

Fast Linear Expected-Time Algorithms for Computing Maxima and Convex Hulls

Jon L. Bentley*
 Kenneth L. Clarkson*
 David B. Levine#

ABSTRACT

This paper examines the expected complexity of boundary problems on a set of N points in K -space. We assume that the points are chosen from a probability distribution in which each component of a point is chosen independently of all other components. We present an algorithm to find the maximal points using $KN + O(N^{1-1/K} \log^{1/K} N)$ expected scalar comparisons, for fixed $K \geq 2$. A lower bound shows that the algorithm is optimal in the leading term. We describe a simple maxima algorithm that is easy to code, and present experimental evidence that it has similar running time. For $K=2$ and $K=3$, an algorithm computes the convex hull of the set in $2KN + O(N^{1-1/K} \log^{1/K} N)$ expected scalar comparisons; for fixed $K \geq 4$, that time suffices to find a superset of the extreme points of the convex hull with expected size $O(\log^{K-1} N)$.

1. Introduction

This paper examines the expected complexity of two problems in K -dimensional computational geometry: computing the maximal points and the convex hull of a point set. These problems arise in a variety of applications; we will sketch some examples later in this paper. Because the maxima and the convex hull provide natural representations of the boundary of the point set S , both objects play a fundamental role in computational geometry.

All the algorithms in this paper exploit a single insight: for these problems, a certificate of exclusion typi-

cally can quickly demonstrate that most of the N input points are not in the final output. For maxima, a particular point usually dominates all but $o(N)$ of the points in S (assuming that each component of a point is chosen independently of all other components). For convex hulls, a particular hyperrectangle typically contains all but $o(N)$ points in S but contains no points on the hull. These certificates are used in a sieve approach: a first phase sieves out points not in the output structure, and a second phase constructs the final output. Previous papers present algorithms for these problems with $O(N)$ expected time; our certificate-based algorithms show that the constant factors hidden in the big-ohs can be quite small. For instance, the best previous algorithm for convex hulls has the leading term of $2^K(K+1)N$ while our algorithm has the leading term of $2KN$.

Section 2 of this paper describes maxima, and Section 3 describes convex hulls. Conclusions are then offered in Section 4.

2. Maxima

In this section we will study the problem of computing the maximal points in a point set. The first subsection provides definitions and reviews previous work. The second subsection presents "theoretical" algorithms and proofs of their run time, while the third subsection presents a related "practical" algorithm and experiments and conjectures regarding its run time. The final subsection describes the history of the algorithms.

Preliminaries

Throughout this paper we will deal with a set S of N points in K -space. Point P is said to *dominate* point Q if each coordinate of P is greater than the corresponding coordinate of Q . A point in S that is not dominated by any other point in S is a *maximal* point; the undominated points in S are the *maxima* of the point set. Maxima problems and algorithms are often discussed in terms of N vectors of

* AT&T Bell Laboratories, Murray Hill, NJ 07974.

Department of Computer Science, Williams College, Williamstown, MA 01267. This work was performed while this author was visiting AT&T Bell Laboratories.

length K ; we phrase our discussion in the equivalent language of points for consistency with the description of convex hulls in Section 3.

The problem of computing maxima is basic in computational geometry because the maxima are an interesting characterization of the boundary of a point set. Maxima also arise in diverse applications. Bentley, Kung, Schkolnick and Thompson [1978] describe how maxima are useful in determining the run time of dynamic programming algorithms. Preparata and Shamos [1985] use maxima to solve the "floating currency problem" of economics. In the next section we will see how maxima are useful in computing convex hulls.

Becker, Denby, McGill and Wilks [1987] use maxima to analyze the data in Boyer and Savageau's [1985] *Places Rated Almanac*. The almanac rates 329 metropolitan areas within the United States on 9 different dimensions (climate, housing cost, health care, etc.). Each value is assigned a numerical value, so the data set may be viewed as 329 points in 9-space. The almanac ranked the cities by summing the variables, with equal weight assigned to each of the 9 dimensions. By assigning different weights to the variables, 134 cities can hold first place, and 150 cities can be ranked last. Becker, Denby, McGill and Wilks use the domination relationship as a more natural comparison of cities, both with the 9 original variables and several subsets of 8 variables.

Kung, Luccio and Preparata [1975] describe an algorithm for computing the maxima of N points in K -space in $O(N \log^{K-2} N + N \log N)$ time. Bentley [1980] gives a simpler description of their algorithm using the paradigm of multidimensional divide-and-conquer; he also develops a corresponding data structure for maxima searching (to determine whether a new query point is maximal in the set). Monier [1980] presents a general scheme for analyzing the run time of multidimensional divide-and-conquer algorithms; he shows that for fixed $K \geq 3$, the number of basic operations to build a maxima search tree is $(N \lg^{K-2} N)/(K-2)! + \Theta(N \log^{K-3} N)$.

