Algorithms for Proximity Problems in Higher Dimensions

Matthew T. Dickerson
Department of Mathematics and Computer Science
Middlebury College, Middlebury VT 05753

David Eppstein
Department of Information and Computer Science
Univ. of California, Irvine CA, 92717

Abstract

We present algorithms for five interdistance enumeration problems that take as input a set S of n points in $\boldsymbol{R}^d$ (for a fixed but arbitrary dimension d) and as output enumerate pairs of points in S satisfying various conditions. We present: an $O(n \log n + k)$ time and $O(n)$ space algorithm that takes as additional input a distance δ and outputs all k pairs of points in S separated by a distance of δ or less; an $O(n \log n + k \log k)$ time and $O(n+k)$ space algorithm that enumerates *in non-decreasing order* the k closest pairs of points in S; an $O(n \log n + k)$ time algorithm for the same problem without any order restrictions; an $O(nk \log n)$ time and $O(n)$ space algorithm that enumerates in nondecreasing order the nk pairs representing the k nearest neighbors of each point in S; and an $O(n \log n + kn)$ time algorithm for the same problem without any order restrictions. The algorithms combine a modification of the planar approach of Dickerson, Drysdale, and Sack [11] with the method of Bern, Eppstein, and Gilbert [3] for augmenting a point set to have a linear size bounded degree Delaunay triangulation. Thus, in addition to providing new solutions to these problems, the paper also shows how the Delaunay triangulation can be used as the underlying data structure in a unified approach to proximity problems even in higher dimensions.

1 Introduction

In this paper, we present efficient algorithms for the following problems:

Problem 1. (Fixed-Radius Near-Neighbors Search) *Given a finite set S of n distinct points in $\boldsymbol{R}^d$, and a distance δ. For each point $p \in S$ report all pairs of points $(p, q), q \in S$ such that the distance from p to q is less than or equal to δ.*

Problem 2. (Enumerating Distances in Space) *Given a finite set S of n distinct points in $\boldsymbol{R}^d$, with $d_1 \leq \cdots \leq d_{\binom{n}{2}}$ the distances determined by the pairs of points in S. For a positive integer $k \leq \binom{n}{2}$, enumerate k pairs of points which realize $d_1, \ldots, d_k$.*

Problem 3. (K Nearest Neighbors) *Given a finite set S of n distinct points in $\boldsymbol{R}^d$, and a positive integer $k \leq n-1$, enumerate the k nearest neighbors of each point in S.*

The first two problems are closely related. Simply stated, Problem 2 is to report in nondecreasing order by distances the k closest pairs of points in S (with the simplifying assumption that for multiple pairs with equivalent distances, we may enumerate them in arbitrary order). If δ of Problem 1 is the distance to a unique k^{th} longest distance for the k of Problem 2, then the pairs of points output in the solutions to the two problems are identical. Note that k is not necessarily known in advance for either problem, but for Problem 2 may be determined dynamically by some other condition. We also examine a slightly easier version of Problem 2 where

k is known in advance and the pairs are not necessarily enumerated in order, and also an easier version of Problem 2 where we do not require the neighbors to be enumerated in order by distance.

The algorithms we present in this paper extend the recent planar results of Dickerson and Drysdale [9] and Dickerson, Drysdale, and Sack [11] to higher dimensions by making use of the results of Bern, Eppstein, and Gilbert [3] on *provably good mesh generation.* Bern, et al. showed how for a set S of points in arbitrary dimension, a superset S' of S could be found in $O(n \log n)$ time so that the Delaunay triangulation of S' has linear complexity. Specifically, they give a construction such that $|S'|$ is $O(|S|)$, and with the added property that the degree of each vertex in the Delaunay triangulation of S' is bounded by a constant. Our paper presents a unified approach to Problems 2, 1, and 3. We show for the first time how Delaunay triangulation can be used in efficient solutions to general proximity problems in higher dimensions. We describe our methods with respect to the Euclidean L_2 metric, but we do not use this metric in any essential way; similar techniques and results apply to other metrics on R^d, and in particular to the L_p metrics.

1.1 Background and Previous Results

Problem 2 was posed by Smid [24]. He presented an $O(n \log n)$ time $O(n)$ space algorithm for enumerating the $O(n^{2/3})$ smallest distances for a set of n points in d-space for any L_p metric, and posed as an open problem enumerating the $\Theta(n)$ smallest distances in $O(n \log n)$ time and $O(n)$ space. He used this as a subroutine in solving the following problem: Given a set P of n points in R^d, create a linear size data structure supporting the *insert*(x, P), *delete*(x, P), and *minimum-distance*(P) operations in $O(n^{2/3} \log n)$ time for any L_p metric. Smid showed how the update times for his data structure could be improved to $O(\sqrt{n} \log n)$ time given an $O(n \log n)$ solution to Problem 2 for $k = n$. (Smid also has an algorithm to solve this problem in k-dimensional space in $O((\log n)^{k+2})$ amortized time and $O(n(\log n)^k)$ space [25].) A solution to Problem 2 is also an important substep in one of the fast greedy triangulation algorithms of Dickerson, Drysdale, McElfresh, and Welzl [12]. For this application, k is not known in advance (the enumeration is terminated when the triangulation is complete) and the pairs are required in nondecreasing order of distance.

Closely related to Problem 2 is the following problem recently investigated by Chazelle [8], and by Agarwal, Aronov, Sharir, and Suri [1]:

Problem 4 (Selecting Distances). *Given a finite set S of n points, 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.*

A solution to Problem 2 clearly provides a solution to Problem 4, although we would expect there to be a faster algorithm for *selection* than for the *enumeration* problem which may have large output size. (That is, it would be nice to have an algorithm for selection whose running time is independent of k.)

