Enumerating k Longest Distances for n Points in the Plane

Matthew T. Dickerson*
Jason Shugart
Middlebury College

Abstract

In this paper we present a deterministic algorithm which takes as input a set of n points in the plane and enumerates in descending order the k longest distances between pairs of points. The algorithm requires $O(n+k)$ space, and for uniform distributions has expected case running time of $O(n \log n + min(k \log^2 n, n^2 \log n))$. We also present empirical results showing that for uniform distributions both the expected average and the expected maximum degree of hull vertices in the Delaunay diagram are $\Theta(\log n)$. Finally, we report on actual running times for an implementation of this algorithm.

1 Introduction

In this paper, we examine the following problem:

Problem 1 (Enumerating Longest Distances in the Plane) *Given a finite set S of n distinct points in the plane, let $d_1 \geq \cdots \geq d_{\binom{n}{2}}$ be the distances determined by the pairs of points in S. For a given positive integer $k \leq \binom{n}{2}$, enumerate in descending order k pairs of points that realize $d_1, \ldots, d_k$.*

More simply stated, the problem is to report in descending order the k longest interpoint distances for a set of n points in the plane.

Problem 1 is related to the following problem which has received considerable attention:

Problem 2 (Selecting Distances in the Plane) *Given a finite set S of n distinct points in the plane, let $d_1 \leq \cdots \leq d_{\binom{n}{2}}$ be the distances determined by the pairs of points in S. For a given positive integer $k \leq \binom{n}{2}$, determine the value of d_k and find a pair of points that realizes d_k.*

It is easy to see that a solution to Problem 1 for $k = k_1$ provides a solution to Problem 2 for $k = (\binom{n}{2} - k_1)$.

Problem 2 has recently been investigated by Chazelle [4], and by Agarwal, Aronov, Sharir, and Suri [1]. Chazelle [4] presented a subquadratic solution to Problem 2 based on the batching technique of Yao [14]. The algorithm works in any dimension, and for dimension 2 the running time is $O(n^{9/5} \log^{4/5} n)$. Agarwal, Aronov, Sharir, and Suri [1] have improved that result providing an algorithm which runs in time $O(n^{4/3+\epsilon})$. Dickerson and Drysdale [6] recently presented an $O(n \log n + k \log n)$ time and $O(n+k)$ space algorithm for the problem of *enumerating* the smallest k distances. This also provided a solution to the Problem 2 which is faster than the algorithm of [1] for the case when k is $O(n^{4/3})$. Salowe, who had earlier solved the interdistance selection problem for the L_∞ metric in d-dimensions in $O(dn \log^d n)$ time [11] has recently extended those results to get an $O(n \log n)$ algorithm for the selection problems that works for any L_p metric in arbitrary dimension d, for $k \leq n$ [12]. As an

*This study was supported in part by the funds of the National Science Foundation, NSF RII-8610679.

n	125	250	500	1000	2000	4000	8000	16000	32000
avg	5.614	5.964	6.415	6.669	7.018	7.313	7.628	7.948	8.228
max	8.240	8.780	9.740	10.660	11.640	12.360	13.740	14.230	15.160

Table 1: Expected Degrees of Hull Vertices: U.D. on a Disc

intermediate result, he presented an $O(n \log n + k)$ algorithm for enumerating the smallest k distances (not necessarily in sorted order). However the running time of the algorithms of [6] and [12] are dependent on k which is an undesirable feature for the selection problem; when k is $\Omega(n^{4/3})$, these algorithms are slower than that of [1] for Problem 2.

In this paper we explore Problem 1, enumerating k longest distances for n points in the plane, which is intuitively more difficult than the shallow enumeration problem explored in [6] and [12], and which also provides a solution to the selection problem, Problem 2. Problem 1 has recently been explored by Katoh and Iwano [8] who made use of the k-farthest neighbor Voronoi diagram to provide an algorithm requiring time $O(\min\{n^2, n + (k^{4/3}/(\log k)^{1/3})\} \log n)$, or equivalently time $O(n \log n + (k^{4/3}/(\log k)^{1/3}) \log n)$ with an upper bound of $O(n^2 \log n)$.