Bentley, Kung, Schkolnick and Thompson [1978] study the maxima problem under the assumption that the input point set is drawn from a distribution with *Component Independence*, which we shall abbreviate as the *CI* property. A distribution has the *CI* property if the K components in each point are chosen independently from continuous distributions. Points uniform over the K -dimensional hypercube display the *CI* property, as do points chosen uniformly from any rectilinearly oriented hyperrectangle. *CI* distributions can be more complex: the first component might be chosen from a normal distribution, the second from an exponential distribution, the third from a beta distribution, etc. With probability 1, all components in a point set will be distinct. To study a maxima algorithm that operates by comparing components, one may combinatorially model a *CI* point set by a random selection from the $(N!)^K$ K -tuples of permutations of $1, 2, \dots, N$. Bentley, Kung, Schkolnick and Thompson [1978] show that

the expected number of maxima in a *CI* set of N points in K -space is $O(\log^{K-1} N)$; they use that fact to construct a maxima algorithm with $O(N)$ expected time for any fixed $K \geq 1$. Bentley and Shamos [1978] generalize that technique to yield linear expected-time divide-and-conquer algorithms for other problems and other input distributions. The key fact in their proofs is that if the input is of size N , then the expected size of the output is $O(N^P)$, for some $P < 1$; we will use their result later in this paper.

A fundamental operation in some maxima algorithms is to compare the two points A and B for domination. There are four possible outcomes: A dominates B , B dominates A , A equals B , and A and B are incomparable (none of the three previous results hold). Algorithm C determines that outcome using at most K scalar comparisons. (The code defines domination to be strict; it could easily be changed to report that A dominates B if it is greater in some components and equal in the remainder.)

```

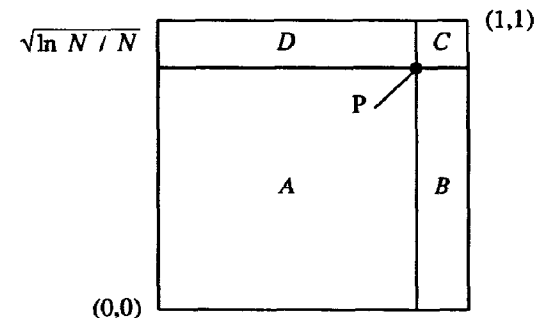
I := 1
while I <= K and A[I] != B[I] do
    I := I + 1
if I > K then
    return A equals B
if A[I] > B[I] then
    for J := I+1 to K do
        if B[J] >= A[J] then
            return A, B are incomparable
    return A dominates B
else
    for J := I+1 to K do
        if A[J] >= B[J] then
            return A, B are incomparable
    return B dominates A

```

Algorithm C. Compare points A and B for domination.

Provably Fast Algorithms

We will first study a maxima algorithm specifically designed for input that is known to be N points distributed uniformly over the unit square. As we progress through this subsection, we will generalize the algorithm to higher dimensions and to general *CI* distributions. The idea underlying Algorithm M1 is illustrated in this picture:



1. Initialize the sets A , C and BD to empty. Let P denote the point $(1 - \sqrt{\ln N / N}, 1 - \sqrt{\ln N / N})$.
2. Compare each point Q in the set S to P . If P dominates Q , place Q in set A ; if Q dominates P , place Q in set C ; otherwise place Q in set BD .
3. If the set C is empty, then compute the maxima of set S by some other algorithm and return.
4. Because C is not empty, no maxima are in set A . Compute the maxima of $C \cup BD$ by some other algorithm.

Algorithm M1. Finding maxima on the unit square.

The point P at $(1 - \sqrt{\ln N / N}, 1 - \sqrt{\ln N / N})$ is not necessarily a member of the set S . It partitions the unit square into four rectangles: P dominates points in A , is dominated by points in C , and is incomparable to points in B and D . We will soon see that with high probability, rectangle C contains at least one point. Because that point dominates all points in A , it certifies that none of the points in A can possibly be maxima. We therefore sieve out those points and compute the maxima by examining the other three sets. The details of this approach are given as Algorithm M1.

We will now analyze the run time of Algorithm M1. Step 1 requires constant time, and Step 2 requires exactly N point comparisons, or $2N$ scalar comparisons. We must now compute the probability that no points fall in square C . Because each side of the square is of length $\sqrt{\ln N / N}$, its area is $\ln N / N$. The probability of a particular input point not lying in C is one minus that area. The probability that none of the N points is in C is therefore $(1 - \ln N / N)^N$, which is at most $e^{-\ln N} = 1/N$ since for any X , $(1 + X/N)^N \leq e^X$. The probability that set C is empty in Step 3 is therefore at most $1/N$.

If set C is empty in Step 3 of Algorithm M1, then we use an $O(N \log N)$ worst-case algorithm to compute the maxima; because the probability of this event is bounded above by $1/N$, the contribution to the expected time of the algorithm is $O(\log N)$. If Step 4 is executed, then the expected size of the set $C \cup BD$ is bounded above by N times the area of B , C and D (plus one because we know that set C contains at least one element), or $1 + 2N\sqrt{\ln N / N} = O(\sqrt{N \ln N})$. The structure of the subset $C \cup BD$ does not have the CI property assumed by Bentley, Kung, Schkolnick and Thompson [1978], but it does have the N^P property of Bentley and Shamos [1978] (the subset can be partitioned into two rectangles, $B \cup C$ and D , each of which exhibits the CI property). We can therefore apply Bentley and Shamos's [1978] linear expected-time maxima algorithm to the set $C \cup BD$. The expected running time of Step 4 of the algorithm is therefore $O(\sqrt{N \log N})$. The run time of the algorithm is summarized in this theorem.

Theorem 2-1. Algorithm M1 finds the maxima of N points chosen uniformly from the unit square in $O(N)$ expected time, using $N + O(\sqrt{N \log N})$ expected point comparisons.

