A LINEAR ALGORITHM FOR FINDING THE CONVEX HULL OF A SIMPLE POLYGON *

Duncan McCALLUM and David AVIS
School of Computer Science, McGill University, Montreal, Quebec H3A 2K6, Canada

Received 12 June 1979; revised version received 26 September 1979

Convex hull, simple polygon, analysis of algorithms

1. Introduction

The problem of determining the convex hull of a set of n points in the plane has recently received a good deal of attention. Several algorithms for the general problem with worst case complexity O(n log n) have appeared (e.g., [3,4,6]). The special case where the points form the vertices of a simple polygon has long been considered easier. Indeed, Sklansky [5] has proposed an O(n) algorithm, but a recently published counter example of Bykat [2] shows that the algorithm can sometimes fail. A slightly different counterexample can be constructed for a similar algorithm of Shamos [4]. In this note we present and prove the validity of a new linear time algorithm for this problem.

2. Definitions, notation and preliminary results

In this note we will be concerned with polygons in the euclidean plane. A *point* is represented by its x and y coordinates and a *polygon* is represented by a list of points in the order that are encountered as the boundary is traversed. We shall refer to these points as *turn points.* We will not always distinguish the polygon from its boundary. A *simple polygon* is a polygon whose boundary is a simple closed curve.

* This research was supported by the National Research Council of Canada under research grant NRC A3013.

The convex hull of a set S of points is denoted Hull(S). An *extreme point* of a polygon P is a point of P that lies on the boundary of Hull(P). It follows from the Jordan Curve theorem that the extreme points of a simple polygon are in sorted order: that is, if $p_{i_1}, p_{i_2}, ..., p_{i_k}$ are the extreme points with $i_1 < i_2 < \cdots < i_k$, then $p_{i_1} p_{i_2} \cdots p_{i_k}$ represents a convex polygon. A simple polygon is *clockwise oriented* if its interior lies to the right as the polygon is traversed. As one can easily determine, and if necessary reverse, the orientation of a simple polygon in linear time, we assume that every simple polygon is represented with a clockwise orientation. Further, in order to simplify both the algorithm and its proof of correctness, we shall assume that no three turn points of the polygon are collinear, and also that there be unique turn points of the polygon with minimum and maximum x-coordinates. Slight modifications to the algorithm render these assumptions unnecessary.

Given three points $p_i = (x_i, y_i)$, $p_j = (x_j, y_j)$ and $p_k = (x_k, y_k)$, let

$$s = x_k(y_i - y_j) + y_k(x_j - x_i) + y_j x_i - y_i x_j .$$

We say that p_k is *left* of (respectively *on, right* of) the directed line $p_i p_j$ if s is positive (respectively zero, negative). When the points are distinct this agrees with intuition. We occasionally allow the possibility that $p_i = p_j$ in which case s is zero and p_k is 'on' the line.

Let $P = p_1 p_2 \cdots p_{n-1} p_n p_{n+1} \cdots p_m$ be a simple polygon such that p_1 and p_n are the points with minimum and maximum x-coordinates respectively.

Define:

$$LH_n = \begin{cases} \mathrm{Hull}(p_1, p_n, \{p_j \mid k \leqslant j < n;\ p_j \text{ is left of } p_1p_n\}), & 1 \leqslant k < n, \\ \mathrm{Hull}(p_1, p_n, \{p_j \mid k \leqslant j \leqslant m;\ p_j \text{ is left of } p_np_1\}), & n \leqslant k \leqslant m. \end{cases}$$

Finally let Path(P, p_i, p_j) be the path in P from p_i to p_j which follows the orientation of P.

It is easily seen that p_1 and p_n split Hull(P) into two paths, Path(Hull(P), p_1, p_n) and Path(Hull(P), p_n, p_1). It follows from convexity and the above-mentioned sorted order property of simple polygons that

$$\mathrm{Path}(\mathrm{Hull}(P), p_1, p_n) \subseteq LH_1 ,$$

$$\mathrm{Path}(\mathrm{Hull}(P), p_n, p_1) \subseteq LH_n .$$

Certainly, $LH_1 \cup LH_n \subseteq \mathrm{Hull}(P)$. Thus we must have $LH_1 \cup LH_n = \mathrm{Hull}(P)$. It suffices, then, to consider just the problem of finding the *left hull* LH_1 of the polygonal path $p_1p_2 \cdots p_n$. In the next section we present an algorithm for solving this problem.

