is the left highpoint); otherwise output A as the lone highpoint.

Step 2: (HIGHPOINT strategy) Move to the next edge. Let i = j, j = cclock(i), and find its highest point(s). Start the scan at the highpoint from the previous edge (or left highpoint if there are two), examining successive pairs of vertices A and B until $S_B \le S_A$. Output the highpoint(s) (as in Step 1). Repeat Step 2 until all edges have been traversed.

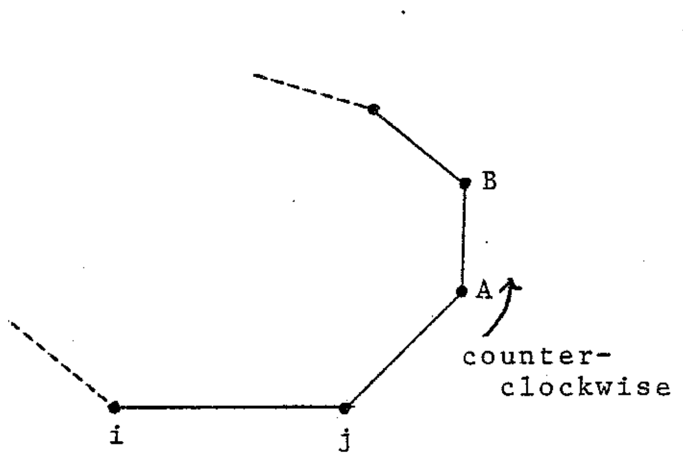


Fig. 2.

<u>Theorem 1</u>: *Algorithm HIGHEST_POINTS produces the highest points above each edge of an N-vertex polygon in O(N) time.*

Proof: Step 1 requires N/2 scalar product calculations on the average, but never more than N. We can start the scan for the highest point of a new edge at the previous highpoint because all points between the new edge and the old highpoint are perpendicularly less distant than the old highpoint, Fig. 3. (Only points in the shaded area can be on the polygon yet not be the previous highpoint.) As each edge is traversed the scan for highpoints commences in a counterclockwise direction, never clockwise. Furthermore, the scan for highpoints can never reach the edge presently being traversed. Since in Step 2, N-1 edges are traversed it follows that never more than O(N) vertices of the polygon are examined as possible highpoints. The actual number of scalar product calculations is approximately 3N-3. •

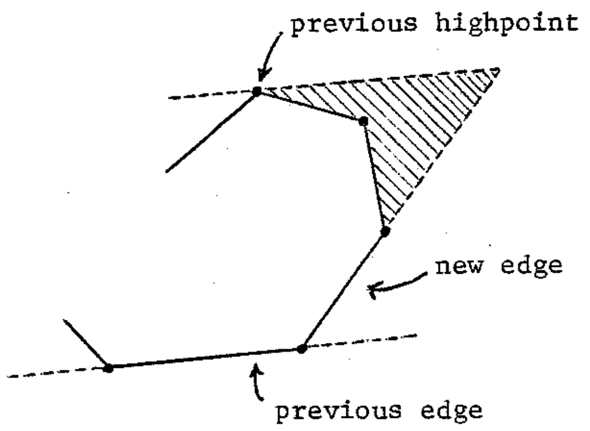


Fig. 3.

3. *THE MINIMUM ENCASING RECTANGLE OF A SET OF POINTS*

Theorem 2. (Freeman and Shapira [2]) *The minimum area encasing rectangle (MER) of a convex polygon must have one side collinear with an edge of the polygon.*

The theorem suggests the following algorithm to compute the MER of a set of points.

MER

Input: A set of N points expressed in cartesian form.

Output: The area of the smallest encasing rectangle of the set.

Step 1: Compute the convex hull of the set.

Step 2: Compute the encasing rectangle collinear to each edge and take the smallest of these as the MER.

The convex hull is the minimum area convex polygon containing the set of planar points. Several algorithms with O(NlogN) worst-case time complexity [3,4,5] are available for the computation of this polygon. We show that Step 2 can be computed in O(N) operations using the HIGHPOINT strategy. Step 1 will thus dominate, leading to an O(NlogN) algorithm for the determination of the MER of a set of N points. For worst-case analysis we assume that all N points are 'on' the hull and passed to Step 2.