It is straightforward to extend Algorithm M1 to K -space, when the N input points are chosen uniformly over the K -dimensional hypercube. The algorithm uses a point P in which each component is $1 - (\ln N / N)^{1/K}$. Point P dominates all points in the hypercube A , and is in turn dominated by all points in the hypercube C . The hypercube C has edge length $(\ln N / N)^{1/K}$ and volume $\ln N / N$. The probability that C is empty is bounded above by $1/N$; in that event, we use the worst-case algorithm of Kung, Luccio and Preparata [1975]. The expected number of undominated points (conditional on the fact that C is not empty) is bounded above by $1 + KN(\ln N / N)^{1/K}$ or $1 + K(N^{1-1/K} \ln^{1/K} N)$. The undominated points do not have the CI property, but they can be partitioned into K hyperrectangles that do, so the subset has the N^P property of Bentley and Shamos [1978] and the maxima can be computed in expected time linear in the size of the subset. This analysis yields the following theorem.

Theorem 2-2. Algorithm M1 finds the maxima of N points chosen uniformly from the K -dimensional hypercube in $O(N)$ expected time, using $N + O(N^{1-1/K} \log^{1/K} N)$ expected point comparisons, for any fixed K .

The analysis of the run time of Algorithm M1 relies on the fact that its input is known to be distributed uniformly over the K -dimensional hypercube. The next algorithm we will study, M2, assumes only that its input is from a CI distribution. The role played by the fixed real number $1 - (\ln N / N)^{1/K}$ in Algorithm M1 is replaced in Algorithm M2 by an order statistic. The order statistic is computed using Floyd and Rivest's [1975] selection algorithm, which selects the M^{th} largest element in a set of size N in $N + \min(M, N - M) + O(\sqrt{N})$ expected comparisons.

Step 1 of the algorithm uses $KN + O(\sqrt{N})$ scalar comparisons between point components, and Step 2 also requires $O(KN)$ time. By the CI property, the probability

1. Compute the $N(1 - (\ln N / N)^{1/K})$ -th largest element in each dimension; construct the point P to consist of those values in the appropriate dimensions.
2. Using the scalar comparisons made in Step 1, partition S into three sets: A contains points dominated by P , C contains points that dominate P , and B contains points incomparable to P .
3. If set C is empty, then compute the maxima of S and return.
4. Because C is not empty, no maxima are in set A . Compute the maxima of $B \cup C$.

Algorithm M2. Finding maxima from a CI distribution.

of any particular input point dominating P is $\ln N / N$, so the analysis of Steps 3 and 4 is exactly the same as in Algorithm M1. Thus we have the following theorem.

Theorem 2-3. Algorithm M2 finds the maxima of N points chosen from a K -dimensional CI distribution in $O(N)$ expected time, using $KN + O(N^{1-1/K} \log^{1/K} N)$ expected scalar comparisons, for any fixed $K \geq 2$.

We will now see a lower bound that shows that the leading term in Theorem 2-3 is optimal. Any correct maxima algorithm must examine all K coordinates of all non-maximal (undominated) points; otherwise, one of the unexamined components could contain a value large enough to make it maximal. Bentley, Kung, Schkolnick and Thompson [1978] showed that on the average there are $O(\log^{K-1} N)$ maxima in a CI set of N points in K -space. These facts combine to yield the following theorem.

Theorem 2-4. A correct maxima algorithm must examine at least $KN - O(\log^{K-1} N)$ point components, on the average, for a CI set of N points in K -space.

We suspect that this bound can be tightened to show that Algorithm M2 is near optimal in its second order term.

Conjecture 2-5. A correct maxima algorithm must examine at least $KN + \Omega(N^{1-1/K})$ point components, on the average, for a CI set of N points in K -space.

Notice that correct maxima algorithms need not necessarily examine all components of all points: for instance, if the first two components of all points satisfy the constraint $A_1 + A_2 = 1$, then those components alone establish that all N points are incomparable.

An Effective Heuristic Algorithm

The algorithms in the previous subsection were designed to be efficient for CI inputs and easy to analyze for that case, but they are not necessarily robust for real inputs. We will now study an algorithm that is easy to implement, very efficient for CI distributions, and somewhat robust for point sets from other distributions.

Algorithm M3 is an online algorithm that augments sequential search with a move-to-front heuristic. Its primary data structure is the sequence T of (indexes of) current maxima. Sequence T is originally empty, and at the conclusion of the algorithm it contains the maxima of S . The algorithm examines all input points in an arbitrary order. As the algorithm examines the input point Q , it compares Q to every point R in T . If Q dominates R , then R is removed from T . If Q is dominated by no R in T , then Q is appended to the end of T . If R dominates Q , then Q cannot be maximal; in this case R is also moved to the front of T . The move to front tends to keep near the front of T maxima that are "powerful" dominators (like point P in the analysis of Algorithm M1).[†] Algorithm M3 gives the details of the algorithm in pseudocode; it implements the sequence T in the array $\text{Max}[1.. \text{TopMax}]$.

```

TopMax := 1
Max[TopMax] := 1
for I := 2 to N do
  J := 1
  while J <= TopMax do
    if pt Max[J] dominates pt I then
      move Max[J] to front
      of Max[1..J]
      J := TopMax + 2
    else if pt I dominates pt Max[J] then
      shift Max[J+1..TopMax]
      to Max[J..TopMax-1]
      TopMax := TopMax - 1
    else if pt I equals pt Max[J] then
      J := TopMax + 2
    else // pts I, Max[J] incomparable
      J := J+1
  if J = TopMax + 1 then
    TopMax := TopMax + 1
    Max[TopMax] := I

```

Algorithm M3. A heuristic maxima algorithm.

