The Ultimate Planar Convex Hull Algorithm?

David G. Kirkpatrick*
and Raimund Seidel+

83-577
October 1983

*Department of Computer Science
University of British Columbia
Vancouver, B.C. Canada V6T 1W5

+Department of Computer Science
Cornell University
Ithaca, New York 14853

THE ULTIMATE PLANAR CONVEX HULL ALGORITHM ?

by

David G. Kirkpatrick[1] and *Raimund Seidel*[2]

[1]Department of Computer Science
University of British Columbia
Vancouver, B.C. V6T 1W5 CANADA

[2]Department of Computer Science
Cornell University
Ithaca, N.Y. 14853 USA

Abstract:

We present a new planar convex hull algorithm with worst case time complexity $O(n \log H)$ where n is the size of the input set and H is the size of the output set, i.e. the number of vertices found to be on the hull. We also show that this algorithm is asymptotically worst case optimal on a rather realistic model of computation even if the complexity of the problem is measured in terms of input as well as output size. The algorithm relies on a variation of the divide-and-conquer paradigm which we call the "marriage-before-conquest" principle and which appears to be interesting in its own right.

Keywords: computational geometry, convex hull, divide-and-conquer, lower bounds

–

The Ultimate Planar Convex Hull Algorithm?

David G. Kirkpatrick*
and Raimund Seidel+

83-577
October 1983

*Department of Computer Science
University of British Columbia
Vancouver, B.C. Canada V6T 1W5

+Department of Computer Science
Cornell University
Ithaca, New York 14853

I. Introduction

The convex hull of a finite point set S in the plane is the smallest convex polygon containing the set. The vertices (corners) of this polygon must be points of S. Thus in order to compute the convex hull of a set S it is necessary to find those points of S which are vertices of the hull. For the purposes of constructing upper bounds we define the convex hull problem, as the problem of constructing the ordered sequence of points of S which constitute the sequence of vertices around the hull.

The convex hull problem was one of the first problems in the field of computational geometry to have been studied from the point of view of computational complexity. In fact, efficient algorithmic solutions were proposed even before the term "computational geometry" was coined. This, along with its very extensive analysis in recent years, reflects both the theoretical and practical importance of the problem.

Of the convex hull algorithms proposed so far several have $O(n\log n)$ worst case time bounds [4,8,14,15,17], where n is the size of the input point set. Shamos [17] even argued that $O(n\log n)$ time bound is worst case optimal. He observed that a set S of n real numbers could be sorted by finding the convex hull of the planar set $S' = \{ (x,x^2) \mid x \in S \}$. But sorting, of course, has an $\Omega(n\log n)$ lower bound on a wide range of computational models. Yao [19] and on weaker computational models Avis [2], van Emde Boas [7], and Preparata and Hong [15] proved the $\Omega(n\log n)$ bound for a less demanding version of the convex hull problem: just the vertices of the convex hull are to be identified, irrespective of their sequence.

In contrast to the results above, it is interesting to observe that algorithms exist which solve the planar convex hull problem in $O(nH)$ time, where H is the number of vertices found to be on the hull [6,9]. For small H, these algorithms seem to be superior to the $O(n\log n)$ methods. (This, of course, does not contradict the previously cited lower bound results, as H could be as large as n). It is notable, however, that all of the lower bound arguments mentioned above are insensitive to H in that they assume that some fixed fraction of the data points are vertices of the convex hull.

In this paper we present a convex hull algorithm with worst case time complexity $O(n\log H)$. Thus its running time is not only sensitive to both n and H, but it is also worst case optimal in the traditional sense when the running time is measured as a function of n only. However, we also show that our algorithm is asymptotically worst case optimal even if the complexity of the problem is measured as a function of both n and H.

Our algorithm is based on a variation of the divide-and-conquer paradigm that appears to be interesting in its own right. Traditional divide-and-conquer algorithms adhere to the following strategy: First break the problem into subproblems (*divide*), then recursively solve the subproblems (*conquer*), and finally combine the subsolutions to form the global solution (*marry*). Our algorithm reverses the last two steps. After dividing the problem it first determines how the solutions of the subproblems will combine (without actually computing them!) and then proceeds to solve the subproblems recursively. We thus call this approach the "marriage-before-conquest" principle. Its advantage lies in the fact that it allows to remove parts of the subproblems that upon merging (or marrying) turn out to be redundant. Thus it reduces the sizes of the subproblems that are to be solved recursively. We have recently been able to apply the marriage-before-conquest principle also successfully to the maximal vector problem [10]. It remains to be seen whether this principle has other applications.

Section 2 and 3 of this paper describe our new algorithm. In section 4 we show how our algorithm can be randomized, and section 5 deals with the lower bound aspects of the convex hull problem. Throughout the paper, unless stated otherwise, we deal with sets of points in the plane. For a point p, $x(p)$ and $y(p)$ denote its standard cartesian coordinates. We will feel free to use loose but descriptive geometric terminology such as "vertical line", "a point lies above a line", etc.