3. The algorithm

Input: n, the number of points, and an array $P = p_1, p_2, ..., p_n$ representing a simple path such that p_1 is the point with minimum x-coordinate, p_n is the point with maximum x-coordinate and no three points are collinear.

Output: A linked list,

$$1 \to \mathrm{link}(1) \to \mathrm{link}(\mathrm{link}(1)) \to \cdots n ,$$

stored in the array link containing the indices of the extreme points of LH_1 in clockwise order.

Method: The algorithm scans the points in reverse order from p_n to p_1, updating two stacks that are stored in the array link. After scanning p_k, the stacks are as follows:

Stack A (accept) contains the indices in P of those points so far scanned which have been tentatively accepted as extreme points. The bottom index in A is always n. Elements in A will always be in decreasing order from bottom to top. Upon termination of the algorithm, A will contain the extreme points of LH_1. The variable ja points to the top element in the stack. The variable ia points to the next element in the stack, if there is one; otherwise ia = ja = n.

Stack T (temporary) contains index 1 at the bottom and the indices of those points so far scanned, if any, which have been rejected as extreme points of LH_1, but which are extreme points of LH_k. Except for the bottom index, elements of T will be in decreasing order from the bottom to the top. Upon termination of the algorithm, T will contain just the index 1. The variable jt will point to the top element in T.

Before formally stating the algorithm, we give an informal description of its operation. For the remainder of the section we will refer to the regions diagrammed in Fig. 1. The points a_1, ..., a_5 are stored in stack A, and the points t_1, t_2, t_3 are stored in stack T. The regions R_1, ..., R_5 are defined formally in Section 4. The key point is that at each major iteration of the algorithm, stacks A and T combined contain the vertices of the convex hull of the points p_1, p_n and the turn points of P that have already been examined. As stated above, the turnpoints of P are examined in reverse order from p_n to p_1. Consider the examination of p_k. If it lies in R_1, it may be immediately rejected as an extreme point of P. If p_k lies in R_2, then it must be added to stack A since it is tentatively an extreme point of P. Stack T must be backtracked at this point, since the inclusion of p_k may have rendered some points in T interior to LH_k. Any such points must lie on the top of stack T and are discarded. If p_k lies in R_3, the situation is similar, except that both stacks A and T must be backtracked in order to check for the top elements becoming interior points of LH_k. Consider the case where p_k lies in R_4. Since P is simple, it can be shown that the top element p_{ja} stored in A can no longer be an extreme point of P. It is, however, an extreme point of LH_k, and so it is removed from A and placed onto T. The stack A is backtracked since the inclusion of p_k may render other points interior to LH_k. Again only the top elements need be considered. Note that any turnpoints of P stored on stack A except the previous top element p_{ja} are interior to both LH_k and P. They are not, therefore, transferred to stack T, but are discarded. The only remaining possibility is that p_k lies in R_5. This case may, however, be excluded by an application of the Jordan Curve Theorem. A proof of this fact forms an important part of the verification