We will turn now to an analysis of the run time of Algorithm M3. The expected size of the sequence T after M elements have been examined is precisely the expected number of maxima in a set of M points, which is monotone increasing in M . The expected size of T is therefore $O(\log^{K-1} N)$, and the expected total cost of the N searches is $O(N \log^{K-1} N)$. To understand the constants in the big-oh, a series of experiments investigated the average number of maxima. Ten point sets were generated for N at each power of two from 32 to 65536 and for each K from 2 to 5. The N points were uniformly distributed over the K -dimensional hypercube. Figure 1 plots the number of maxima observed (averaged over the ten point sets), using the dimension as the plot symbol.

The distribution of the maxima and the move-to-front heuristic might, however, make the algorithm substantially faster than the $O(N \log^{K-1} N)$ expected bound. We examined the running time in the same series of experiments shown in Figure 1. Figure 2 shows the average number of point comparisons used, divided by N . The plot symbol is the dimension; the lines denote mean values.

The average number of point comparisons per point is substantially less than the number of maxima in the set. At $N=65536$, the mean ratios were 1.037, 1.331, 3.998, and 20.49, for $D=2, 3, 4$, and 5, while the mean number of maxima were 13.2, 61.6, 268.9, and 881.6. When we first ran the experiments, we expected the ratios to increase with N (though we hoped not quite as quickly as the

[†] We conducted a simple experiment to test the importance of the move to front. We used the algorithm as described on ten random point sets with $K=2$ and $N=100,000$; the mean number of point comparisons was $1.0253N$. We ran the experiment again using random insertion into a sequence that is not reorganized, and the mean number of comparisons rose to $1.341N$. All later experiments therefore used move-to-front.

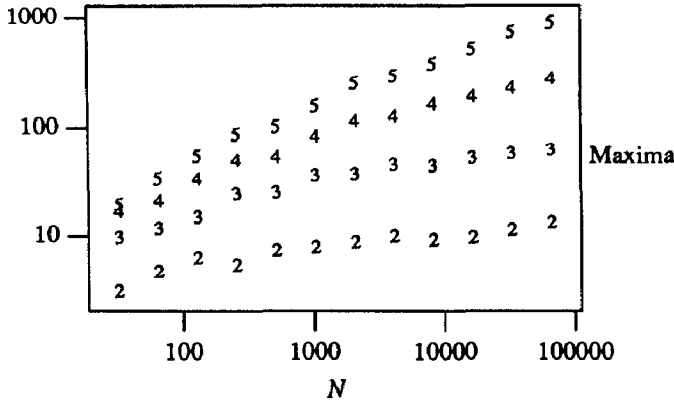


Figure 1. The number of maxima.

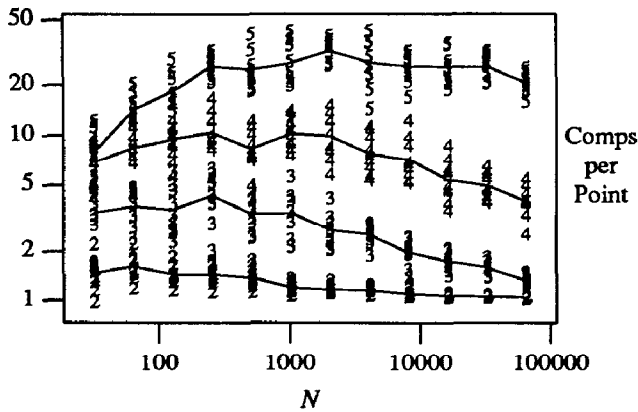


Figure 2. Comparisons per point.

$O(\log^{K-1} N)$ bound). When we examined Figure 2, however, we conjectured that the number of comparisons per point is approaching a different constant for each dimension K .

In Figure 2, though, each curve appears to grow to a peak and then decrease again. After running a handful of large experiments (N up to 1,000,000), we conjectured that the ratio approaches 1 for all values of K . To test this hypothesis, we re-expressed the data in Figure 2 by plotting the number of point comparisons beyond N , which we shall refer to as the *surplus* comparisons.[†] The means of surplus comparisons are shown Figure 3.

A weighted least squares regression shows that for $K=2$ the number of surplus comparisons grows as $7.6N^{.515} - 27.7$; the regression yields well-behaved residu-

[†] Our first attempt to examine the surplus comparisons resulted in a warning from the regression/plotting package that it had tried to take the logarithm of a negative number. Investigation showed that on one set of $N=32$ points in $K=2$ -space, Algorithm M3 found 2 maxima using 31 point comparisons. We first feared that this was evidence of a program bug, but examining the input point set showed us how it happened. We leave the explanation as a puzzle for the reader. (Hint: the probability of Algorithm M3 finding the maxima of a planar set in exactly $N-1$ point comparisons is at least $1/N^2$.)