We present an algorithm for the solution to Problem 1 which makes use of the standard Delaunay diagram as its basic data structure. The algorithm uses $O(n + k)$ space and has an expected case running time of $O(n \log n + min(k \log^2 n, n^2 \log n))$ for uniform distributions. It is simple to state, is based on a common data structure, and given the Delaunay diagarm it is easily implementable. We have implemented the algorithm and tested it on values of n and k ranging up to $n = 32000$ and $k = 10^8$. After the algorithm has been presented, we mention some empircal results achieved by this implementation. We also present empirical results on the degrees of vertices in the Delaunay diagram.

2 Expected Degrees of Hull Vertices in the Delaunay Diagram

In this section, we present some empirical results on the expected degrees of hull vertices in the Delaunay diagram of points in a uniform distribution.

In Table 1, we give both the average and maximum degrees of convex hull vertices for a set of n points taken from a uniform random distribution on a disc. For each value of n, we averaged the results of 50 different sets.

When compared to a logarithm function of n, the r^2 value for the average degree is 0.997 and the r^2 value for the maximum degree is 0.996. We ran similar tests for uniform distributions of both integer and real points over a square, and all empirical tests supported the following proposition:

Proposition 1 *In a uniform distribution of n points, the expected average and maximum degree of hull vertex in the Delaunay diagram is $\Theta(\log n)$).*

We note that this is not an unexpected result; a similar result for the greedy triangulation was reported by Manacher and Zobrist [9].

3 Enumerating k Longest Distances

We present an $O(n \log n + min(k \log^2 n, n^2 \log n))$ expected time algorithm for Problem 1. We begin with our algorithm, and then give a proof of its correctness and running time.

In Algorithm 1, let Q be a priority queue which supports an **Add** operation and a **DeleteMax** operation. The **DeleteMax** operation deletes from Q the pair (v, w) of maximum distance and returns (v, w). The **Add** (v, w) operation adds a pair (v, w) to Q if the pair is not already in the queue. Here and throughout the paper, we let $d(v, w)$ be the Euclidean (or L_2) distance function. We also use the term *antipodal pairs* in the standard way, to refer to a pair of points in S that admits parallel supporting lines, where a supporting line is a straight line passing through a vertex of S such that all points in S fall on the same side of that line [10].

3.1 Algorithm 1

Step 1: Preprocessing Phase Given a finite set S of n distinct points, construct the Delaunay diagram D of S and the convex hull H of S.

Step 2: Initializing Priority Queue For all antipodal pairs (v, w) with $v, w \in S$, **Add**(v, w).

Step 3: Enumeration

```
    FOR $i := 1$ TO $k$ DO
        a) Let $(v, w) :=$ DeleteMax($Q$).
        b) Report $(v, w)$ with $d_i = d(v, w)$.
        c) $\forall (w, x) \in D$, if $d(v, x) < d(v, w)$ then Add($v, x$).
        d) $\forall (u, v) \in D$, if $d(u, w) < d(v, w)$ then Add($u, w$).
END
```

3.2 Proof of Correctness

We now prove the correctness of Algorithm 1 by induction on k. This means proving that for a value of i at Step 3.b), the algorithm really does report the i^{th} longest distance.

For the base case $i = 1$, we note that the pair of points (v, w) realizing the longest distance d_1 would have been put into the queue in Step 2, since (v, w) is an antipodal pair [15]. In order to prove the inductive case, we need the following theorem.

Theorem 1 *Let S be a set of n points in the plane, D the Delaunay diagram of S, and (v, w) a pair of points which realizes d_i, the i^{th} longest interpoint distance in S. Then one of the following holds:*

a) points v and w are antipodal,
b) there exists a point $u \in S$ such that $(u, v) \in D$ and $d(u, w) > d_i$, or
c) there exists a point $x \in S$ such that $(w, x) \in D$ and $d(v, x) > d_i$.