II. The Main Algorithm

In this section we show how the "marriage-before-conquest" principle can be used for an improved convex hull algorithm. We construct the convex hull in two pieces, the upper hull and the lower hull (see Figure 2.1). It should be clear that if the two chains forming the upper and lower hull are given, they can be concatenated in constant time (at most two vertical edges may need to be inserted) to yield the sequence of vertices around the hull. Also observe that an algorithm for constructing the upper hull could easily be modified to construct the lower hull also. Therefore we concentrate at first on constructing an algorithm for finding the sequence of vertices on the upper hull.

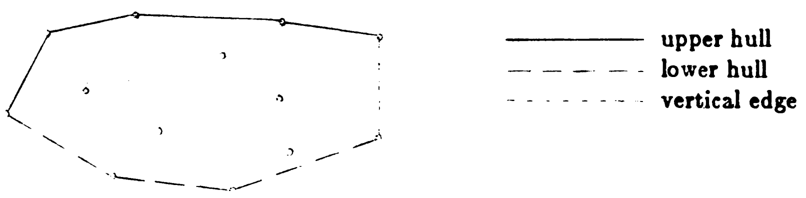

Figure 2.1

Exploiting the "marriage-before-conquest" principle, our convex hull algorithm should do something like the following: First find a vertical line that divides the given point set in two approximately equal sized parts. Next determine the "bridge" crossing this line, i.e. the edge of the upper hull that intersects this line. Eliminate the points that lie underneath the bridge, and finally apply the algorithm recursively to the two sets of the remaining points on the left and right side of the vertical line.

The only difficult part in such an algorithm appears to be the construction of the bridge. We show a linear time solution to this problem in Section 3.

The following PIDGIN-ALGOL routine presents our convex hull algorithm in some detail. It takes as input a set $S = \{p_1, \ldots, p_n\}$ of n points in the plane and prints the sequence of indices of the vertices on the upper hull of S. It uses the function BRIDGE specified in section 3, which given a set $S \subset \mathbf{R}^2$ and a real a returns the indices of the left and right endpoint of the edge of the upper hull that intersects the vertical line $L = \{(x,y) \mid x=a\}$.[1]

Algorithm 2.1:

Procedure UPPER-HULL(S)

1. Initialization

Let min and max be the indices of two points in S that form the left and right endpoint of the upper hull of S respectively, i.e.

$$x(p_{min}) \le x(p_i) \le x(p_{max}) \text{ and}$$
$$y(p_{min}) \ge y(p_i) \quad \text{if} \quad x(p_{min}) = x(p_i),$$
$$y(p_{max}) \ge y(p_i) \quad \text{if} \quad x(p_{max}) = x(p_i) \quad \text{for } i=1,\ldots,n.$$

If $min = max$ then print min and stop.

Let $T := \{p_{min}, p_{max}\} \cup \{\, p \in S \mid x(p_{min}) < x(p) < x(p_{max}) \,\}$.

2. CONNECT(min, max, T)
where CONNECT(k,m,S) is

[1] In the case that two edges of the upper hull, (p_i,p_j) and (p_j,p_k), intersect L, i.e vertex p_j lies on L, BRIDGE will return (j,k).

begin

2.1 Find a real number a such that
$x(p_i) \le a$ for $\lceil |S|/2 \rceil$ points in S and
$x(p_i) \ge a$ for $\lfloor |S|/2 \rfloor$ points in S.

2.2 Find the "bridge" over the vertical line $L = \{(x,y) \mid x=a\}$, i.e
$(i,j) :=$ BRIDGE(S,a).

2.3[2] Let $S_{left} := \{p_i\} \cup \{p \in S \mid x(p) < x(p_i)\}$.
Let $S_{right} := \{p_j\} \cup \{p \in S \mid x(p) > x(p_j)\}$.

2.4 If $i=k$ then print(i)
else CONNECT(k,i,S_{left}).
If $j=m$ then print(j)
else CONNECT(j,m,S_{right}).

end.

Theorem 2.1:
The above algorithm correctly determines the sequence of vertices on the upper hull of S in $O(n)$ space and $O(n \log H_s)$ time, where H_s is the number of edges on the upper hull of S.

Proof:
If the upper hull of S consists if only one vertex (i.e. all of S lies on one vertical line) then the algorithm is trivially correct and reports that vertex in linear time in step 1.
Otherwise the correctness of the algorithm follows from an inductive argument. A call CONNECT(k,m,S) discovers a previously unknown edge (p_i,p_j) on the upper hull. If p_i turns out to be the leftmost vertex of the upper hull its index will be printed, otherwise the recursive call CONNECT(k,i,S_{left}) will cause the sequence of vertices of the upper hull from p_k up to p_i to be printed. Similarly, if p_j is the rightmost vertex of the upper hull its index will be printed, otherwise the call CONNECT(j,m,S_{right}) will cause the portion of the upper hull from p_j up to p_m to be printed.

For the complexity bounds first observe that step 1 of the algorithm can easily be implemented to run in linear time. Thus it remains to show that the procedure CONNECT takes no more than $O(n \log H_s)$ time. Note that using the median finding algorithm of Blum et al. [1, p.99] and using our bridge finding algorithm of section 3 steps 2.1 to 2.3 can be implemented to run in linear time. Thus the running time of CONNECT is determined by $f(|S|,H_s)$ where the function f must satisfy the recurrence relation

$$f(n,h) \le \begin{cases} cn & \text{if } h=2 \\ cn + \max\limits_{h_l+h_r=h} \{ f(\frac{n}{2},h_l) + f(\frac{n}{2},h_r) \} & \text{if } h>2, \end{cases}$$

where c is some positive constant and $n \ge h > 1$.