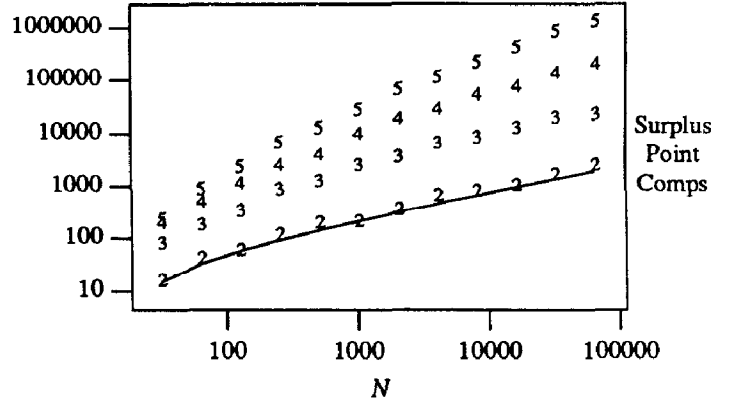


Figure 3. Surplus point comparisons.

als (apparently normally distributed). We have therefore plotted the function $7.6\sqrt{N} - 28$ on the Figure 3, which is a close fit to the experimental data. Algorithm M1 helps to explain why Algorithm M3 might behave this way: a single point very near $(1,1)$ tends to stay near the front of the sequence, and dominates all but roughly \sqrt{N} of the other inputs. Regressions for the other surplus values yield $37.1N^{.594} - 237$ for $K=3$, $31.8N^{.799} - 333$ for $K=4$, and $32.1N^{.966} - 733$ for $K=5$ (though the residuals are not particularly well behaved). This data and the heuristic arguments support the following:

Conjecture 2-6. Algorithm M3 finds the maxima of N points chosen from a K -dimensional CI distribution in $O(N)$ time, using $N + o(N)$ point comparisons, for any fixed dimension K .

We suspect that one might even be able to tighten the number of surplus comparisons to near $O(N^{1-1/K})$.

Our discussion so far has concentrated on CI distributions, which are known to have few maxima, on the average. We will turn now to a distribution that tends to have many maxima. Points in K -space from the *Ball* distribution are chosen uniformly from the positive orthant of the unit-radius sphere centered at the origin. When applied to Ball inputs, both Algorithms M1 and M2 find that point P is undominated and call the worst-case algorithm in Step 3, with probability approaching 1. Figure 4 shows the results of experiments running Algorithm M3 on the Ball distribution for $K=3$ and N between 32 and 65536; the circles show the number of maxima computed and the crosses show the number of comparisons per point.

A weighted least squares regression on the number of maxima shows that the expected number is $1.72N^{.661}$. We conjecture that the expected number of maxima in a K -dimensional Ball distribution is $\Theta(N^{1-1/K})$. Algorithm M3 compares every maximal point to every other maximal point; if a set has M maxima, then the algorithm makes at least $M(M-1)/2$ point comparisons. The algorithm therefore makes at least $\Theta(N^{2-2/K})$ comparisons for K -dimensional Ball distributions. Regression shows that the average number of comparisons per point grows as

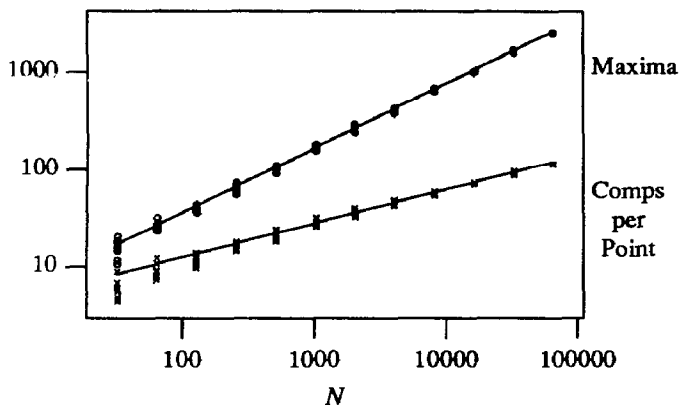


Figure 4. Performance on Ball input, $K=3$.

$2.51N^{.348}$, so most of the nonmaximal points are apparently discarded with relatively little computation. We ran a similar set of experiments for the Ball distribution with $K=2$; the number of maxima grew as \sqrt{N} and the number of comparisons per point grew as $\log N$, or slower. We do not contend that the Ball distribution models inputs for any particular application, but these experiments show that Algorithm M3 is reasonably efficient even when there are many maximal points.

History

Our attention was drawn to the maxima problem by Julio Urgel [1989] in April of 1989. He needed to compute maxima in a program that automates aspects of contract negotiation: N potential contracts (as many as 10,000) are evaluated in various dimensions (K varies between 5 and 10), and dominated points (inferior contracts) can be excluded from later (expensive) processing. He called one of the authors regarding Bentley's [1980] description of the multidimensional divide-and-conquer maxima algorithm. He was interested in obtaining software that implemented the algorithm; we told him that we knew of no implementation. During our discussion, he stated that he was interested in solving problems of size up to $N=10,000$ with K between 5 and 10. He added that the straightforward $O(KN^2)$ algorithm was too slow on his machine, and that the data appeared to be roughly modeled by a CI distribution. He desired an algorithm that was simple to code and reasonably efficient on his input sets; he was not concerned with worst-case behavior.

That afternoon we designed Algorithm M3 and implemented it in about 100 lines of C++ code. Besides the algorithm itself, the code included routines for generating random input data, gathering and printing statistics, and providing a simple animation of the algorithm. The animation helped us to understand the algorithm's behavior on planar inputs. Later that evening we conducted simple experiments that indicated that Algorithm M3 might have linear running time, and communicated the algorithm to Urgel.