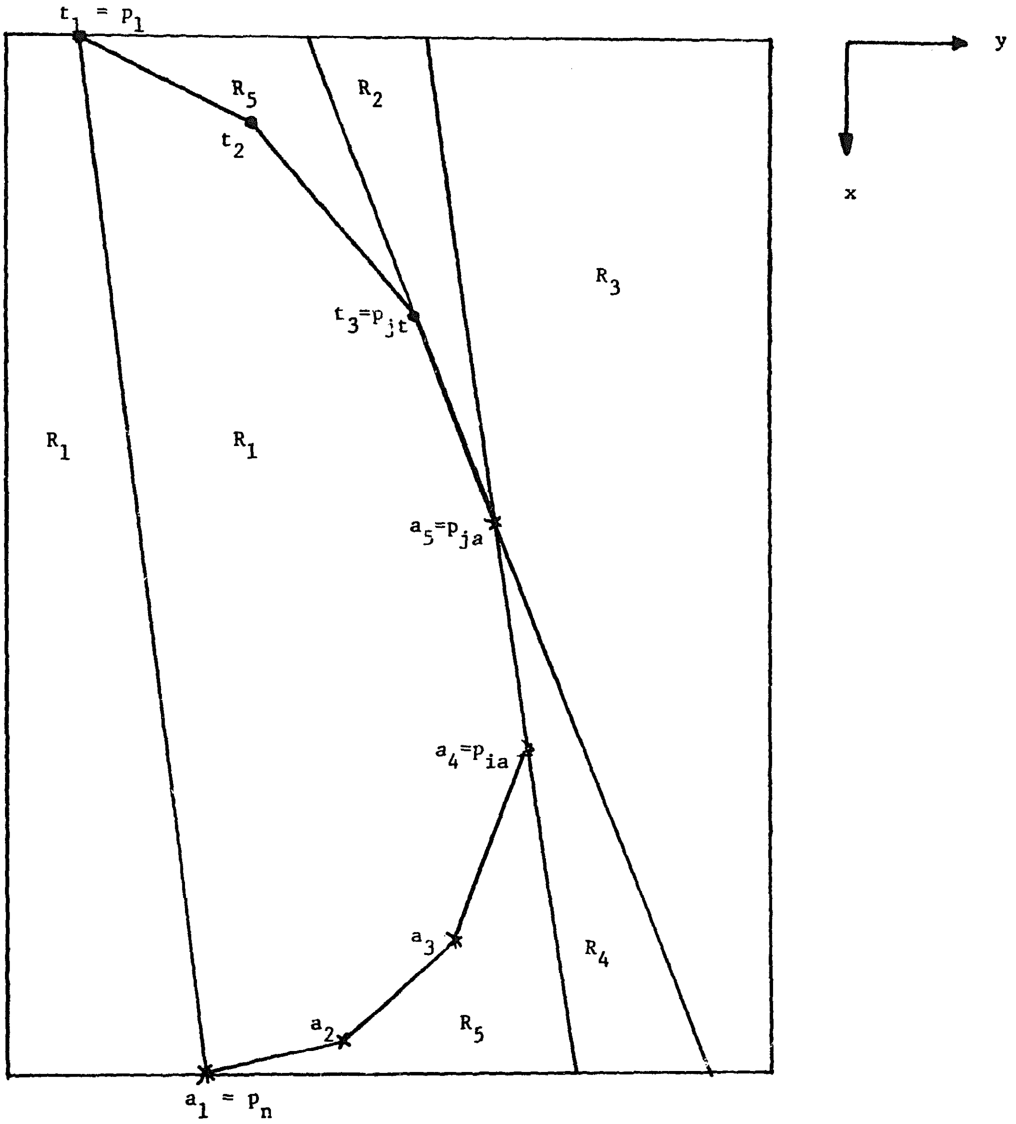

Fig. 1.

of the correctness of the algorithm that is given in Section 4.

The fact that the algorithm has a linear running time follows essentially from the fact that a point is either placed onto stack A, onto stack A and stack T, or is discarded. When a stack is backtracked, the points removed always lie on the top and may never be replaced onto the same stack. The major difference between our algorithm and the earlier algorithms is the maintenance of the stack T. Although the points on this stack can never lie on the convex hull of P (except point p_1), they are essential for rejecting points interior to R_1. The retention of these *interior* points as tentative extreme points leads to the pitfalls discovered by Bykat [2]. We now give a formal statement of the algorithm.

The comments in the statement of the algorithm refer to the regions shown in Fig. 1. These regions are established each time step HH3 is executed; they are defined formally in Section 4. The backtracking steps are to maintain convexity of the paths represented by the indices stored in the respective stacks.

```
Procedure halfhull(n,p,link)
      begin
HH1:    jt←1;
        link(1)←1;
        link(n)←n;
        k←n–1;
        while pk right of p1pn do k←k–1;
        link(k)←n;
        if k=1 return;
        ja←k;
        ia←n;
HH2:    k←k–1;
HH3:    if k=1 then
          begin
            link(k)←ja;
            return;
          end;
        if pk right of piapja then
          begin
            comment pk in R4, push ja onto T;
            if pk left of pjapjt then
              begin
                link(ja)←jt;
                jt←ja;
              end;
            comment pk in R3, backtrack T;
            else while pk right of pjtplink(jt)
              do jt←link(jt);
            comment backtrack A;
            while pk right of piaplink(ia)
              do ia←link(ia);
            comment push k onto A;
            link(k)←ia;
            ja←k;
            goto HH2;
          end;
        if pk left of pjapjt then go to HH2;
        comment if pk in R1, reject;
                  otherwise pk in R2, push k onto A;
        link(k)←ja;
        ia←ja;
        ja←k;
        comment backtrack T;
        while pk right of pjtplink(jt)
          do jt←link(jt);
        goto HH2;
      end halfhull;
```