We start at the bottommost edge ij of the hull. (We assume that our hull algorithm computes the convex hull in standard form.) As in Step 1 of algorithm *HIGHEST_POINTS*, we scan counterclockwise applying the scalar product formula (1) to each pair of adjacent vertices A and B until $S_B < S_A$, Fig. 4. The perpendicular distance from A to line ij may then be computed by solving the following set of simultaneous equations to determine the point C where a perpendicular line from A crosses ij:

$$y_C - y_i = m(x_C - x_i),$$
$$y_C - y_A = -(x_C - x_A)/m,$$

where $m = (y_j - y_i)/(x_j - x_i)$ is the slope of line ij. The Euclidean distance formula can be used to calculate the distance ℓ between points A and C, Fig. 4. Note that ℓ is the length of one side of the encasing rectangle collinear to edge ij.

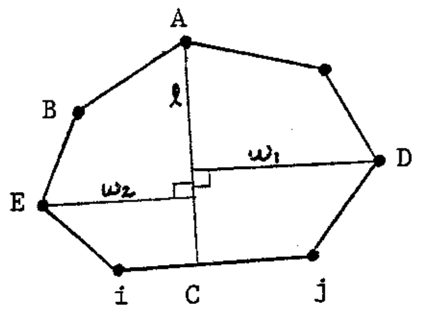


Fig. 4.

The length of the other side of the rectangle may be computed in a manner analagous to the procedure above. Starting at vertex j, scan counterclockwise to find the point D highest above line AC, and similarly scan counterclockwise starting at A to find the vertex point E highest above CA, Fig. 4. Perpendicular lines emanating from D and E may then be dropped onto AC and CA, and their lengths, w_1 and w_2, computed by again solving the appropriate set of simultaneous equations and applying the Euclidean distance formula. The sum of these lengths is the overall width of the encasing rectangle. Thus, the area AREA of the encasing rectangle collinear to edge ij is $\ell * (w_1 + w_2)$.

We then move to the next edge, and let i = j, j = cclock(j). To determine the area of the encasing rectangle with one side collinear to this line, we employ the HIGHPOINT strategy. To find new highpoints A, D and E we use the previously determined highpoints as the starting points for the new highpoint scans. Once these points are found the area of the new rectangle NEWAREA can be computed by solving 3 sets of simultaneous equations and applying the Euclidean distance formula in each case. If NEWAREA < AREA the value of AREA is replaced by

NEWAREA. This process may be applied iteratively to determine each successive encasing rectangle. When all edges have been traversed AREA will contain the area of the smallest encasing rectangle.

Theorem 3: *The MER of a convex polygon can be found in O(N) time.*

Proof: For each rectangle, as a result of the HIGHPOINT strategy, we can compute A, D and E in an average of k operations, where k is a fixed constant. And, for each rectangle, we must solve 3 sets of simultaneous equations and apply the Euclidean distance formula. This requires $\leq$ c operations, where again c is some fixed constant. Therefore the total time required is $(k + c)N = 0(N)$. •

Because the HIGHPOINT strategy can be used to enumerate all encasing rectangles we have the following:

Theorem 4: *The MER of a set of N points can be found in O(NlogN) time.*

Also, the convex hull of a simple polygon can be found in O(N) time [6]. Thus, we have:

Theorem 5: *The MER of an N-sided simple polygon can be found in O(N) time.*

(A polygon is simple if and only if nonconsecutive sides do not intersect and consecutive sides intersect only at a single point.)

The MER algorithm was coded in FORTRAN and tested on an IBM 3032 (FORTX,OPT=2). Uniform random variates were generated on the boundary of an ellipse and passed to a modified Graham convex hull algorithm [3,7,8]. For all sample sizes all points remained on the hull and were passed via a doubly circular linked-list to a subprogram which computed the MER of a convex polygon. In Table 1 we give the time taken by this subprogram. 5 realizations of 100 runs were made for each sample size. Average times are in seconds and appear in the table along with the standard deviations.

Table 1

N	MER/convex polygon
125	3.556 (.0114)
250	7.096 (.0344)
500	14.184 (.0532)
1000	28.116 (.0729)

As expected, the results indicate that the MER/convex polygon subprogram exhibits O(N) time complexity.