Over the next couple of days we tried to prove that Algorithm M3 had linear running time, but we could make no progress. We returned to perform more data analysis (roughly as presented in the last subsection), and eventually conjectured that Algorithm M3 uses $N + o(N)$ point comparisons. With that conjecture and some insight from watching algorithm animations, we derived and analyzed Algorithms M1 and M2.

The paradigm of "divide-and-conquer for linear expected time" was originally developed by Bentley, Kung, Schkolnick and Thompson [1978] for the maxima problem. Bentley and Shamos [1978] then extended the paradigm to several other problems, including convex hulls. By conscious analogy to that effort, we attempted to apply the idea of a quick certificate of exclusion to computing convex hulls. That is the topic of the next section.

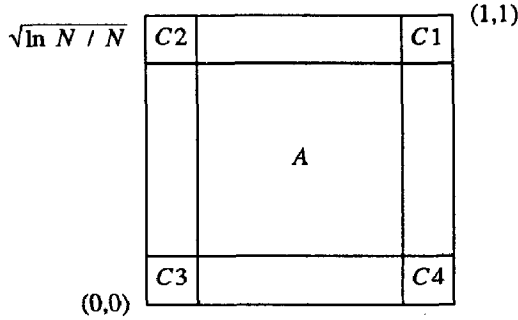
3. Convex Hulls

The convex hull of the point set S is the smallest convex set containing the points in S . Preparata and Shamos [1985] describe in detail the history of algorithms for computing hulls, which we will now sketch briefly. Graham [1972] shows that the hull of N planar points can be computed in $O(N \log N)$ worst-case time; Preparata and Hong [1977] extend that time bound to $K=3$ -space. Bentley and Shamos [1978] showed that the hull can be computed in $O(N)$ time for $K=2$ and $K=3$ for a number of input distributions, including all CI distributions. Golin and Sedgewick [1988] describe a sieve algorithm for computing a superset of the extreme points of the hull that has size $O(\sqrt{N})$ in $O(N)$ time, if the points are chosen uniformly from a rectilinearly oriented hypercube. Their algorithm is simple to code and is particularly efficient.

Convex hulls of K -dimensional point sets become more expensive to compute when $K \geq 4$. Seidel [1981] shows that $O(N^{\lfloor (K+1)/2 \rfloor})$ worst-case time suffices for any $K \geq 3$; his algorithm is optimal in the potential size of the output for even K . Clarkson and Shor [1989] describe a Las Vegas algorithm optimal for both even and odd K . Bentley, Kung, Schkolnick and Thompson [1978] observed that every convex hull point is maximal under at least one of the 2^K possible assignments of K plus and minus signs to the K components of the input points. They observed that by computing the maxima under all 2^K assignments, $O(N)$ time suffices to find a superset of the extreme points with expected size $O(\log^{K-1} N)$, for any fixed K . (Thus the algorithms in Section 2 allow the superset to be computed in $2^K N + o(N)$ point comparisons, for fixed K .) Golin and Sedgewick [1988] extend their planar algorithm to K -space with a coefficient in the leading term of $2^K N$; we will sketch their method shortly.

We will now describe two certificate-based convex hull algorithms, H1 and H2, which correspond to Algorithms M1 and M2 in Section 2. We will start with the planar version of Algorithm H1, which assumes that its input consists of N points uniform over the unit square. It uses a square A whose sides are distance $\sqrt{\ln N / N}$ from

the sides of the unit square; square A defines the four corner squares $C1$, $C2$, $C3$ and $C4$. These objects are shown in this picture:



Just as point P in algorithm M1 certifies that most points are not maxima, so square A will certify that most points are not on the boundary of the convex hull. Those points can be sieved out of later processing. With very high probability, there is at least one point in each of $C1$, $C2$, $C3$ and $C4$. If each of those squares is occupied, then we know that square A is properly contained in the convex hull of S , because square A is within the hull of those four points. Thus all points in A can be ignored in further processing.[†] The details of this approach are given as Algorithm H1.

We will now analyze the run time of Algorithm H1. Step 1 requires constant time, and Step 2 requires at most $4N$ scalar comparisons. We must now compute the probability that no points fall in any of the corner squares. We will first consider square $C1$; the proof of Theorem 2-1 showed that the probability that it is empty is at most $1/N$.

1. Initialize the set B to empty. Let A denote the square centered within the unit square with edge length $(1 - 2\sqrt{\ln N / N})$.
2. Test each point in the set S for inclusion in A . If it is not contained in A , then insert it in set B .
3. Initialize the counts $C[1..4]$ to zero. Examine each point in the set B ; if both of its components lie outside both ranges of A , increment the appropriate $C[i]$ counter. If any of the $C[i]$ counters is zero, then compute the convex hull of set S and return.
4. Because the counters are all nonzero, no extreme points are in set A . Compute the hull of B .

Algorithm H1. Finding the convex hull on the unit square.

[†] A similar result is implicit in the Voronoi diagram algorithm of Bentley, Weide and Yao [1980]. In Step 2 in Section 3, they show that with probability $8N^{1-c}$ is N , no point further than $\lg N/N$ from the boundary of the unit square is on the convex hull.

