DIMACS Technical Report 94-22
January 1995

Randomizing Optimal Geometric Algorithms

by

Larry Shafer[1]
Dept. of Computer Science
Rutgers University
New Brunswick, New Jersey 08903

William Steiger[2,3]
Dept. of Computer Science
Rutgers University
New Brunswick, New Jersey 08544

[1]Research Supported in Part by a Research Experiences for Undergraduates (REU) supplement to NSF grant CCR-9111491.
[2]Permanent Member
[3]Research Supported in Part by NSF grant CCR-9111491.

DIMACS is a cooperative project of Rutgers University, Princeton University, AT&T Bell Laboratories and Bellcore.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

The parametric search technique of Megiddo has been used to produce optimal deterministic algorithms for a number of geometric problems, e.g., planar ham-sandwich cuts and slope selection. Unfortunately these algorithms inherit huge constants from the parallel algorithms used by the technique. Here we give very simple randomized versions and show (experimentally and analytically) that the constants in the asymptotically optimal running times are small. These randomized versions could therefore be argued to be the methods of choice for their respective tasks. We also give a simple, new, randomized planar convex hull algorithm whose expected running time matches the average-case complexity of good deterministic algorithms, and with good protection (probability $\rightarrow 0$ quickly) against worst-case behaviour.

1 Introduction and Summary

One of the main advantages of probabilistic algorithms is that they can almost always avoid worst case behavior exhibited by deterministic algorithms that perform the same task. Random-Quicksort, which partitions with a randomly chosen element, is perhaps, the most familiar example. In fact the best known algorithms for some problems are probabilistic, the most dramatic example being primality testing (it is in random P but there is no known polynomial-time deterministic algorithm). The problems of computing the diameter of a point set in three dimensions and of finding the median of n numbers are two other examples where the best known algorithms are probabilistic.

In this paper we highlight another way in which probabilistic algorithms may be useful - by dramatically reducing the constants in *optimal deterministic* algorithms. This possibility arises because some of the techniques used in these algorithms - namely parametric search and ε-approximation - can force very large constants in the optimal asymptotic running times. Throughout we are working in a model of computation where each binary comparison and each arithmetic operation is taken as a "step"; i.e., the unit cost RAM. As an alternative to an optimal deterministic algorithm with a large constant, we offer probabilistic algorithms whose expected complexities have the correct asymptotic order, and also achieve this order with reasonable constants. We also give a randomized planar convex hull algorithm whose expected running time matches the average-case complexity of good deterministic algorithms, and with good protection (probability $\rightarrow 0$ quickly) against worst-case behaviour.

In the remainder of the introduction we describe the problems for which we have devised simple randomized algorithms. We state the properties we can prove and some that we have observed in computational experiments. Sections 2-4 contain more details concerning the algorithms and proofs of the theorems. However the space constraints force us to leave many details for the full paper.

1.1 Slope Selection

Given n points in general position in the plane and a positive integer k, slope selection is the problem of finding that pair of points which determine the line with the k^{th} smallest slope, amongst all the $\binom{n}{2}$ lines determined by the points. Cole et. al. [2] described an optimal $O(n \log n)$ algorithm that depends crucially on the parametric search technique of Megiddo [14]. Parametric search features the use of parallel algorithms in designing efficient serial ones. But the parallel algorithms can pass on enormous constants to the serial ones that they help produce. This is precisely what happens in [2]. The first result of this paper is a new, simple, randomized slope selection algorithm in which randomization allows us to bypass the parallel sorting network that was used in the deterministic algorithm. The new algorithm, *Rand-Slope*, is described in Section 2. We are able to show that it is a Las Vegas algorithm and we describe its complexity.

Theorem 1 *Given n points in the plane, Rand-slope selects the line with the k^{th} smallest slope from the $\binom{n}{2}$ lines determined by the points. With probability $\rightarrow 1$, as $n \rightarrow \infty$, the*

complexity is $O(n \log n)$ steps.

This compares favorably with an earlier randomized algorithm of Matoušek [12] which he proved to have expected running time $O(n \log n)$. A more careful analysis of that algorithm, borne out as well by studying an actual implementation where the RAM operations were counted over a variety of inputs, reveals that the constant hidden in $O(n \log n)$ is more than 20. The present algorithm is simpler. It admits a further simplification, *Rand-Slope-2*, which behaves very well in practice; extensive testing on many random inputs suggest that it has an implied constant of about 10. Thus, we observe empirically that the complexity is $< 10n \log n$ with high probability.