4. Analysis of the algorithm

Before analyzing the algorithm we introduce some further notation. Suppose that at a general step of the algorithm the stacks A and T contain respectively α and τ indices. Let the corresponding turn points of p be denoted $a_1, ..., a_\alpha$ and $t_1, ..., t_\tau$ respectively, where $a_1 = p_n$ and $t_1 = p_1$ are the bottom elements of the stacks. An important part of the proof will be to show that every time the algorithm begins execution of step HH3, LH_{k+1} is the convex polygon represented by $t_1 \cdots t_\tau a_\alpha \cdots a_1$.

Define

$$R = \{(x, y) | x_1 \leqslant x \leqslant x_n, y_{min} \leqslant y \leqslant y_{max}\},$$

where y_{min} and y_{max} are the minimum and maximum y coordinates. In the sequel, sets will be considered open or closed relative to R.

Prior to each execution of HH3, we will consider the following five open regions of R, which depend on the contents of stacks A and T at that moment (see Fig. 1):

$$R_1 = \text{Interior}(\{p \in R | \text{p is right of } p_1p_n\} \cup \text{Hull}(a_1, ..., a_\alpha, t_\tau, ..., t_1)),$$

$$R_2 = \{p \in R | \text{p is right of } p_{ja}p_{jt} \text{ and left of } p_{ia}p_{ja}\},$$

$$R_3 = \{p \in R | \text{p is right of } p_{ja}p_{jt} \text{ and right of } p_{ia}p_{ja}\},$$

$$R_4 = \{p \in R | \text{p is left of } p_{ja}p_{jt} \text{ and right of } p_{ia}p_{ja}\},$$

$$R_5 = \text{Interior}(R - \{R_1 \cup R_2 \cup R_3 \cup R_4\}).$$

These regions are illustrated in Fig. 1. Note that since A contains at least two indices immediately prior to execution of HH3, R_1, R_2 and R_4 will be non-empty regions. R_3 will be empty if and only if p_k has maximum y-coordinate. R_5 will be empty if and only if T contains exactly one index and A contains exactly two indices. The main part of the proof is contained in the following lemma:

Lemma. Every time HH3 is executed, the following conditions hold:

(a) LH_{k+1} is the convex polygon $t_1 \cdots t_\tau a_\alpha \cdots a_1$,

(b) if $\tau \geqslant 2$, $t_2, ..., t$ are interior points of LH_1,

(c) a_α is the most recently scanned extreme point of LH_{k+1}.

Proof. By induction on the number of times that HH3 is executed. Initially $\tau = 1$ and $\alpha = 2$ and conditions (a)–(c) are easily verified. We assume inductively that the conditions are satisfied immediately prior to the scanning of point p_k. Note that initially $a_\alpha = p_{ja}$, $a_{\alpha-1} = p_{ia}$, $t_\tau = p_{jt}$. By our assumption of non collinearity, p_k must lie in $R_1 \cup R_2 \cup R_3 \cup R_4 \cup R_5$. We consider each case separately:

(i) $p_k \in R_1$: The algorithm leaves stacks unchanged and $LH_k = LH_{k+1}$.

(ii) $p_k \in R_2$: The algorithm pushes p_k onto stack A and backtracks stack T. Suppose $t_{\tau'}$ is at the top of T after backtracking. Then it is easily verified that $\{t_{\tau'+1}, ..., t_\tau\} \subseteq \text{Hull}(a_\alpha, p_k, t_{\tau'})$. Thus the polygon $t_1 \cdots t_{\tau'} p_k a_\alpha \cdots a_1$ contains LH_{k+1} and p_k, is convex by construction, and is the smallest such polygon; hence it is LH_k.

(iii) $p_k \in R_3$: The algorithm backtracks A and T, then pushes p_k onto stack A. Suppose that after backtracking, $a_{\alpha'}$ is at the top of A and $t_{\tau'}$ is at the top of T. Then $\{a_{\alpha'+1} \cdots a_\alpha, t_{\tau'+1}, ..., t_\tau\} \subseteq \text{Hull}(a_{\alpha'}, p_k, t_{\tau'})$ so that, as in (ii), LH_k is the convex polygon $t_1 \cdots t_{\tau'} p_k a_{\alpha'} \cdots a_1$.