We turn next to square $C2$, with the complication that we now know that $C1$ is not empty. Fortunately, we know only that $C1$ contains at least one point, so the probability that none of the remaining $N-1$ points is in $C2$ is $(1 - \ln N / N)^{N-1}$, which is bounded above by $1/[N(1 - \ln N / N)] = 1/[(N - \ln N)]$. Set $C3$ is similar, and the probability that $C4$ is empty given that $C1$, $C2$ and $C3$ are not is bounded above by $1/[N(1 - \ln N / N)^3]$. Thus the probability that any one of the four squares is empty is bounded above by the sum of these four values, which is in turn bounded above by $4/[N(1 - \ln N / N)^3]$, which is $O(1/N)$.

If one of the $C[i]$ counters is zero in Step 3 of Algorithm H1, then we use an $O(N \log N)$ worst-case algorithm to compute the hull; because the probability of this event is bounded above by $O(1/N)$, the contribution to the expected time of the algorithm is $O(\log N)$. If Step 4 is executed, then the expected size of the set B is bounded above by N times the area outside of A (plus four for the points known to be in each corner), or $4 + 4N\sqrt{\ln N / N} = O(\sqrt{N \ln N})$. The structure of the subset B has the N^P property of Bentley and Shamos [1978], so we can apply their linear-time hull algorithm to the set B . The expected running time of Step 4 of the algorithm is therefore $O(\sqrt{N \log N})$. The run time of the algorithm is summarized in this theorem.

Theorem 3-1. Algorithm H1 finds the convex hull of N points chosen uniformly from the unit square in $O(N)$ expected time, using $4N + O(\sqrt{N \log N})$ expected scalar comparisons.

It is interesting to compare Algorithm H1 to a similar hull algorithm of Golin and Sedgewick [1988], which we shall refer to as Algorithm GS. Algorithm H1 uses $4N$ scalar comparisons to sieve out all but $O(\sqrt{N \log N})$ points from consideration. Algorithm GS uses more computation to produce a smaller set: the sieve stage uses $8N$ scalar comparisons and $4N$ arithmetic operations to produce a subset of size $O(\sqrt{N})$. In K -space, the sieve in Algorithm GS uses $2^{K+1}N$ scalar comparisons and $2^K(K-1)N$ arithmetic operations (or $2^K(K+1)N$ total primitive operations) to produce a subset of size $O(N^{1-1/K})$. The analysis of algorithm GS assumes that the input point set is chosen uniformly over a rectilinearly oriented hyperrectangle.

We will now extend Algorithm H1 to K -space, for input uniform over the K -dimensional hypercube. The algorithm uses a hypercube A centered in the unit hypercube with edge length $1 - 2(\ln N / N)^{1/K}$. In $2K$ scalar comparisons, all points within A can be discarded. With probability $1 - O(1/N)$, there is at least one point in each of the 2^K corners defined by A , so no points in A are extreme points. The expected number of remaining points (conditional on the fact that the corner squares are not empty) is bounded above by $2^K + 2KN(\ln N / N)^{1/K}$, or $O(N^{1-1/K} \ln^{1/K} N)$ for fixed K .

For the case that $K=3$, we can use the linear expected-time algorithm to compute the convex hull of the small subset; when one of the corner squares is empty, we

-
1. Initialize the set B to empty. Compute the $N((\ln N / N)^{1/K})$ -th and $N(1 - (\ln N / N)^{1/K})$ -th largest element in each dimension; construct the hyperrectangle A to consist of those values in the appropriate dimensions.
 2. Test each point in the set S for inclusion in A . If it is not contained in A , then insert it in set B .
 3. Initialize the counts $C[i]$ to zero. Examine each point in the set B ; if all of its components lie outside all ranges of A , increment the appropriate $C[i]$ counter. If any of the $C[i]$ counters is zero, then process set S and return.
 4. Because the counters are all nonzero, no extreme points are in set A . Process set B .

Algorithm H2. Finding a convex hull superset for a CI set.

use an algorithm with $O(N \log N)$ worst-case running time. Thus we have:

Theorem 3-2. Algorithm H1 finds the convex hull of N points chosen uniformly from the cube in $O(N)$ expected time, using $6N + O(N^{1-1/3} \log^{1/3} N)$ expected scalar comparisons.

For the case that $K \geq 4$, we must settle for computing a small superset of the extreme points. Even though the conditioning of the algorithm destroys the CI property, the inputs retain the N^P property because they can be partitioned into $O(3^K)$ hyperrectangles, each of which is CI.

Theorem 3-3. For fixed $K \geq 4$, Algorithm H1 finds a superset of the extreme points of the convex hull of N points chosen uniformly from the K -dimensional hypercube in $O(N)$ expected time, using $2KN + O(N^{1-1/K} \log^{1/K} N)$ expected scalar comparisons. The expected size of the superset is $O(\log^{K-1} N)$.

We turn next to Algorithm H2, which corresponds to Algorithm M2 in the last section. Its input is a set of N points from a K -dimensional CI distribution. The details of this approach are given as Algorithm H2. Its analysis combines the techniques used in the analysis of Algorithm H1 and Algorithm M2. The result is the following theorem.

Theorem 3-4. Algorithm H2 processes N points chosen uniformly from the K -dimensional CI distribution in $O(N)$ expected time, using $2KN + O(N^{1-1/K} \log^{1/K} N)$ expected scalar comparisons. For $K=2$ and $K=3$, it computes the convex hull of the point set. For $K \geq 4$, it computes a superset of the extreme points whose expected size is $O(\log^{K-1} N)$.