A basic ingredient of all our algorithms is a fast, randomized selection routine based on the one presented by Floyd and Rivest [5],[6]. Their method can find the median of n numbers with $1.5n + o(n)$ expected comparisons. Analysis and experiments show that if this method is not implemented carefully, the cost is likely to be $> 2n$ even for moderately large values of n. We can tune parameters that govern sample size and the way that recursion is managed so that with high probability, the cost is very close to $1.5n$. These results will be discussed in the full paper.

1.2 Planar Ham Sandwich Cuts

Given sets $A = \{a_1, \ldots, a_r\}$ and $B = \{b_1, \ldots, b_s\}$ of points in the plane, $n = r + s$, a *ham-sandwich cut* is a line h with the property that at most half of the points in A lie in either of the open halfspaces defined by h and the same for the points in B. The ham sandwich theorem guarantees the existence of such a line and we consider the algorithmic question of finding a cut. Lo and Steiger [9] (see also [10] and [11]) described an $O(n)$ algorithm which settled the complexity question for the plane. However their optimal time algorithm inherits very large constants because it relied on the technique of ε-approximation. In Section 3 we present a simple, randomized version, *Rand-ham*, which allows us to bypass the parallel sorting network step (or other direct ways to achieve the ε-approximation). It is Monte-Carlo and we can prove that it has good complexity with high probability.

Theorem 2 *Given sets $A = \{a_1, \ldots, a_r\}$ and $B = \{b_1, \ldots, b_s\}$ of points in the plane, $n = r + s$, with high probability the algorithm Rand-ham will find a ham sandwich cut in at most $80n$ steps.*

It is true that the optimal deterministic algorithm is also quite simple. However the constant appears to be very large, probably more than $20,000$.

1.3 Planar Convex Hulls

Given a set $S = \{p_1, \ldots, p_n\}$ of n points in the plane, we seek $CH(S)$, the convex hull. One of the simplest and most attractive algorithms is the *Quickhull* algorithm of Eddy [3] (see related algorithms of Green and Silverman [7], and Bykat [1]). On random point configurations, Green and Silverman found Quickhull to have the best average-case performance among

several convex hull algorithms. Overmars and Van Leeuwen[15] showed that the average-case cost of Quickhull is linear when points are chosen randomly from a uniform distribution over some bounded convex region.

Quickhull finds a hull vertex and uses it to partition the input set. In the partitioning, many points of S may be eliminated. Starting with L and R, the points of S with min and max x-coordinate, partition S using $\ell = \overline{LR}$ into sets A, the points *above* ℓ and B, the points *below* ℓ. During this step we discover $\alpha \in A$, the point of maximal distance above ℓ and $\beta \in B$, the point of maximum distance below ℓ. Focusing on A, the partition point α is a hull vertex and all points of A below both $\overline{L\alpha}$ and $\overline{\alpha R}$ may be discarded. This pruning is what makes Quickhull so fast. However if the points are in convex position, then like Quicksort, Quickhull can perform $O(n^2)$ steps if the partitioning is not balanced; the number of hull vertices between L and α should be about the same as the number between α and R. It is easy to construct a set S_n of n points where Quickhull *will always* take $O(n^2)$ steps.

Randomization can be used to provide good protection against such worst case behaviour while retaining good performance on average. Our Las Vegas algorithms choose a partitioning hull vertex at random (instead of the point of maximum or minimum distance from ℓ). We studied several methods of making this random choice. We dont know yet if one is best, so we combine them in an algorithm *Rand-Hull* which does several methods at once, alternately, and halts when the fastest one terminates.

Theorem 3 *Given a set S of n points in the plane, let T_n denote the complexity of Rand-Hull. Then $P[T_n = O(n \log h)] \to 1$ as $n \to \infty$, where $h = |CH(S)|$*

The theorem guarantees protection against worst case performance, even on the sets like S_n where Quickhull is quadratic. Note also that the theorem only gives an upper bound on T_n. Simulation experiments using uniformly distributed points from regions of positive area in the plane show that *Rand-Hull* has the same average-case behaviour as Quickhull.

2 Slope Selection

Cole, Salowe, Steiger, and Szemerédi [2] gave an extremely complex $O(n \log n)$ optimal deterministic algorithm for the slope selection problem. A much simpler randomized algorithm was developed by Matoušek [12]; it has expected running time $O(n \log n)$. Extensive testing of a tuned version of Matoušek's algorithm shows that in fact the complexity is $< 21n \log n$ (We tested several values of n. For each value we generated 25 sets of n points, computed the median slope line [determined by pairs of points] in each set, and averaged the number of RAM steps over the 25 repetitions).