We claim that $f(n,h) = O(n \log h)$. To prove this we show that $f(n,h) = cn \log h$ satisfies the above recurrence relation. This is trivially true for the base case $h=2$. For $h>2$ note that

$$f(n,h) \le cn + \max_{h_l+h_r=h} \{ c\frac{n}{2}\log h_l + c\frac{n}{2}\log h_r \}$$

---

[2] S_{left} contains p_i and the points of S to the left of the vertical line through p_i. M. McQueen from McGill University has pointed out that S_{left} could be restricted to contain p_k, p_i and all the points of S above the straight line through p_k and p_i. S_{right} can be restricted analogously.

$$= cn + \frac{1}{2} cn \max_{h_l + h_r = h} \{ \log(h_l h_r) \}.$$

Using elementary calculus it is easy to verify that the maximum is realized when $h_l = h_r = \frac{h}{2}$. Thus

$$f(n,h) \leq cn + \frac{1}{2} cn \log\left(\frac{h}{2}\right)^2 = cn + cn \log\left(\frac{h}{2}\right)$$

$$= cn + cn \log h - cn = cn \log h.$$

The linear space bound is trivial.

Q.E.D.

Corollary:
The convex hull of a set of n points in the plane can be found in time $O(n \log H)$ using $O(n)$ space, where H is the number of vertices found to be on the hull.

III. Finding the Bridge

We are given a set S of n points in the plane and a vertical line L which has points of S to its left and right. We are to find the edge of the upper hull of S that intersects L. If two edges intersect L, i.e. L contains a vertex v of the upper hull, we want to identify the edge for which v is the left endpoint. Call this edge the <u>bridge</u> and its endpoints <u>bridge points</u> (see Figure 3.1). Let us define a <u>supporting line</u> of S to be a non-vertical straight line which contains at least one point of S but has no points of S above it. Obviously the bridge must be contained in some supporting line. Call this line b and let s_b be the slope of b.

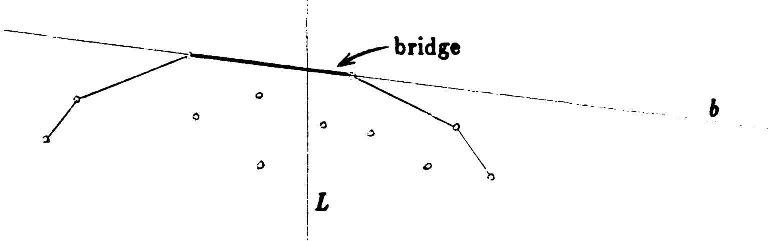

Figure 3.1

For our purposes, finding the bridge means identifying the two bridge points. One possible way of achieving this is to successively eliminate points from S as candidates for bridge points. For this purpose we pair up the points of S into $\lfloor n/2 \rfloor$ couples. The following two lemmas show how forming pairs of points facilitates the elimination of candidates for bridge points.

Lemma 3.1:
Let p,q be a pair of points of S.
If $x(p)=x(q)$ and $y(p)>y(q)$ then q cannot be a bridge point.

Proof : trivial.

Lemma 3.2:
Let p,q be a pair of points of S with $x(p)<x(q)$, and let s_{pq} be the slope of the straight line h through p and q.

(1) If $s_{pq} > s_b$ then p cannot be a bridge point.

(2) If $s_{pq} < s_b$ then q cannot be a bridge point.

Proof : (for case (1); the proof for case (2) is symmetrical)
Assume p was a bridge point. By virtue of $s_{pq}>s_b$ and $x(p)<x(q)$, q would lie above the bridge line b which would contradict the fact that b is a supporting line of S (see Figure 3.2).

Q.E.D.

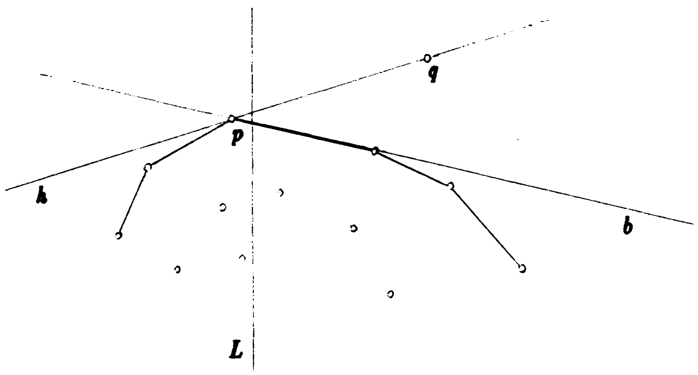

Figure 3.2

These two lemmas can be used to eliminate a bridgepoint candidate from every one of the $\lfloor n/2 \rfloor$ pairs. However, it is not clear at first how a condition like $s_{pq} > s_b$ can be tested without explicitly knowing s_b, the slope of b, and hence knowing the bridge, which after all is the entity that we want to compute. The solution to this problem is suggested by the following lemma:

Lemma 3.3:
Let h be the supporting line of S with slope s_h.

