A New Convex Hull Algorithm for Planar Sets

WILLIAM F. EDDY
Carnegie-Mellon University

A new algorithm, CONVEX, that determines which points of a planar set are vertices of the convex hull of the set is presented. It is shown that CONVEX operates in a fashion similar to the sorting algorithm QUICKERSORT. Evidence is given which indicates that in some situations CONVEX is preferable to earlier algorithms. A Fortran implementation, intended to minimize execution time, is presented and an alternative, which minimizes storage requirements, is discussed.

Key Words and Phrases: convex hull, QUICKERSORT, partitioning, sorting
CR Categories: 5.30, 5.31
The Algorithm: CONVEX, A New Convex Hull Algorithm for Planar Sets. *ACM Trans. Math. Software 3*, 4 (Dec. 1977), 411–412.

INTRODUCTION

The convex hull of a planar set is the minimum-area convex polygon containing the planar set. A convex polygon is clearly determined by its vertices. Graham [1] suggests an algorithm for determining which points of a planar set are vertices of its convex hull. Because his algorithm requires sorting the points, if there are N points then at least $O(N \log N)$ operations are needed to determine the vertices. Recently, Preparata and Hong [3, 4] have shown that there exist sets of points for which every algorithm requires at least $O(N \log N)$ operations to determine the vertices of the convex hull. Jarvis [2] gives an algorithm which requires $O(N \cdot C)$ operations, where C is the number of vertices. For some configurations of the points in the plane (those with small values of C) the algorithm given by Jarvis will be faster than the algorithm of Graham; for other configurations it may be slower. An adaptive algorithm, CONVEX, is presented here which never requires more than $O(N \cdot C)$ operations to determine the vertices of the convex hull and may require substantially fewer. However, CONVEX may require more operations than Graham's algorithm for some configurations of points. Evidence is presented which suggests that in applications CONVEX is preferable to the "sorting" algorithms [1, 3, 4] and to Jarvis's algorithm [2].

METHOD

Operationally, this algorithm is analogous to the sorting algorithm QUICKERSORT [5]. At each step QUICKERSORT partitions the input array with respect to a

This research was supported in part by the National Science Foundation under Grant DCR75-08374.
Author's address: Department of Statistics, Carnegie-Mellon University, Schenley Park, Pittsburgh, PA 15213.

particular point and yields two output arrays, one containing those elements of the input array which are smaller than the point and the other containing those elements which are larger. QUICKERSORT continues to partition the subarrays until subarrays of length 1 are obtained. CONVEX partitions the input array with respect to a particular line, yielding two arrays of points. For convenience the two sides of the partitioning line will be referred to as "above" and "below." The key to the speed of CONVEX is that not every subarray need be partitioned. Consider the points inside a triangle with vertices which are vertices of the convex hull; these points cannot be vertices of the convex hull. It is possible to eliminate these points from further consideration by choosing the partitioning lines to be sides of such a triangle.

TREE REPRESENTATION

This algorithm may be interpreted as a preorder traversal of a particular binary tree. The root of the tree represents the original set of points. The left son of each node of the tree represents the subset of points above a partitioning line and the right son represents the subset below. The leaves of the tree represent either null subsets or subsets inside a triangle whose vertices are vertices of the convex hull. On the left half of the tree left sons are visited first; on the right half of the tree right sons are visited first.

PARTITIONING

At each step the subset of points represented by the current node of the tree is partitioned by algorithm SPLIT. As input, SPLIT requires a line joining two vertices of the convex hull and a set of points distinct from these vertices. It produces as output (1) the subset A of points above the line, (2) the point farthest above the line if $A \neq \emptyset$ (the empty set), (3) the subset B of points below the line, and (4) the point farthest below the line if $B \neq \emptyset$. If the equation of the line is $Z(X, Y) = a + bX + Y = 0$, then $Z(X_0, Y_0)$ determines for a point (X_0, Y_0) its position relative to the line. Points which lie on the line are not vertices of the convex hull and may be neglected in succeeding computations. The initial partitioning line joins any two vertices; for simplicity they are chosen to be the points with minimum and maximum X-coordinates.