4. *DIAMETER OF A SET*

The problem here is: given N points in the plane, find the two that are farthest apart. At least two algorithms exist for the determination of these points. One is a naive, but completely straightforward algorithm, which examines the distance between all possible pairs of points. Since there are $\binom{N}{2} = N(N-1)/2$ such pairs, this algorithm runs in $O(N^2)$ time. Another is a clever O(NlogN) algorithm due to M.I. Shamos [9] based upon the following two theorems:

Theorem 6: (Hocking [10]) *The diameter of a set must be realized by two points on its convex hull.*

Theorem 7: (Yaglom [11]) *The diameter of a convex polygon is the greatest distance between parallel lines of support.*

(A line of support L of a polygon P meets the boundary of P and P lies entirely on one side of L.) As Shamos realized, the following algorithm suggests itself:

DIAM

Input: A set of N points expressed in cartesian form.

Output: The endpoints and length of the diameter.

Step 1: Find the convex hull of the set.

Step 2: Enumerate all pairs of vertices for which parallel lines of support exist (Shamos refers to these as 'antipodal' pairs [9]), and take the pair separated by the largest distance.

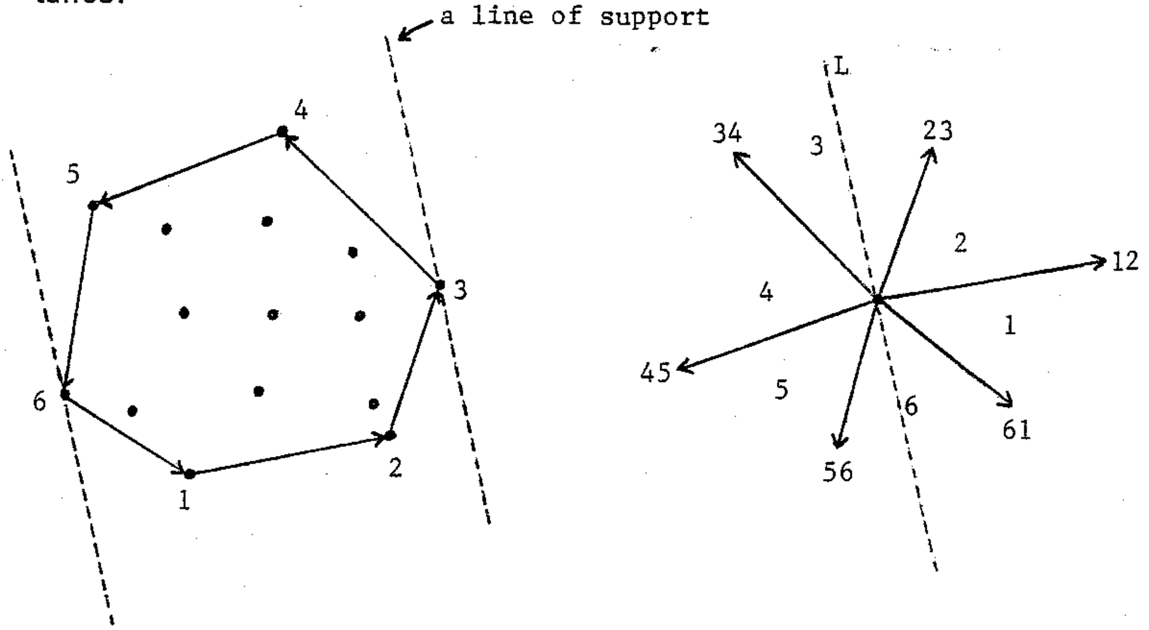


Fig. 5. Finding Antipodal Pairs

The complexity of Step 1 is O(NlogN). However, if all N points are on the hull, Step 2 may be computed in O(N) time by using a data structure suggested by Shamos [9]. The idea is to treat the edges of the convex hull as vectors and translate them to the origin, Fig. 5. In this transformation, edges go to vectors, and vertices to sectors. All antipodal pairs may then be found by extending an infinite line L through the origin and rotating it counterclockwise. The antipodal pair does not change until L passes through some new vector of the diagram. In Fig. 5, pair 3,6 turns into 4,6 as L passes through vector 34, 4,6 turns into 4,1 and so on. It is clear that, for each vector passed, a new antipodal pair is determined. (If two vectors are simultaneously encountered 4 new antipodal pairs result.) Because there are N vectors to pass, it follows that by scanning sequentially around the diagram (swinging line L through at least 180 degrees; Fig. 5), O(N) time is required to compute all antipodal pairs. As a consequence we have:

Theorem 8: *The diameter of a set of N points can be found in O(NlogN) time.*

Theorem 9: *The diameter of a convex polygon can be found in O(N) time.*

Theorem 10: *The diameter of a simple polygon can be found in O(N) time.*

We now show that there is another method by which the antipodal pairs may be found. In Fig. 6, AB and BC are two lines of support of the convex hull passing through vertex point B. Let the highest points above these two lines be H_1 and H_2, respectively. (H_1 is the right highpoint of AB if AB has two highpoints and correspondingly H_2 is the left highpoint of BC if BC has two highpoints.) Because of the convexity property of the hull, parallel lines of support to AB and BC can pass through points H_1 and H_2. Thus, (B,H_1) and (B,H_2) are antipodal pairs.

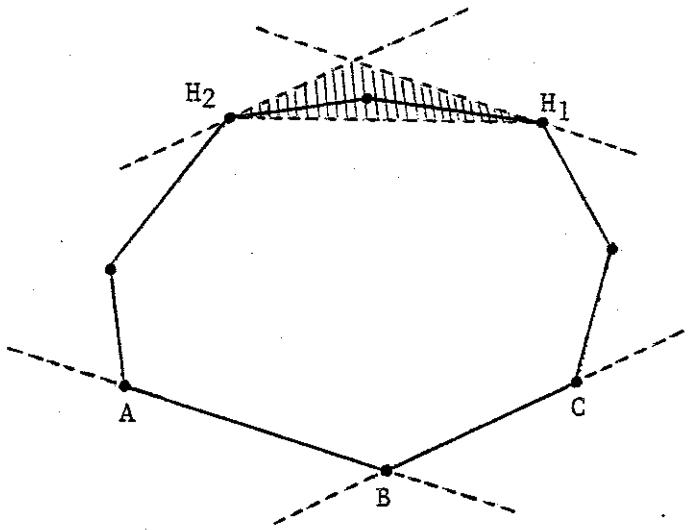


Fig. 6.

Furthermore, only the chain of points on the convex hull between H_1 and H_2 will admit to parallel lines of support in conjunction with point B; Fig. 6 (shaded area). This result is valid for any 3 consecutive points on the hull and thus we have proved:

Theorem 11: *For any vertex point B on a convex polygon, the antipodal pairs corresponding to B are the sequence $(B,H_1)...(B,H_2)$, where H_1 and H_2 are the vertices that are the highest points above each of the edges adjoining B.*

Applying Theorem 11 and the HIGHPOINT strategy leads to the following algorithm:

LARGEST_ANTIPODAL_PAIR

Input: A convex polygon in standard form.

Output: The endpoints of the diameter and its length.

Step 1: Start with the bottommost point i on the hull and its two common edges. Find the highest point above each edge (these points could possibly be the same). Label these points H_1 and H_2. Compute the interpoint distances between i and all vertices in the chain $H_1 \ldots H_2$. Keep only the largest of these distances L_p and its corresponding antipodal pair (A_1,A_2). Let INIT_POINT = H_1.

Step 2: Let i = cclock(i). Find the highest point above each edge adjoining i. Set $H_1 = H_2$ (or H_1 = clock(H_2) if two highpoints) and use the HIGHPOINT strategy to find H_2. Compute distances between i and all vertices in the chain $H_1 \ldots H_2$. As these values are computed, compare with the largest pair already in storage, and if necessary reassign L_p and (A_1,A_2). Repeat Step 2 until i = INIT_POINT.

INIT_POINT prevents the algorithm from scanning entirely around the polygon. This would be wasteful of time because each antipodal pair would be produced twice. An analogy can be drawn between the use of INIT_POINT and swinging the line L through 180 degrees in the Shamos algorithm. Also, when two edges of the hull are parallel we must backtrack to produce an extra antipodal pair. This corresponds to the special case in the Shamos algorithm when L passes two edges simultaneously. It can easily be proved, using theorem 3, that algorithm LARGEST_ANTIPODAL_PAIR runs in O(N) time.