(1) $s_h < s_b$ iff h contains only points of S that are strictly to the right of L.

(2) $s_h = s_b$ iff h contains a point of S that is strictly to the right of L and a point of S that is to the left of or on L.

(3) $s_h > s_b$ iff h contains only points of S that are to the left of or on L.

Proof : trivial.

Thus to test whether $s_{pq} > s_b$ it suffices to find the supporting line h of S with slope s_{pq} and to determine whether h contains points of S to the right or to the left of L. Of course, finding this supporting line h requires linear time which is clearly too expensive to be done for every one of the $\lfloor n/2 \rfloor$ pairs individually. However, this problem can be overcome by judiciously choosing a slope s_h with the property that if $s_h > s_b$ then $s_{pq} > s_h$ (and hence $s_{pq} > s_b$) for a large number of pairs p,q and, if $s_h < s_b$ then $s_{pq} < s_h$ (and hence $s_{pq} < s_b$) for a large number of pairs p,q. A natural choice for an s_h with this property is the median of the slopes of the lines defined by the $\lfloor n/2 \rfloor$ pairs of points.

Now we are ready to give a more detailed PIDGIN-ALGOL description of our bridge finding algorithm. The function BRIDGE(S,a) takes as parameters a set $S = \{p_1,...,p_n\}$ of $n > 1$ points and a real number a representing the vertical line $L = \{(x,y) \mid x=a\}$. It is assumed that the point p_{min} in S with minimum x-coordinate is unique and that $x(p_{min}) \le a$. Similarly, the point p_{max} in S with maximum x-coordinate is assumed to be unique and with $x(p_{max}) > a$. BRIDGE(S,a) returns as its value a pair (i,j), where p_i and p_j are the left and right bridge point respectively.

Algorithm 3.1

Function BRIDGE(S,a)

0. *CANDIDATES* := $\emptyset$

1. If $|S|=2$ then return((i,j)), where $S=\{p_i,p_j\}$ and $x(p_i)<x(p_j)$.

2. Choose $\lfloor |S|/2 \rfloor$ disjoint sets of size 2 from S.
   If a point of S remains, then insert it into *CANDIDATES*.
   Arrange each subset to be an ordered pair (p_i,p_j), such that $x(p_i) \le x(p_j)$.
   Let *PAIRS* be the set of these ordered pairs.

3. Determine the slopes of the straight lines defined by the pairs. In case the slope does not exist for some pair, apply Lemma 3.1, i.e:
   For all (p_i,p_j) in *PAIRS* do
   if $x(p_i)=x(p_j)$ then delete (p_i,p_j) from *PAIRS*
   if $y(p_i)>y(p_j)$ then insert p_i into *CANDIDATES*
   else insert p_j into *CANDIDATES*
   else let $k(p_i,p_j) := \dfrac{y(p_i)-y(p_j)}{x(p_i)-x(p_j)}$.

4. Determine K, the median of $\{ k(p_i,p_j) \mid (p_i,p_j) \in PAIRS \}$.

5. Let *SMALL* := $\{ (p_i,p_j) \in PAIRS \mid k(p_i,p_j) < K \}$.
   Let *EQUAL* := $\{ (p_i,p_j) \in PAIRS \mid k(p_i,p_j) = K \}$.
   Let *LARGE* := $\{ (p_i,p_j) \in PAIRS \mid k(p_i,p_j) > K \}$.

6. Find the set of points of S which lie on the supporting line h with slope K, i.e.:

   Let MAX be the set of points $p_i \in S$, s.t. $y(p_i) - K * x(p_i)$ is maximum.
   Let p_k be the point in MAX with minimum x-coordinate.
   Let p_m be the point in MAX with maximum x-coordinate.

7. Determine if h contains the bridge, i.e.:
   if $x(p_k) \leq a$ and $x(p_m) > a$ then return((k,m)).

8. h contains only points to the left of or on L:
   if $x(p_m) \leq a$ then
      for all $(p_i,p_j) \in LARGE \cup EQUAL$ insert p_j into $CANDIDATES$.
      for all $(p_i,p_j) \in SMALL$ insert p_i and p_j into $CANDIDATES$.

9. h contains only points to the right of L:
   if $x(p_k) > a$ then
      for all $(p_i,p_j) \in SMALL \cup EQUAL$ insert p_i into $CANDIDATES$.
      for all $(p_i,p_j) \in LARGE$ insert p_i and p_j into $CANDIDATES$.

10. return(BRIDGE($CANDIDATES,a$)).

Theorem 3.1:
The function BRIDGE outlined above correctly determines the left and right bridge point in $O(n)$ worst case time and space.

Proof :
The algorithm is trivially correct if S contains only two points. As long as S contains more than two points, BRIDGE either finds the bridge in step 7 or discards redundant points of S applying the rules of Lemma 3.1 and 3.2 (steps 3,8,9) and calls itself recursively with a smaller pointset. Using the linear time median algorithm of Blum et al. [1, p.99], the body of BRIDGE without the recursive call can be executed in linear time and space. Furthermore, at least one quarter of the points of S are eliminated and not contained in $CANDIDATES$. Thus the worst case time and space requirements for the algorithm are bounded by

$$f(n) = \begin{cases} O(1) & n=2 \\ f(\frac{3n}{4}) + O(n) & n>2. \end{cases}$$

But it is well known that such a recursive function is $O(n)$ [1,p.64].