Proof If v and w are antipodal, then condition (a) is met and we are done.

Assume that condition (a) is not met. Let $\vec{vw}$ be the ray beginning at point v and passing through w, and let L_v and L_w be the parallel lines passing through v and w respectively and perpendicular to $\vec{vw}$. Since v and w are not antipodal, L_v and L_w can not be supporting lines for S. Either there exists some point $x \in S$ which falls on the opposite side of L_w from v, or there exists some point $x \in S$ which falls on the opposite side of L_v from w, or both. Without loss of generality, we assume the former and will show that condition (c) is met. (The corresponding case is analogous.)

Let $x \in S$ be a point on the opposite side of L_w from v. The perpendicular bisector of x and w intersects $\vec{vw}$ on the x side of L_w, and divides the plane into two halfplanes H_x and H_w, with H_x containing x and H_w containing both w and v. Let $V(w)$ be the Voronoi region of w. Every point on H_x is closer to x than to w, and so no point on H_x can fall in $V(w)$. The ray $\vec{vw}$ passes through w, and then out of H_w and into H_x, and so it follows that $\vec{vw}$ passes through $V(w)$ and into the Voronoi region for some other point $x' \in S$. (Possibly though not necessarily $x' = x$). Thus there is a Delaunay edge from w to some point $x' \in S$ with $d(v, x') > d(v, w)$, and condition (c) is met. □

Based on Theorem 1, we can now finish the proof that the algorithm is correct using induction on i. Assume that the algorithm correctly enumerates the $i-1$ longest distances $d_1, \ldots, d_{i-1}$. We need *to show that at step i, a pair realizing d_i has been added to the* priority queue. (Indeed, Theorem 1 states something stronger, namely that *all* pairs realizing distance d_i will have been added by this step.) Let (v, w) be such a pair realizing d_i. If (v, w) is an antipodal pair, then (v, w) was added to the queue in step (2). If (v, w) is not an antipodal pair, then by Theorem 1 for some $j < i$ there exists either a Delaunay edge (u, v) with $d(u, w) = d_j$ or a Delaunay edge (w, x) with $d(v, x) = d_j$. In either case, (v, w) will have been added to the priority queue in Step 3 part (c) or (d).

We have now shown that the algorithm correctly reports the k longest distances for n points in the plane.

3.3 Analysis

We implement our priority queue using a balanced search tree such as a B-Tree, or a 2-3-Tree. The tree contains at most $\binom{n}{2}$ nodes, so our **Add** and **DeleteMax** operations require $O(\log n)$ time.

The preprocessing phase given in Step 1 can be performed in time $O(n \log n)$ for a set S of n points using any of a number of algorithms [13]. For Step 2, the number of antipodal points is $O(h)$ and can be computed in time $O(h)$ where h is the number of points on the convex hull H of S [10]. In the worst case $h = n$ and the step requires a total of $O(n \log n)$ time.

The k **DeleteMax** operations of Step 3.a) require a total of $O(k \log n)$ time. The potentially disastrous parts of Step 3 are substeps c) and d). A single point may have $n-1$ Delaunay edges, and so this step may require $\Theta(n \log n)$ time for a single iteration. If k is $\Theta(n^2)$, this gives us an $O(n^3 \log n)$ time algorithm, which is worse than the naive approach using an optimal sorting algorithm. Fortunately, we can get a much tighter amortized bound. Remember that though a single vertex may have $n-1$ Delaunay edges, there are still only a total of $O(n)$ Delaunay edges. Each edge has two endpoints, and so it may be a part of only $2n-2$ reported distances. We therefore have $O(n)$ edges, each of which will be "examined" in Step 3 parts c) and d) at most $O(n)$ times. In the worst case, therefore, Step 3 requires a total of $O(n^2 \log n)$ time for all iterations, which is no worse than using an optimal sorting algorithm on all n^2 distances.