Here we present a new, simpler algorithm, *Rand-slope*. We can prove that it always solves the problem and that it terminates in $O(n \log n)$ steps with probability that converges to 1 as $n \to \infty$ (Theorem 1). We also present a modifcation, *Rand-slope-2*, which in the same empirical testing, solves the slope selection problem in $< 9n \log n$ steps.

As in [2] it is convenient to consider the dual form of the problem: the point (x, y) is mapped to the line with equation $v = xu + y$ and the line with equation $y = mx + b$ is

mapped to the point $(-m, b)$, a transformation that preserves incidence and the above/below relation of points and lines. Therefore in the dual, we are given n lines in general position and an integer k; we seek the vertex v in the line arrangement with the k^{th} smallest x-coordinate. Here is a coarse description of the randomized algorithm. The inputs are arrays $M = \{m_1, \ldots, m_n\}$ and $B = \{b_1, \ldots, b_n\}$ giving the slopes and intercepts of the lines, along with n and k; the output is the desired vertex, v.

- **Rand-Slope(M,B,n,k;v)**

  1. choose $s = n \log n$ distinct pairs $(i, j), i < j$, each chosen according to the uniform distribution.
  2. for each pair compute $x_{ij} = (b_j - b_i)/(m_i - m_j)$, the x-coordinate of the intersection of the i^{th} and j^{th} lines. Let $u_1 < \cdots < u_s$ denote these x-coordinates, in order.
  3. use binary search with *approximate ranking* to find u_ℓ such that the k^{th} vertex is in $I = [u_\ell, u_{\ell+1}]$.
  4. Use Bentley-Ottman line sweep on I to find v.

- **End**

Some of the steps need explanation. Let $t_1 < \cdots < t_N$ be the sorted x-coordinates of the $N = \binom{n}{2}$ vertices of the arrangement. Recall that the rank of a vertical line $x = t$ in an arrangement of n lines is $|\{t_i : t_i \leq t\}|$, the number of vertices to its left. This may be computed in $O(n \log n)$ steps by counting the number of inversions between the permutation that sorts the slopes of the lines and the permutation that sorts their intercepts with $x = t$.

Step 1: To get one pair, choose i uniformly from $(1, 2, \ldots, n)$ and j uniformly from $(1, 2, \ldots, n) \backslash i$. It is easy to get s *distinct* pairs in $O(s)$ steps.

Step 3: From Step 2 we have the x-coordinates of the s vertices in our random sample; call them $S = \{u_1, \ldots, u_s\}$. To start the binary search we compute $\mu = \text{median}(S)$. Next we use the approximate ranking of Cole, et.al. [2] to decide whether the desired vertex v is to the left, to the right, or on, the line $x = \mu$. If there are many vertices between v and $x = \mu$ this procedure will terminate in time $O(n)$; if v is "close" to $x = \mu$ it may require $O(n \log n)$ steps. Take the case $v > \mu$, the other being similar. We continue, as above, by searching for the largest vertex in $S' = \{u_i : u_i > \mu\}$ which is smaller than v. This vertex is t_ℓ, and we get $t_{\ell+1}$ in similar fashion.

Step 4: We rank $x = t_\ell$ (it is some number $j < k$) and then sweep $k - j$ vertices to the right.

Proof of Theorem 1: (sketch) The amortized analysis of Step 3 is the only delicate part. With probability $\to 1$ as $n \to \infty$ all but a constant number of the $O(\log s)$ binary search steps may be decided by approximate ranking in $O(n)$ steps. The total cost of refining the approximations during the binary search is $O(n \log n)$.

The complexity of Step 4 is bounded by $O(|\{i : t_i \in I\}| \log n)$. Obviously the expected value of $|\{i : t_i \in I\}|$ is $\binom{n}{2}/s = O(n/\log n)$. An elementary probability lemma (omitted

now) states that with probability $\to 1$ *no* interval $I = (t_\ell, t_{\ell+1})$ of successive sample vertices contains more than cn vertices of the arrangement. Therefore Step 4 is also $O(n \log n)$ with high probability. ∎

The simplification effected by *Rand-Slope-2* is in Step 3. Instead, we use

- Step 3'. Let $j = sk/\binom{n}{2}$ and select μ, the j^{th} smallest x_{ij} . Rank μ and sweep *left* if the rank is greater than k, *right* otherwise.