AN EXAMPLE

In order to clarify the operation of this algorithm, the following example is presented. Figure 1 displays a planar set of points, the input to the algorithm. Only those points which are important to the operation of the algorithm, the output vertices of the convex hull, have been labeled. Table I presents in order the input and output of successive partitioning steps with a legend describing the different subsets of points. Figure 2 is the associated binary tree.

SPEED OF COMPUTATION

The number of operations required by this algorithm is of the same order as the total number of points at all the nodes of the binary tree; this is equal to the total number of points passed to SPLIT. It is possible that the tree is of height N and

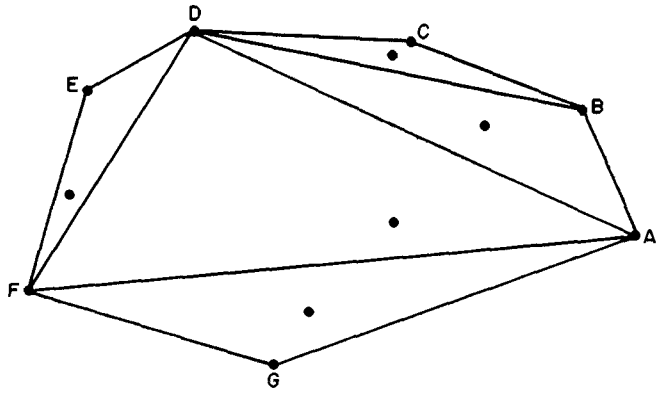

Fig. 1. Sample input to CONVEX with vertices labeled and partitioning lines drawn

Table I. Successive Calls to SPLIT for Sample Points of Figure 1
(A dash indicates that the information is not relevant to the operation of the algorithm.)

Input		Output			
		Above		Below	
Subset	Line	Subset	Extreme	Subset	Extreme
1	AF	2	D	10	G
2	AD	3	B	7	—
3	AB	∅		4	—
4	BD	5	C	α	—
5	BC	∅		6	—
6	CD	∅		β	—
7	DF	8	E	γ	—
8	DE	∅		9	—
9	EF	∅		δ	—
10	FG	11	—	∅	
11	GA	ϵ	—	∅	

Legend

Subset number	Points	Subset number	Points
1	all	9	above DF, below DE
2	above AF	10	below FA
3	above AD	11	below FA, above FG
4	above AD, below AB	α	inside Δ ABD
5	above BD	β	inside Δ BCD
6	above BD, below BC	γ	inside Δ ADF
7	above AF, below AD	δ	inside Δ DEF
8	above DF	ϵ	inside Δ AFG

hence there are a total of $N(N + 1)/2$ points. Consider, for example, the set of points which lie on the boundary of a circle at angles 0, π, $\pi/2$, $\pi/4$, $\pi/8$, ... It is also possible that the tree is of height 1 and hence there are only N points, if all the points lie on a line.