(iv) $p_k \in R_4$: The algorithm pushes a_α onto stack T, backtracks A to $a_{\alpha'}$, then pushes p_k onto stack A. As in (ii) and (iii), LH_k is the convex polygon $t_1 \cdots t_\tau a_\alpha p_k a_{\alpha'} \cdots a_1$. It remains to show that a_α is an interior point of LH_1. This will follow if we can show that $R_3 \cap \text{Path}(P, p_1, p_k) \neq \emptyset$. For if a point p is contained in this intersection, then $a_\alpha \in \text{Hull}(p, p_k, t_\tau)$. On the other hand, $\text{Path}(P, a_\alpha, p_n) \subset \text{closure}(R_1)$ by induction hypothesis (a), and separates $R - R_3$ into two components with p_k and p_1 in opposite components. If $\text{Path}(P, p_1, p_k) \subset R - R_3$, then it must intersect $\text{Path}(P, a_\alpha, p_n)$ which is impossible. Thus we have shown that $R_3 \cap \text{Path}(P, p_1, p_k) \neq \emptyset$ and a_k is interior to LH_1.

(v) $p_k \in R_5$: This case is impossible. Suppose that $\{a_{\alpha-1}, t_\tau\} = \{p_i, p_j\}$ and that $i < j$. Then $\text{Path}(P, p_i, p_j)$ separates closure $(R_1 \cup R_5)$ into two components with a_α and R_5 in separate components. For this path cannot contain a_α by inductive hypothesis (c) and does not intersect R_5 by inductive hypothesis (a). Now $\text{Path}(P, p_k, a_\alpha) \subseteq$ closure $(R_1 \cup R_5)$ because by inductive hypothesis (a) $\text{Path}(P, p_{k+1}, a_\alpha) \subseteq$ closure (R_1) and the line $p_k p_{k+1}$ lies in closure $(R_1 \cup R_5)$ since this region is convex. Therefore $\text{Path}(P, p_k, a_\alpha)$ must cross $\text{Path}(P, p_i, p_j)$ which is impossible.

Cases (i) to (v) are exhaustive and so the lemma follows by induction.

Theorem. Procedure halfhull finds the left hull of p in linear time.

Proof. The validity of the algorithm follows from the lemma applied when $k = 2$, noting that $LH_2 = LH_1$. The main step, HH3, of the algorithm is executed at most $n - 2$ times. A given turn point may be placed into neither stack, into stack A once, or into both stacks A and T once each. Once discarded, a point is never reconsidered and so the algorithm runs in linear time.

5. Conclusion

We have exhibited an O(n) algorithm for finding the convex hull of a simple polygon. It is clear, however, that the algorithm will work on a much larger class of polygons. We are unable to characterize this class, although it is easily shown that the extreme points of such polygons must appear in sorted order. Under the linear decision tree model, Avis [1] has found an $\Omega(n \log n)$ lower bound for the general problem of finding the convex hull of a set of points in the plane. A similar result for the more powerful quadratic decision tree model has been recently announced by Yao [7]. Thus it would be of interest to characterize the class of polygons for which an O(n) algorithm exists.

Acknowledgment

The authors gratefully acknowledge the help and encouragement of Godfried Toussaint during the course of this research.

References

[1] D. Avis, On the complexity of finding the convex hull of a set of points, Technical Report No. SOCS 79.2, McGill University (1979).

[2] A. Bykat, Convex hull of a finite set of points in two

dimensions, Information Processing Lett. 7 (1978) 296–298.

[3] R. Graham, An efficient algorithm for determining the convex hull of a planar set, Information Processing Lett. 1 (1972) 132–133.

[4] M. Shamos, Problems in computational geometry, Carnegie Mellon University (1975) revised (1977).

[5] J. Sklansky, Measuring concavity on a rectangular mosaic, IEEE Trans. Comput. 21 (1972) 1355–1364.

[6] G. Toussaint, S. Akl and L. Devroye, Efficient convex hull algorithms for points in two and more dimensions, Technical Report No. 78.5, McGill University (1978).

[7] A. Yao, private communication (1979).