For $n < 200$ this algorithm always solved the slope selection problem in $< 9n \log n$ steps.

3 Planar Ham-Sandwich Cuts

As in the previous section, it is convenient to transform the problem to a dual setting. Under this transformation we have sets $A = \{a_1, \ldots, a_r\}$ and $B = \{b_1, \ldots, b_s\}$ of lines in general position in the plane, $n = r + s$. The dual of a *ham-sandwich cut* is a point μ with the property that at most half the A lines are above or below it and the same for the B lines. Thus μ is an intersection of the median levels of the A and B lines (recall that the p^{th} level in a line arrangement is the locus of points which (i) are in some line of the arrangement and (ii) have exactly $p-1$ lines above). Here is a coarse description of our Las Vegas algorithm *Rand-Ham*. The inputs are the arrays M, B giving the slope and intercept of the n lines, r, the number of A lines, and n; the output is the ham sandwich vertex v.

- **Rand-Ham(M,B,r,n;v)**

  1. choose a sample of k distinct A lines at random, each chosen according to the uniform distribution.
  2. for each pair $i \neq j$ in the sample compute $x_{ij} = (b_j - b_i)/(m_i - m_j)$, the x-coordinate of the intersection of the i^{th} and j^{th} A lines. Let $u_1 < \cdots < u_K$ denote these x-coordinates, in order, $K = \binom{k}{2}$.
  3. use binary search on the x_{ij} to find the u_ℓ such that $v \in I = [u_\ell, u_{\ell+1}]$.
  4. As in [11] construct a trapezoid τ with vertical sides on $x = t_\ell$ and on $x = t_{\ell+1}$, discard *all* lines that miss τ, and continue recursively on I.

- **End**

Steps 1 and 2 are self-evident. They replace the costly construction of the epsilon approximation used in [9]. With high probability, the sample of A lines will be an ε-approximation for the A lines (with respect to segments) if $k > C\varepsilon^{-2} \log \varepsilon^{-1}$.

Some of the other steps need elucidation: In Step 0, we find μ_A and μ_B, the median slopes of the A and B lines.

Step 3: From Step 2 we have the x-coordinates of the $K = \binom{k}{2}$ vertices in our random sample; call them $S = \{u_1, \ldots, u_K\}$. To start the binary search we compute $\mu = \text{median}(S)$.

If the median levels of the A and B lines at $x = \mu$ are ordered in the same way as μ_A and μ_B, we know that $v > \mu$, and we continue searching in the $S = \{u_i : u_i > \mu\}$.

Step 4: As in [11], at both $x = t_\ell$ and $x = t_{\ell+1}$, we compute the $(1/2 - \varepsilon)r$-th and $(1/2 + \varepsilon)r$-th smallest intercepts of A lines. These four points determine τ.

Proof of Theorem 2: (sketch) The algorithm finds a cut with high probability. It was proved in [11] that if the sample is an ε-approximation (i) at least half the A lines will "miss" τ and (ii) the ham-sandwich vertex will be contained in τ. This occurs with high probability as long as $k > C\varepsilon^{-2}\log\varepsilon^{-1}$.

The time bound is probably conservative. We can take $k = 300$ and $\varepsilon = 1/7$ in Step 1, so $K < 50000$. Each binary search step is two selections, $1.5n$ RAM steps each, for a total of at most $54n$. In Step 4 the lines can be tested in time $2n$. Each phase of the recursion eliminates at least a quarter of all remaining lines and summing the geometric series that bounds the complexity gives the time bound of $< 76n$.∎

4 Randomized Quickhull

The Quickhull algorithm has inputs L and R, points with respective x-coordinates $L_x < R_x$, and a set S of points above the line $\overline{LR}$ whose x-coordinates are between L_x and R_x. The output is CH, the upper hull of $S \cup L \cup R$.

- **Quickhull(L,R,S;CH)**
  1. If $|S| > 1$ find a point $P \in S$ whose (vertical) distance above $\overline{LR}$ is maximal.
  2. Find S_{left}, the points in S above $\overline{LP}$ whose x-coordinates are between L_x and P_x and S_{right}, the points above $\overline{PR}$ with x-coordinates between P_x and R_x.
  3. Append P to CH and continue with subproblems Quickhull$(L, P, S_{left}; CH)$ and Quickhull$(P, R, S_{right}; CH)$.
- **End**

In $|S| = 1$ in Step 1, that point joins CH and we stop. If S is empty we just stop. The cost of Steps 1 and 2 is $O(|S|)$.