Our average case time bound can be improved even further. We note that the expected degree of interior points (points sufficiently far from the convex hull) in the Delaunay diagram in a uniform distribution is a constant [5,7]. And Bern, Eppstein and Yao [2] have recently shown that the maximum expected degree of interior points is $\Theta(\log n / \log \log n)$. However since our algorithm is reporting longest distances, we can expect that many of the endpoints will be hull points and not interior points. Proposition 1 states that the average and maximum expected degree of hull points in the Delaunay diagram is $\Theta(\log n)$, and so the total expected number of edge examinations for all iterations of Step 3 is $O(k \log n)$, with each **Add** operation requiring time $O(\log n)$ giving us a total expected running time of $O(n \log n + min(k \log^2 n, n^2 \log n))$.

The Delaunay diagram requires $O(n)$ space to store at most $3n-6$ edges. It appears that the priority queue could potentially grow as big as $\binom{n}{2}$ as we continue to add pairs of points, but we note that we never need to store more than the largest k pairs in the queue. In fact, after iteration i of the algorithm, the largest $k-i$ pairs in the queue will suffice. So for every **Add** operation increasing the size of the queue beyond $k-i$ we may also delete and throw away the smallest distance in the queue. Since deleting the minimum value of a balanced tree can also be performed in $O(\log n)$ time, this does not asymptotically increasing the running time of the algorithm. Thus the priority queue may be stored in $O(k)$ space for a total of $O(n+k)$ space.

	$n = 1000$	$n = 2000$	$n = 4000$	$n = 8000$	$n = 16000$	$n = 32000$
$k = 5000$	1454	1491	1488	1541	1627	1735
$k = 6000$	1761	1807	1808	1874	1970	2097
$k = 7000$	2076	2126	2135	2209	2318	2462
$k = 8000$	2397	2448	2469	2547	2666	2827
$k = 9000$	2723	2777	2805	2890	3021	3197
$k = 10000$	3052	3112	3143	3238	3378	3569

Table 2: Average Running Times for a Uniform Random Distribution

4 Implementation and Testing

The algorithm has been implemented in C. We note that a nice feature of our algorithm is that it makes use of the standard Delaunay diagram as its basic data structure. Though the diagram is by no means simple to compute, it is commonly used in many applications and may already be available. Not counting the Delaunay diagram code or the priority queue implementation (another data structure which is commonly available) our algorithm proved both quick and easy to implement, and required less than 50 lines of code. It has been tested for values of n to 32000 and values of k to 10^8. These tests have been done on sets of points in uniform random distributions in a square and disc, as well as for the set of n points on a $\sqrt{n} \times \sqrt{n}$ square grid.

Table 2 gives the running times of Algorithm 1 for various values of n and k. In this series of tests, we used random real points in a uniform distribution over a disc. For a given n, we generated 50 random point sets, and for each point set ran the algorithm for all of the values of k. The times given in Table 2 are the averages of the 50 runs for Step 3) of Algorithm 1 only, as that is where the interesting expected run time analysis was. From Table 2, it is not difficult to see the algorithm's running time was linear in k. For the $n = 8000$ column, for instance, the r^2 value is .999. For fixed k, we also compared our times to the $\log^2 n$ curve. For $k = 10000$, the data gives an r^2 value of .995. We ran similar tests for other uniform distributions, including uniform random distributions of real points in a 10000×10000 square, and uniform random points from a 10000×10000 integer grid. In almost all cases, the r^2 values were greater than .995 for both fixed k and fixed n.

Additionally, we note that in Table 2 if we remove the $n = 1000$ column, the r^2 value for $k = 10000$ is 1.000. A closer look at the algorithm gives a hint as to why this is the case. The priority queue initially contains only antipodal pairs, and so has initial size $O(h)$ and so initially the queue operations require time $O(\log h)$ rather than $O(\log n)$ until the queue has had a chance to grow. For uniform distributions, it is well known $h << n$.