Each time a SPLIT is performed either (1) a new vertex of the convex hull is

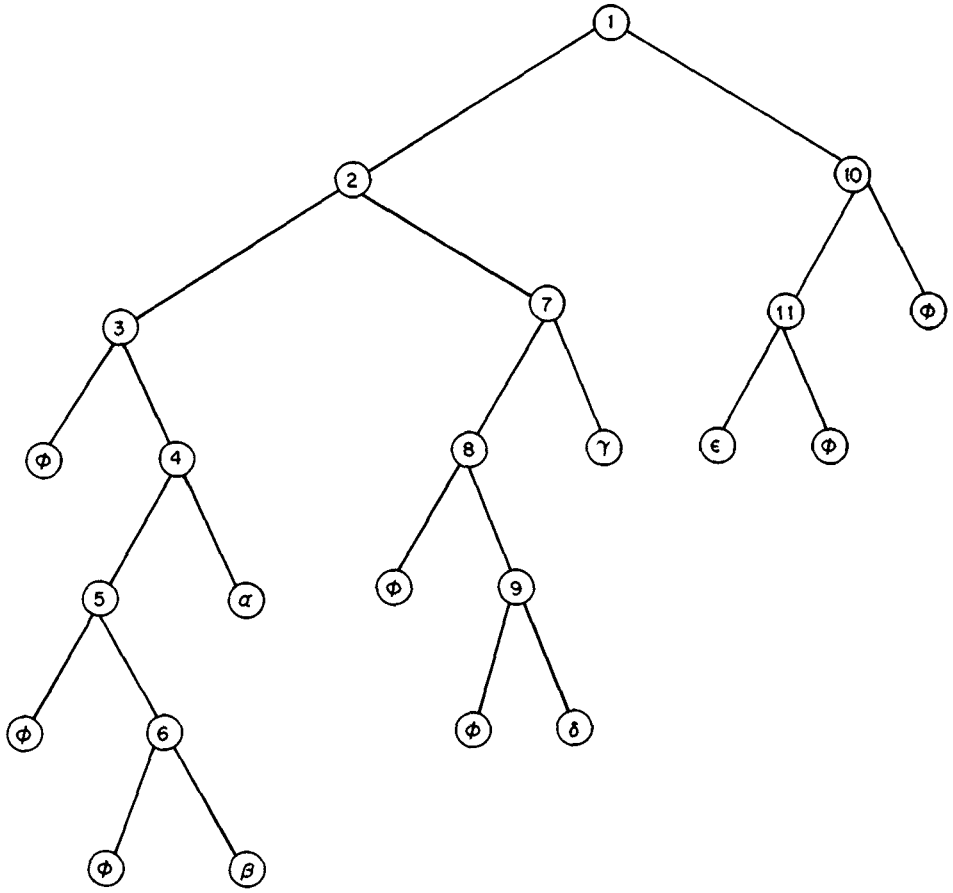

Fig. 2. Binary tree of successive calls to SPLIT in Table I

found or (2) it is determined that the partitioning line is an edge of the convex hull. Since the number of edges equals the number of vertices, say C, the total number of operations required is certainly less than or equal to $O(N \cdot C)$ but this *upper* bound may be extremely crude. It should be noted that if a point is not on the convex hull then it must be passed to SPLIT at least three times before it may be eliminated from consideration. Thus a practical *lower* bound for the total number of points passed to SPLIT is $3N + O(1)$.

Since the time required by this algorithm depends on the particular configuration of points, it seems natural to consider samples from different probability distributions. For points sampled from a circular Gaussian distribution, it is known [5] that for sufficiently large N the $E(C)$ is $O(\sqrt{(\log N)})$. In this case the crude bound for the expected number of operations is $O(N\sqrt{(\log N)})$. For a sample from a uniform distribution on the interior of a circle the $E(C)$ is $O(N^{1/3})$ for large N [5]. Here the crude bound becomes $O(N^{4/3})$.

Pseudorandom samples were generated for sample sizes 10, 32, 100, 316, and 1000 for five distributions: (1) uniform on the boundary of a circle, (2) uniform on

Table II. Average Number of Operations for Nine Random Samples from Each of Five Distributions

(Numbers in parentheses are estimated standard deviations of the averages. Bottom number in each cell is average number of vertices. All numbers rounded to integers.)

Distribution / N	Uniform on boundary of circle	Uniform on interior of circle	Uniform on interior of square	Circular Gaussian	Circular exponential
10	23 (0)	21 (1)	21 (0)	21 (0)	21 (0)
	10	5	6	5	5
32	145 (0)	98 (2)	99 (2)	93 (0)	90 (0)
	32	9	9	8	7
100	707 (5)	343 (4)	330 (6)	305 (2)	299 (2)
	100	15	12	9	8
316	2998 (7)	1153 (5)	1141 (24)	975 (7)	966 (9)
	314*	22	15	11	9
1000	11913 (19)	3691 (10)	3616 (68)	3094 (18)	3021 (8)
	958*	33	19	12	10

* These numbers artificially reduced due to truncation error; see "Implementation."

the interior of a circle, (3) uniform on the interior of a square, (4) circular Gaussian, (5) circular (double) exponential. The fifth distribution has a density which is proportional to exp $(\sqrt{(X^2 + Y^2)})$; the conditional distribution on any ray from the origin is standard exponential. The total number of points passed to SPLIT and the number of vertices of the convex hull were recorded. Average values for 9 samples are given in Table II.