Q.E.D.

At this point we want to mention that our bridge finding algorithm was inspired by the linear time two variable linear programming algorithms of M. Dyer [5] and N. Megiddo [13]. A closer look even shows that the bridge problem can be formulated as a linear programming problem. However, for the sake of simplicity and completeness it seems worthwhile to spell out the bridge finding algorithm explicitly.

IV. The Expected Time Case

The divide-and-conquer algorithms in the two preceding sections are not terribly complicated. At first sight it even seems possible to actually implement these algorithms in some high level programming language in an hour's time, or so. However, one quickly discovers that the major obstacle to doing so is the median find algorithm. Thus quite naturally the question arises whether it is possible to do without it.

The median find algorithm is used in our algorithms to find a vertical line that divides a given point set evenly. What happens if we follow the example of Quicksort and choose a separating line at random? Ample experimental results have shown that Quicksort is one of the fastest sorting algorithms and these results have been supported by a careful theoretical analysis of the algorithm [11,16]. As it turns out the method of choosing a separator at random can also be successfully applied to our algorithms, thus changing the worst case time complexity to $O(n^2)$ but retaining the $O(n\log H)$ expected case time complexity.

Theorem 4.1:
If step 2.1 in Algorithm 2.1 is replaced by

"2.1 Let $a = x(p_i)$, where p_i is randomly chosen from $S-\{p_m\}$ such that the choice of every point in $S-\{p_m\}$ is equally likely."

then the modified algorithm has $O(n\log H_s)$ expected case time complexity.

Proof:
The expected case running time of the modified algorithm can be bounded by the function g that must satisfy the following recurrence relation:

$$g(n,h) \leq \begin{cases} bn & \text{if } n \geq h = 2 \\ bn + \dfrac{1}{n-1} \displaystyle\sum_{1\leq i<n} \max_{h_l+h_r=h} \{\, g(i,h_l) + g(n-i,h_r) \,\} & \text{if } n \geq h > 2, \end{cases}$$

where b is some positive constant.

We claim that $g(n,h) = O(n\log h)$, i.e. there is positive real constant c, such that for all $n \geq h \geq 2$, $g(n,h) \leq cn\log h$ [3]. We prove our claim by induction.

The claim is trivially true for all n if $h=2$ and for all $n \leq 5$ otherwise. Now we want to show the claim for some $n>5$ and $h<n$ on the assumption that $g(n',h') \leq cn'\log h'$ for all $n'<n$ and $h'<h$. By definition of g and our inductive assumption we thus have

$$g(n,h) \leq bn + \frac{1}{n-1} \sum_{1\leq i<n} \max_{h_l+h_r=h} \{ ci\log h_l + c(n-i)\log h_r \}.$$

Using elementary calculus it is easy to show that for every i the maximum is realized when $h_l = i\frac{h}{n}$ and $h_r = (n-i)\frac{h}{n}$. Therefore

$$g(n,h) \leq bn + \frac{c}{n-1} \sum_{1\leq i<n} \left(i\log i\frac{h}{n} + (n-i)\log(n-i)\frac{h}{n} \right)$$

$$= bn + \frac{c}{2(n-1)} \sum_{1\leq i<n} i\log i\frac{h}{n}$$

[3]In this proof we use w.l.o.g. the natural logarithm.

$$= bn + \frac{c}{2(n-1)}\log\frac{h}{n}\sum_{1\le i<n} i + \frac{c}{2(n-1)}\sum_{1\le i<n} i\log i.$$

As $\sum_{1\le i<n} i\log i \le \frac{1}{2}n^2\log n - \frac{1}{4}n^2$ (see [1,p.94]) and $\sum_{1\le i<n} i = \frac{1}{2}n(n-1)$ we have

$$g(n,h) \le bn + \frac{c}{4}n\log h - \frac{c}{4}n\log n + \frac{c}{4}\frac{n}{n-1}n\log n - \frac{c}{8}\frac{n^2}{n-1}$$

$$\le bn - \frac{c}{8}n + \frac{c}{4}n\frac{\log n}{n-1} + \frac{c}{4}n\log h.$$

As $\frac{\log n}{n-1} < \frac{1}{2}$ for all integers $n>5$, there exists a real constant $c>0$ such that $bn - \frac{c}{8}n + \frac{c}{4}n\frac{\log n}{n-1} < 0$ for all $n>5$ and hence

$$g(n,h) \le \frac{c}{4}n\log h < cn\log h.$$

Q.E.D.

The median find algorithm is used on one more occasion in our algorithms: in the bridge finding procedure. Again we can dispense with the median find algorithm and use random choice instead. It turns out that in that case the worst case complexity of our bridge finding procedure is $O(n^2)$, however the expected case running rime is still $O(n)$.

Theorem 4.3: If step 4 of Algorithm 3.1 is replaced by

"4. Randomly choose an element (p_i,p_j) from *PAIRS* such that the choice of every element is equally likely, and let $K := k(p_i,p_j)$,"

then the modified algorithm has expected case time complexity $O(n)$.