5 Summary and Open Problems

We have presented an algorithm for enumerating the k greatest interpoint distances for a set of n points in the plane. For uniform distributions the algorithm uses $O(n+k)$ space and has an expected running time of $O(n \log n + min(k \log^2 n, n^2 \log n))$. When k is $O(n^{4/3})$, the algorithm also provides a solution to Problem 2 which is asymptotically faster than the best known selection algorithm of [1]. Thus for the selection problem, Algorithm 1 which efficiently selects long distances provides a nice complement to the algorithms of [6] or [12] which are efficient at selecting small distance.

It would be nice to improve the worst case results for this problem. The nature of the problem intuitively suggests the use of generalized Voronoi diagrams, specifically farthest point Voronoi diagrams rather than closest point diagrams. The solution of Katoh and Iwano [8] makes use of k-order Voronoi diagrams in an algorithm which is essentially $O(k^{4/3} \log n)$ in the expected and worst case.

Thus their algorithm is not as efficient as ours in the average case, but has a significantly better worst case complexity. We pose as an open problem: Can Problem 1 be solved in $O(n \log n + k \log n)$ worst case time?

6 Acknowledgements

The authors would like to thank Volker Turau for suggesting an improvement to Step 2 of the algorithm, making use of the all antipodal points algorithm rather than an earlier version of the paper which used all pairs of point on the convex hull.

References

[1] P.K. Agarwal, B. Aronov, M. Sharir, and S. Suri, "Selecting Distances in the Plane." *Proceedings of the Sixth Annual ACM Symposium on Computational Geometry* (1990) 321-331.

[2] M. Bern, D. Eppstein, and F. Yao, "The Expected Extremes in a Delaunay Triangulation." *IJCGA* 1:1 (1991) 79-91.

[3] M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, and R.E. Tarjan, "Time Bounds for Selection". *J. Computer and Systems Sciences* 7(1973), 448-461.

[4] B. Chazelle, "New Techniques for Computing Order Statistics in Euclidean Space" *Proceedings of the First Annual ACM Symposium on Computational Geometry* (1985) 125-134.

[5] I.K. Crain, "Monte-Carlo Simulation of Random Voronoi Polygons; preliminary resulls." *Search* 3 (1972) 220-221.

[6] M. Dickerson and R.L. Drysdale, "Enumerating k Distances for n Points in the Plane." *Proceedings of the Seventh Annual Symposium on Computational Geometry* (1991) 234-238.

[7] R. Dwyer, "Higher-Dimensional Voronoi Diagrams in Linear Expected Time." *Discrete and Computational Geometry* 6 (1991) 343-367.

[8] N. Katoh and K. Iwano, "Finding k Best Distances of n Points in the Plane." *Kobe University and IBM, Japan*, 1991.

[9] G. Manacher and A. Zobrist, "Probabilistic methods with heaps for fast-average-case greedy algorithms." *Advances in Computing Research* vol. 1 (1983) 261-278.

[10] F. Preparata and M. Shamos, Computational Geometry, Springer-Verlag, 1985.

[11] J.S. Salowe, "*L*-Infinity Interdistance Selection by Parametric Search." *Information Processing Letters* 30(1989) 9-14.

[12] J.S. Salowe, "Shallow Interdistance Selection and Interdistance Enumeration." *Proceedings WADS '91, Lecture Notes in Computer Science* Spring-Verlag Berlin (1991) 117-128.

[13] M. Shamos and D. Hoey "Closest Point Problems." *Proceedings of the 16th Annual Symposium on Foundations of Computer Science* (1975) 151-162.

[14] A. Yao, "On Constructing Minimum Spanning Trees in k-dimensional Space and Related Problems." *SIAM J. Computing* 11(1982), pp. 721-736.

[15] M. Yaglom and V.G. Boltyanskii, *Convex Figures* Holt, Rinehart, and Winston, 1961.