The following set $S_n = \{P_1, \ldots, P_n\}$ causes Step 1 to choose $P_2, \ldots, P_{n-1}$ in sequence so the algorithm will perform $O(n^2)$ RAM steps. $P_1 = (1, 0)$, $P_2 = (2, 1)$, $P_n = (n, 0)$, and define s_i as the slope of the line through P_{i-1} and P_n and t_i as the slope of the line through P_i and P_n. For $i = 2, \ldots, n-2$ $P_{i+1} = (i + 1, y_{i+1})$, and $y_{i+1} = y_i + (s_i + t_i)/2$.

Rand-hull chooses a random hull vertex in Step 1. One way to do this is to choose a pair $P_i \neq P_j$ in S, and then find the supporting line ℓ whose slope is the same as that of $\overline{P_iP_j}$ and with no point of S above it. A point $P \in S \cap \ell$ is a random hull vertex if both L and R are below ℓ. If one is above, say L, take P to be the point in S so all points are below $\overline{LP}$.

If S is convex, P behaves like the partition element in random quicksort: Suppose there are h vertices of CH with x-coordinates between L_x and R_x. The vertex P chosen in Step 1

is "good" if at least $h/5$ of these vertices have smaller x-coordinate than P_x and at least $h/5$ have bigger x-coordinate. The probability that a good vertex is not chosen in, say 5 levels of recursion, is small. This gives $O(\log |CH|)$ as the expected depth of recursion. The proof of Theorem 3 rests on establishing these statements.

We ran Quickhull and 10 repetitions of *Rand-hull* on the set S_n above, for several values of n. In this experiment the average complexity of *Rand-hull* was $12n \log n$ RAM steps while Quickhull averaged $6.8n^2$ RAM steps.

Finally we note that the bound of Theorem 3 matches the optimal worst-case deterministic bound in [8]. However *Rand-hull* is very different from a randomized version of that algorithm and also differs from a modification described by McQueen and Toussaint [13].

References

[1] A. Bykat. Convex hull of a finite set of points in two dimensions. *Information Processing Letters 7*, 296-298, 1978.

[2] R. Cole, J. Salowe, W. Steiger, and E. Szemeredi. An optimal-time algorithm for slope selection. *SIAM J. Comput. 18*, 4, 792-810, 1989.

[3] W. Eddy. A new convex hull algorithm for planar sets, *ACM Trans. Math. Software 3* 398-403 & 411-412, 1977.

[4] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Berlin, 1987.

[5] R. Floyd and R. Rivest. Expected Time Bounds for Selection. *Comm. of the ACM 8*, 165-172, 1975.

[6] R. Floyd and R. Rivest. Algorithm 489, The algorithm $SELECT$ for finding the ith smallest of n elements. *Comm. of the ACM 8*, 173, 1975.

[7] P.J. Green and B.W. Silverman. Constructing the convex hull of a set of points in the plane, *Computer Journal 22*, 262-266, 1979.

[8] D. Kirkpatrick and R. Seidel. The Ultimate Planar Convex Hull Algorithm? *SIAM J. of Computing, 15*, 287-299, 1986.

[9] Chi-Yuan Lo and W. Steiger. An Optimal-Time Algorithm for Ham Sandwich Cuts in the Plane. *Second Canadian Conference on Computational Geometry*, 5-9 (1990).

[10] Chi-Yuan Lo, J. Matoušek, and W. Steiger. Ham-sandwich cuts in R^d. *Proc. 24th ACM Symposium on Theory of Computing*, 539-545, 1992.

[11] Chi-Yuan Lo, J. Matoušek, and W. Steiger. Algorithms for Ham-sandwich cuts. *Discrete and Computational Geometry* (to appear).

[12] J. Matoušek. Randomized optimal algorithm for slope selection. *Information Processing Letters, 39*, 183-187, 1991.

[13] M. McQueen and G. Toussaint. On the ultimate convex hull algorithm in practice, *Pattern Recognition Letters 3*, 29-34, 1985.

[14] N. Megiddo. Applying Parallel Computation Algorithms in the Design of Serial Algorithms. *J. of the ACM 30*, 852-865, 1983.

[15] M. Overmars and J. van Leeuwen. Further comments on Bykat's convex hull algorithm, *Information Processing Letters 10*, 209-212, 1980.