Algorithm H2 represents a substantial improvement over the sieve time of Golin and Sedgewick's [1988] algorithm,

which uses $2^{K+1}KN$ scalar comparisons. The expected number of remaining points is only slightly higher than the $O(N^{1-1/K})$ of Algorithm GS.

Like a maxima algorithm, a correct convex hull algorithm must necessarily inspect all K components of all points within the hull. Algorithm H2 is therefore within a factor of two of optimal in the leading term. Tightening the bound will require a precise model of computation.

4. Conclusions

Rectangles have long been used as certificates in computational geometry. The bounding box of a polygon, for instance, is the smallest rectilinearly oriented rectangle that contains the polygon. If a point is outside a polygon's bounding box, that certifies that the point is not contained in the polygon. If two polygons have bounding boxes that do not intersect, that certifies that the polygons do not intersect. The certificates in this paper are, however, fundamentally different: they are known to be valid only after a significant amount of processing. If they are found to be invalid, then a standard algorithm is called.

The certificate-based sieve algorithms of this paper yield efficient expected-time algorithms for boundary problems. A certifying point is able to show that most points are not maximal, and a certifying hyperrectangle is able to show that most points are not on the boundary of the convex hull. All those points may be removed by a sieving step. These algorithms are applicable to all distributions with Component Independence. The expected run time of the maxima Algorithm M2 is optimal in the leading term; Algorithm M3 is simple to implement and we conjecture that it is also optimal in the leading term. The expected run time of convex hull Algorithm H2 is within a factor of two of optimal, and its leading coefficient of $2KN$ is a substantial improvement over the previous best result of $2^{K+1}KN$. The history at the end of Section 2 exhibits interesting interactions among consulting, algorithm design, data analysis, and mathematical analysis of algorithms.

We conclude by mentioning several problems that merit further research.

Prove Conjecture 2-5, which raises the lower bound on the run time of maxima algorithms. Is it possible to tighten the lower bound and the upper bound to match in second-order terms?

Prove Conjecture 2-6, which states the expected number of point comparisons used by maxima Algorithm M3 is $N + o(N)$.

Design a simple convex hull algorithm corresponding to maxima Algorithm M3.

Design an algorithm that finds the convex hull of a CI set in $O(N)$ expected time.

Extend certificate techniques to other problems. Consider, for instance, the problem of intersecting half planes under the probabilistic model used by Bentley and Shamos [1978]: a small circle centered at the ori-

gin certifies that half planes that do not intersect it are, with high probability, not in the final polygon.

Acknowledgments

We are grateful for the helpful comments of Rick Becker, Brian Kernighan, Julio Urgel, and Chris Van Wyk.

References

- Becker, R. A., L. Denby, R. McGill and A. R. Wilks [1987]. "Analysis of data from the Places Rated Almanac", *The American Statistician* 41, 3, August 1987, pp. 169-186.
- Bentley, J. L. [1980]. "Multidimensional divide-and-conquer", *Communications of the Association for Computing Machinery* 23, 4, April 1980, pp. 214-229.
- Bentley, J. L., H. T. Kung, M. Schkolnick and C. D. Thompson [1978]. "On the average number of maxima in a set of vectors and applications", *Journal of the Association for Computing Machinery* 25, 4, October 1978, pp. 536-543.
- Bentley, J. L. and M. I. Shamos [1978]. "Divide and conquer for linear expected time", *Information Processing Letters* 7, 2, February 1978, pp. 87-91.
- Bentley, J. L., B. W. Weide and A. C. Yao [1980]. "Optimal expected-time algorithms for closest point problems", *ACM Transactions on Mathematical Software* 6, 4, pp. 563-580.
- Boyer, R. and D. Savageau [1985]. *Places Rated Almanac* (Revised Edition), Rand McNally.
- Clarkson, K. L. and P. W. Shor [1989]. "Applications of random sampling in computational geometry, II", to appear in *Discrete and Computational Geometry*.
- Floyd, R. W. and R. L. Rivest [1975]. "Expected time bounds for selection", *Communications of the ACM* 18, 3, March 1975, pp. 165-172.
- Golin, M. and R. Sedgewick [1988]. "Analysis of a simple yet efficient convex hull algorithm", *Proceedings of the Fourth Annual Symposium on Computational Geometry*, pp. 153-163.
- Graham, R. L. [1972]. "An efficient algorithm for determining the convex hull of a finite planar point set", *Information Processing Letters* 1, pp. 132-133.
- Kung, H. T., F. Luccio and F. P. Preparata [1975]. "On finding the maxima of a set of vectors", *Journal of the Association for Computing Machinery* 22, 4, October 1975, pp. 469-476.
- Monier, L. [1980]. "Combinatorial solutions of multidimensional divide-and-conquer recurrences", *Journal of Algorithms* 1, 1, March 1980, pp. 60-74.
- Preparata, F. P. and S. J. Hong [1977]. "Convex hulls of finite sets of points in two and three dimensions", *Communications of the Association for Computing Machinery* 20, 2, February 1977, pp. 87-93.
- Preparata, F. P. and M. I. Shamos [1985]. *Computational Geometry*, Springer-Verlag.
- Seidel, R. [1981]. "A convex hull algorithm optimal for points in even dimensions", M.S. Thesis, Technical Report 81-14, Department of Computer Science, University of British Columbia, Vancouver, Canada.
- Urgel, J. [1989]. Private communication of Julio Urgel, Harvard Business School, April-May 1989.