Proof:
The expected case running time of the modified algorithm is pessimistically described by the function f, where for some positive constant b

$$f(n) \le \begin{cases} bn & \text{if } n\le 2 \\ bn + \frac{4}{n}\sum_{1\le i<n/4} f(n-i) + \frac{4}{n}\sum_{n/4\le i\le n/2} f(n-(i-\frac{n}{4})) & \text{if } n>2. \end{cases}$$

It is an easy exercise in induction to show that $f(n) = O(n)$.

Q.E.D.

V. Lower Bounds

The results of this section demonstrate that our $O(n\log H)$ upper bound for the convex hull problem is the best possible on a quite general model of computation. Specifically, we prove an $\Omega(n\log H)$ lower bound for this problem on d-th order algebraic decision trees, for any fixed d.

There exist at least four variants of the convex hull problem characterized by more or less stringent conditions on the form of the output. Let $S = \{p_1, \ldots, p_n\}$ be a set of points in $\mathbf{R}^2$, and let $ext(S)$ denote the set of vertices of the convex hull of S. The convex hull sequence problem asks for the elements of $ext(S)$ in consecutive cyclic order. The convex hull set problem asks for the elements of $ext(S)$ in arbitrary order. The convex hull multiset problem asks for a listing, in arbitrary order, of elements of S that coincide with elements of $ext(S)$. (This differs from the set problem only if S is a multiset). Finally, the convex hull size problem asks for the cardinality of $ext(S)$ (i.e. H).

It should be clear that the algorithm outlined in section 3 can be adapted to solve all of these problem variants in worst case time $O(n\log H)$. Furthermore, since the sequence variant is at least as hard as the set variant, which in turn is at least as hard as the size variant, it will suffice to demonstrate a lower bound on the convex hull size problem, preferably using input point sets with no multiplicities. In fact we establish a lower bound on the even weaker convex hull size verification problem: given S and H, confirm that $|ext(S)| = H$. We show that any d-th order algebraic decision tree algorithm for this verification problem must take $\Omega(n\log H)$ steps in the worst case, even if it can be assumed that all input points are distinct.

We follow Steele and Yao [18] and Ben-Or [3] in adopting algebraic decision trees as our model of computation. A d-th order algebraic decision tree algorithm (hereafter a tree algorithm) T for testing membership in a set $W \subset \mathbf{R}^n$ is a rooted tree whose internal nodes are labelled by multivariate polynomials of degree at most d and whose leaves are labelled either YES or NO. Each internal node has out-degree three; the edges are labelled $<, =$, and $>$ reflecting possible outcomes on comparison with 0. Every input $\vec{z} \in \mathbf{R}^n$ determines a unique root to leaf path in T in the obvious way. We say that T decides membership in W if, for every $\vec{z} \in \mathbf{R}^n$, $\vec{z}$ leads to a YES leaf of T if and only if $\vec{z} \in W$.

Yao [19] establishes an $\Omega(n\log n)$ worst case lower bound for the convex hull set problem on algebraic decision trees of order two. The result is generalized by Ben-Or [3], who demonstrates the same $\Omega(n\log n)$ lower bound for the convex hull size problem on algebraic decision trees of any fixed order d. Ben-Or's result is just one of a number of applications of the following general theorem concerning tree algorithms.

Theorem 5.1: [3, Theorem 8]

Let $W \subset \mathbf{R}^n$ be any set and let T be any d-th order algebraic decision tree that solves the membership problem for W.
If W has N disjoint connected components, then T must have height (and hence worst case complexity) $\Omega(\log N - n)$.

We use the following generalization of the element distinctness problem [3] to establish our lower bound. The multiset size verification problem asks to confirm, given a multiset $Z = \{z_1, \ldots, z_n\} \subset \mathbf{R}$ and an integer k, that Z has k distinct elements.

Corollary 5.1:

The multiset size verification problem requires $\Omega(n\log k)$ steps in the worst case, with any d-th order decision tree algorithm.

Proof :

It suffices to observe that the set $M_k = \left\{ (z_1, \ldots, z_n) \in \mathbf{R}^n \;\middle|\; |\{z_1, \ldots, z_n\}| = k \right\}$ has at least $max\{k^{n-k}, k!\}$ disjoint connected components.

Q.E.D.

We are now prepared to demonstrate our lower bound.

Theorem 5.2:

The convex hull size verification problem requires $\Omega(n \log H)$ steps, in the worst case, with any d-th order decision tree algorithm.

Proof :

We reduce the multiset size verification problem to the convex hull size verification problem in the following obvious way:

Let $Z = \{z_1, \ldots, z_n\}$ and k be an instance of the multiset size verification problem. Define $S = \{p_1, \ldots, p_n\} \subset \mathbf{R}^2$ by $p_i = (z_i, z_i^2)$. Then the set $ext(S)$ has exactly k elements if and only if Z has exactly k distinct elements.

Q.E.D.