The bottom number in each cell in Table II is $\bar{C}$, the average value of C. Notice that $N \cdot \bar{C}$ grows more quickly, for each distribution, than the average number of points passed to SPLIT (the upper number in each cell). Hence this algorithm is an improvement on Jarvis's algorithm, at least for these situations. Notice also that, except for the extreme case represented by the first column, $N \log N$ grows more quickly than the average number of points passed to SPLIT. Hence this algorithm is an improvement on the "sorting" algorithms for these situations.

IMPLEMENTATION

The application for which this algorithm was originally intended requires repeated use of the program on different subsets of a sample of only a few hundred points. This suggested the use of indirect addressing and a liberal use of storage in an effort to increase the speed. A linked list is used to refer to vertices of the convex hull allowing insertions to be made at fixed cost no matter how many vertices have been found. The following loop in the calling program will allow the transfer of successive vertices (in counterclockwise order) from the array X to the array Y.

```
      K = IL(1)
      DO 1 I = 1, NH
      J = IH(K)
      Y(1, I) = X(1, J)
      Y(2, I) = X(2, J)
1     K = IL(K)
```

A different implementation of the algorithm is possible. To minimize the use of storage, data points could be rearranged in place in a manner similar to QUICKERSORT. In fact, the vertices of the convex hull could be moved to the initial ele-

ments of the data array. Then the only additional storage required would be for a pushdown stack to store the pointers to subarrays not yet partitioned. For this alternative implementation right sons (in the binary tree) would be visited first on the left half of the tree and left sons first on the right half. There would probably be considerable loss in speed of execution. The program presented here required less than 11 seconds of CPU time on an IBM 370/158 (in BC mode) to produce the first column of numbers in Table II. This time includes generating the samples, printing the results, and all related calculations for what is admittedly an extreme case.

The only difficulty which has arisen while using this program can be noted in the last two cells of the first column in Table II. When the convex hull has a large number of vertices, adjacent sides are nearly parallel and truncation (roundoff) error can significantly affect the results. Increasing the precision of the arithmetic alleviates this problem.

There are several configurations of points which are handled by the program CONVEX in a special way. Whenever the input contains only one or two points they are the vertices of the convex hull (unless they are identical in the case when there are two). Whenever all the points lie on a vertical line, this is detected and the two extremes are returned without any calls to SPLIT. Vertical boundaries are given special treatment: (1) When the two points which determine the initial partition are chosen, care is taken to assure that they are vertices and not just boundary points. (2) When partitioning, SPLIT uses the information whether the set being partitioned is above or below the initial partitioning line to determine its output.

There is no feature inherent in the basic idea of this algorithm which prevents implementation for dimensions higher than 2. However, because the vertices of the convex hull in higher dimensions cannot be ordered, the programming details will be considerably more complicated. Of course in higher dimensions the gains to be made by this procedure are smaller; in fact, in $N - 1$ dimensions all N points are vertices of the convex hull if they are linearly independent.

ACKNOWLEDGMENT

I would like to thank J.A. Hartigan and B.H. Margolin for helpful conversations. C.L. Lawson pointed out several errors in the way the original program handled special cases, suggested several test configurations, and made other valuable remarks.

REFERENCES

1. Graham, R.L. An efficient algorithm for determining the convex hull of a planar set. *Inform. Proc. Letters 1* (1972), 132–133.
2. Jarvis, R.A. On the identification of the convex hull of a finite set of points in the plane. *Inform. Proc. Letters 2* (1973), 18–21.
3. Preparata, F.P., and Hong, S.J. Convex hulls of finite planar and spatial sets of points. Report R-682, Coordinated Science Lab., U. of Illinois, Urbana, Ill., April 1975.
4. Preparata, F.P., and Hong, S.J. Convex hulls of planar and spatial sets of points. Abstract 726-68-1, *Notices AMS 22* (1975), A-596.
5. Raynaud, H. Sur l'enveloppe convexe des nuages de points aleatoires dans R^n.I. *J. Appl. Prob. 7* (1970), 35–48.
6. Scowen, R.S. Algorithm 271: Quickersort. *Comm. ACM 8*, 11 (1965), 669–670.

Received September 1975; revised June 1976 and January 1977