As in the case of the MER, we tested FORTRAN versions of both the Shamos antipodal pair finder (DIAM) and the algorithm LARGEST_ANTIPODAL_PAIR (LPAIR). Uniform random variates were generated for two distributions: uniform on the boundary of an ellipse and uniform on the boundary of a circle. Again, for each sample size, all points remained on the hull. Five realizations of 100 runs were made; average times and standard deviations (in seconds) appear in Tables 2 and 3 below.

Table 2
(points generated on boundary of an ellipse)

N	LPAIR	DIAM
125	.466 (.0089)	.698 (.0048)
250	.926 (.0152)	1.432 (.0164)
500	1.832 (.0130)	2.886 (.0207)
1000	3.644 (.0462)	5.810 (.0678)
2000	6.990 (.0914)	11.762 (.1180)

Table 3
(points generated on boundary of a circle)

N	LPAIR	DIAM
125	.470 (.0100)	.718 (.0084)
250	.908 (.0045)	1.430 (.0354)
500	1.802 (.0259)	2.854 (.0252)
1000	3.524 (.0528)	5.700 (.0791)
2000	6.890 (.0557)	11.304 (.0691)

Clearly, both algorithms exhibit O(N) behavior. LARGEST_ANTIPODAL_PAIR is slightly faster than the Shamos algorithm DIAM. One reason for this could be that we used trigonometric functions to compute the angular displacement of each edge of the convex polygon in DIAM. A gain in speed might be accomplished by devising a scheme to avoid the use of these functions. Machine architectures can also affect the comparative speed of algorithms which are heavily compute bound [12].

5. *CONCLUDING REMARKS*

We have demonstrated the usefulness of the HIGHPOINT strategy. The technique is viable in two dimensions because the vertices of the convex hull are in polar order about an interior point. Unfortunately, the technique is not extendible to 3 coordinate space. This is not uncommon in computational geometry where often a new technique performs spectacularly in 2 dimensions, but not in 3 dimensions. Hence, the problem of the minimum volume encasing box of a set of N points cannot be solved by use of the HIGHPOINT strategy. Nevertheless, the technique is obviously a viable one for two dimensional covering problems.

REFERENCES

[1] Shamos, M.I. Computational Geometry, Ph.D. Thesis, Dept. of Comp. Science, Yale University, 1978.

[2] Freeman, H. and Shapira, R. "Determining the minimum area encasing rectangle for an arbitrary closed curve." CACM 18, no. 7, 1975, pp. 409-413.

[3] Graham, R.L. "An efficient algorithm for determining the convex hull of a finite planar set." Info. Proc. Lett. 1, no. 1, 1972, pp. 132-133.

[4] Bentley, J.L. and Shamos, M.I. "Divide and conquer for linear expected time." Info. Proc. Lett. 7, no. 2, 1978, pp. 87-91.

[5] Akl, S.G. and Toussaint, G.T. "A fast convex hull algorithm." Info. Proc. Lett. 7, no. 5, 1978, pp. 219-222.

[6] McCallum, D and Avis, D. "A linear algorithm for finding the convex hull of a simple polygon." Info. Proc. Lett. 9, no. 5, 1979, pp. 201-206.

[7] Anderson, K.R. "A reevaluation of an efficient algorithm for determining the convex hull of a finite planar set." Info. Proc. Lett. 7, no. 1, 1978, pp. 53-55.

[8] Noga, M.T. Convex Hull Algorithms, Masters Thesis, Dept. of Computer Science, Virginia Tech, 1981.

[9] Shamos, M.I. "Geometric Complexity." Proc. Seventh Annual ACM Symp. on Theory of Computing, May 1975, pp. 224-233.

[10] Hocking, J.G. and Young, G.S. Topology, Addison-Wesley, 1961.

[11] Yaglom, I.M. and Boltyanskii, V.G. Convex Figures, Holt, Rinehart, and Winston, 1961.

[12] van der Nat, M. "A fast sorting algorithm, a hybrid of distributive and merge sorting." Info. Proc. Lett. 10, no. 3, 1980, pp. 163-167.

[13] Sklansky, J. "Measuring concavity on a rectangular mosaic." IEEE Trans. on Computers, vol. C-21, no. 12, Dec. 1972, pp. 1355-1362.

[14] Bykat, A. "Convex hull of a finite set of points in two dimensions." Info. Proc. Lett. 7, no. 6, 1978, pp. 297-298.