The proof of the above theorem is somewhat disappointing in that the convex hull problem formed in the reduction has multiplicities (in fact all of its points) on the convex hull. This straightforward reduction leaves open the possibility that there exists an algorithm solving the convex hull size verification problem (or any of the other variants) in $o(n \log H)$ steps for point sets that are known a priori to contain no duplicates. Fortunately, we can strengthen our lower bound to include tree algorithms based on this rather dubious assumption as well. We will show that a convex hull algorithm that is only guarantueed to be correct when the input points are distinct could be used to solve a certain perturbed convex hull problem without input restrictions. An algorithm for this perturbed problem in turn yields a solution for the multiset size problem,

For the sake of notation let $(\vec{x}, \vec{y})$ be shorthand for $(x_1, \ldots, x_n, y_1, \ldots, y_n)$ and let $\overline{x}$ and $\overline{y}$ denote $\frac{1}{n}\sum_{i=1}^{n} x_i$ and $\frac{1}{n}\sum_{i=1}^{n} y_i$ respectively. Define

$$C_H = \left\{ (\vec{x}, \vec{y}) \in \mathbf{R}^{2n} \;\middle|\; |\, ext(\{(x_i, y_i) \mid 1 \leq i \leq n\})\,| = H \right\}, \text{ and}$$

$$P_H = \left\{ (\vec{x}, \vec{y}) \in \mathbf{R}^{2n} \;\middle|\; \begin{array}{c} |\, ext(\{(\, x_i + i(x_i - \overline{x})\epsilon \,,\, y_i + i(y_i - \overline{y})\epsilon \,) \mid 1 \leq i \leq n\})\,| = H \\ \text{for all } \epsilon > 0 \text{ sufficiently small} \end{array} \right\}.$$

Note that testing membership in C_H is the convex size verification problem. The intuitive meaning for P_H is the following: P_H encodes the point sets $\{(x_i, y_i) \in \mathbf{R}^2 \mid 1 \leq i \leq n\}$ with the property that if each point $p_i = (x_i, y_i)$ moved radially away from $\overline{p} = (\overline{x}, \overline{y})$ for sufficiently small but positive time ϵ at speed proportional to the index i and proportional to the distance from p_i to $\overline{p}$, then the convex hull of the new point set would have H extreme points. Observe therefore, that if $(\vec{x}, \vec{y}) \in C_H$ and the encoded 2-dimensional point set has no point on a convex hull edge, then $(\vec{x}, \vec{y}) \in P_H$.

The following lemma shows that the convex hull size verification problem with this dubious distinctness restriction is no easier to solve than the general membership problem for P_H.

Lemma 5.1:

Let T be any d-th order decision tree algorithm for deciding membership in C_H, assuming that all of the points (x_i, y_i), $1 \leq i \leq n$, are distinct.
Then there exists a d-th order decision tree T', with height(T') $\leq (d+1)$height(T), that decides membership in P_H without the distinctness assumption.

Proof:
We define a transformation on every subtree of T. The leaves of T are not changed (i.e. they retain their YES-NO labels). Consider an arbitrary subtree rooted at a vertex v_j with label $f_j(\vec{x}, \vec{y})$ (see Figure 5.1).

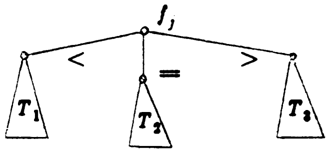

Figure 5.1

Define the multivariate polynomials $f_{j,0}, f_{j,1}, \ldots, f_{j,d}$ by the equality

$$f_j(\vec{x}', \vec{y}') = f_{j,0}(\vec{x}, \vec{y}) + f_{j,1}(\vec{x}, \vec{y})\epsilon + \cdots + f_{j,d}(\vec{x}, \vec{y})\epsilon^d,$$

where

$$\vec{x}' = (\, x_1 + (x_1 - \bar{x})\epsilon \,,\, x_2 + 2(x_2 - \bar{x})\epsilon \,, \ldots ,\, x_n + n(x_n - \bar{x})\epsilon \,), \text{ and}$$

$$\vec{y}' = (\, y_1 + (y_1 - \bar{y})\epsilon \,,\, y_2 + 2(y_2 - \bar{y})\epsilon \,, \ldots ,\, y_n + n(y_n - \bar{y})\epsilon \,).$$

Clearly, the degree of each $f_{j,i}$ is at most d.

Let T_i' be the transformed versions of T_i, $i=1,2,3$. The transformed version of the full subtree is given by Figure 5.2.

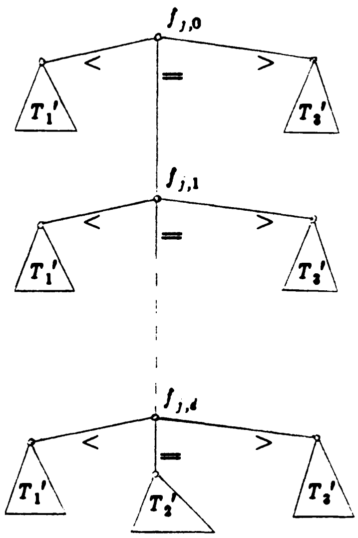

Figure 5.2

A straightforward inductive argument shows that height(T') $\leq (d+1)$height(T). The correctness of T' follows from the following observations.

i) If $\epsilon>0$ is chosen to be sufficiently small, then the set $\{ (x_i + i(x_i - \bar{x})\epsilon , y_i + i(y_i - \bar{y})\epsilon) , 1 \leq i \leq n \}$ has distinct elements.

ii) The decision tree T' with input $(\vec{x},\vec{y})$ agrees with the decision tree T with input $(\vec{x}',\vec{y}')$, provided $\epsilon>0$ is chosen to be sufficiently small.

Thus T' decides membership in P_H without assuming that all of the pairs (x_i,y_i) are distinct.

Q.E.D.

The next lemma shows that deciding membership for P_H is no easier than the multiset size verification problem.

Lemma 5.2:

The multiset size verification problem reduces to the membership problem for P_H.

Proof:

It suffices to note that as no three distinct points on a parabola can be collinear

$$(x_1, \ldots, x_n) \in M_H \Leftrightarrow (x_1, \ldots, x_n, x_1^2, \ldots, x_n^2) \in P_H.$$

Q.E.D.

The preceding corollary and lemmas immediately yield the final theorem:

Theorem 5.3:

The convex hull size verification problem requires $\Omega(n \log H)$ steps, in the worst case, with any d-th order decision tree algorithm, even if the input points may be assumed to be destinct.

VI. Conclusions

We have introduced a variation of the familiar divide-and-conquer paradigm and have illustrated this approach in the development of a new algorithm for the planar convex hull problem. Our algorithm unifies and improves the best worst case complexity bounds known for this problems in terms of the size of input and output (i.e. number of data points and number of hull vertices). In fact, we demonstrate that the algorithm is worst case optimal in terms of these two parameters in a very general model of computation.

In a companion paper [10] we apply the same strategy to the maximal vector problem. We are able to demonstrate an $O(n \log V)$ upper bound for the 2-dimensional maximal vector problem and an $O(n(\log V)^{d-2})$ upper bound for the d-dimensional maximal vector problem, for $d \geq 3$. These bounds tighten the best bounds known for the maximal vector problem. It remains to be seen whether our "marriage-before-conquest" approach can be applied successfully to other problems.

The results of this paper suggest other more specific open problems as well. In particular, it is natural to ask whether our results on planar convex hulls (like those for the maximal vector problem) extend to higher dimensions. For example, does there exist an $O(n \log H)$ algorithm for the 3-dimensional convex hull problem?

Another practical open question is whether, like the algorithm of Bentley and Shamos [4], our convex hull algorithm modified as suggested in footnote 2 has linear expected time complexity for reasonable input point distributions. We suspect that this is the case.

Acknowledgements:

We are grateful to John Gilbert for his very careful reading of the manuscript.

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman: *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA (1974).

[2] D. Avis: *On the Complexity of Finding the Convex Hull of a Set of Points.* McGill Univ., Montreal, School of Computer Science, Report SOCS 79.2 (1979).

[3] M. Ben-Or: *Lower Bounds for Algebraic Computation Trees.* Proc. 15th ACM STOC, (1983) 80-86.

[4] J.L. Bentley and M.I. Shamos: *Divide and Conquer for Linear Expected Time.* Information Processing Letters 7, (1978) 87-91.

[5] M.E. Dyer: *Two Variable Linear Programs are Solvable in Linear Time.* Manuscript; Dept. of Mathematics and Statistics, Teeside Polytechnic, Middlesbrough, Cleveland, UK (1982).

[6] U.F. Eddy: *A New Convex Hull Algorithm for Planar Sets.* ACM Transaction on Mathematical Software 3, (1977) 398-403 and 411-412.

[7] P. van Emde Boas: *On the O(nlogn) Lower-Bound for Convex Hull and Maximal Vector Determination.* Information Processing Letters 10, (1980) 132-136.

[8] R.L. Graham: *An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set.* Information Processing Letters 1, (1972) 132-133.

[9] R.A. Jarvis: *On the Identification of the Convex Hull of a Finite Set of Points in the Plane.* Information Processing Letters 2, (1973) 18-21.

[10] D.G. Kirkpatrick and R. Seidel: *Output Size Sensitive Algoritms for Finding Maximal Vectors.* in preparation.

[11] D.E. Knuth: *The Art of Computer Programming, Volume 3.* Addison-Wesley, REading, MA (1973).

[12] H.T. Kung, F. Luccio, and F.P. Preparata: *On Finding the Maxima of a Set of Vectors.* Journal of the ACM 22, (1975) 469-476.

[13] N. Megiddo: *Linear-Time Algorithms for Linear Programming in R^3 and Related Problems.* Proc. 23-rd FOCS, (1982) 329-338.

[14] F.P. Preparata: *An Optimal Real Time Algorithm for Planar Convex Hulls.* Communications of the ACM 22, (1979) 402-405.

[15] F.P. Preparata and S.J. Hong: *Convex Hulls of Finite Sets of Points in Two and Three Dimensions.* Communications of the ACM 20, (1977) 87-93.

[16] R. Sedgewick: *Quicksort.* Stanford University, Stanford, California; Ph.D. Thesis, (1975).

[17] M.I. Shamos: *Computational Geometry.* Yale University, New Haven, Connecticut; Ph.D. Thesis, (1978).

[18] J.M. Steele and A.C. Yao: *Lower Bounds for Algebraic Decision Trees.* Journal of Algorithms 3, (1982) 1-8.

[19] A.C. Yao: *A Lower Bound to Finding Convex Hulls.* Journal of the ACM 28, (1981) 780-789.