Problem 1 has also received considerable attention. It was pointed out in [9] that the fixed-radius search arises in many situations when we have a density restriction "no more than m pairs of points may lie within a given distance of each other." The problem was originally solved by Bentley, Stanat, and Williams [5] in worst case time $O(3^d dn \log n + 3^d k)$ where d is the dimension and k the number of pairs reported. Algorithms for problem 1 have also been used by Salowe [21, 22] and Lenhof and Smid [17] as subroutines in parametric-search methods for solving Problems 2 and 4.

Problem 3 is a generalization of the well-known nearest neighbors problem. For classification problems, it is more robust than a simple nearest neighbors search. The graph of k nearest neighbors to each point has certain interesting theoretical properties [18, 19]. Eppstein and Erickson [13] showed how a variety of clustering problems such as those of finding k points with minimum diameter, circumradius, or variance, could all be solved efficiently using algorithms for Problem 3 as a subroutine, improving previous techniques based on kth order Voronoi diagrams. Problem 3 has also been used for contouring in geographic information systems.

Chazelle [8] presented a subquadratic solution to Problem 4 based on the batching technique of Yao [27]. The algorithm works in any dimension in time $O(n^{2-\beta(d)} \log^{1-\beta(d)} n)$ where $\beta(d) = 1/(2^d + 1)$. Thus for dimension 2 the running time is $O(n^{9/5}\log^{4/5} n)$. More recently, Agarwal, Aronov, Sharir, and Suri [1] have improved that result for the *planar* case providing a deterministic algorithm that runs in time $O(n^{3/2}\log^{5/2} n)$, and a randomized algorithm with expected running time of $O(n^{4/3}\log^{8/3} n)$. Agarwal, in a personal communication, has claimed that the randomized algorithm can be made deterministic without affecting the running time significantly. In d-dimensions, he can select the kth distance in time $O(n^{2(1-1/(d+1))+\epsilon})$, where ϵ is any arbitrarily small constant.

Salowe has also solved the interdistance selection problem for the L_∞ metric in d-dimensions in $O(n\log^d n)$ time [20] for $k \le n$, and has since extended these results to get an $O(n\log n + k)$ time algorithm for Problem 2 that works for any L_p metric [21, 22]; however the value of k must be known in advance, and the distances are not enumerated in order. As a sub-step, Salowe also presents an algorithm to solve Problem 1, the fixed-radius near-neighbor search problem, in time $O(n\log n + k)$ for L_p metrics in d-dimensions. This algorithm was inspired by Vaidya's optimal all-nearest-neighbors algorithm [26]. Lenhof and Smid [17] have also presented an algorithm to compute the k closest pairs (k known in advance and the pairs not generated in order) in $O(n\log n + k)$ time, using an approach similar to that of Salowe. Our paper presents algorithms with similar asymptotic running times, using different methods than those of Salowe, and Lenhof, and Smid.

For the planar case of Problem 2, enumerating the smallest k distances, Dickerson, Drysdale, and Sack [11] presented an $O(n\log n + k\log k)$ time and $O(n+k)$ space algorithm. A nice feature of the algorithm of [11] is that it is based on a common data structure, the Delaunay triangulation. It is both simple to state and easy to implement. Unfortunately, this result did not extend directly to higher dimensions. Though the algorithm can be shown to work correctly in any dimension, the running time deteriorates for $d > 2$ because the Delaunay triangulation, even for $d = 3$, may have quadratic size.

Previous work on Problem 3 has generally been independent of work done on the other problems, with very different methods used in the solutions. Problem 3 can be solved by constructing the order $(k+1)$ Voronoi diagram, and then for each point p determining the other k points lying in the same Voronoi region. For small values of k in the plane, this is a fairly efficient method. Lee [16] showed how to construct the order k Voronoi diagram in $O(k^2 n\log n)$ time, and Aggarwal et al. [2] have since improved that result to $O(k^2 n + n\log n)$ time. Dickerson et al.[11] presented an asymptotically faster algorithm requiring $O(n\log n + kn\log k)$ time for the planer case; as with their solution to Problems 2 and 1, it searches the standard Delaunay triangulation. Eppstein and Erickson [13] solved the planar problem for the simpler L_∞ metric in time $O(n\log n + kn)$. Once again, however, these approaches were not efficient in higher dimensions. Vaidya [26] gives an alternate approach based on a modified form of quadtrees; his algorithm works in any dimension and requires $O(kn\log n)$ time.

In this paper, we extend the result of [11]. We show how to make use of the results of [3] on linear-sized higher dimensional Delaunay triangulation to solve proximity problems in higher dimensions. Because of the addition of Steiner points, it is not at all clear that even with the linear size and constant degree bound of the constructed Delaunay triangulation, that the algorithms will be efficient. In particular, any given point may have a large number of neighboring Steiner points that will increase the amount of work done even though we do not want to report these pairs as part of our output. The analysis of our algorithm will require an amortization argument to show that the amount of extra work we do is proportional to the size of the input and *desired* output.

Our results for the unordered variants of Problem 1 and Problem 2 are matched asymptotically by those of Salowe [22] and Lenhof and Smid [17]. And empirical results for the algorithm of [17] in two dimensions compare favorable with the results of [11]. However the method of [17] requires that k be known in advance and does not output the pairs in order, and thus may not be used as a substep in the greedy triangulation. Our methods may also be somewhat simpler: in particular, unlike the approach of [22], our algorithm for Problem 2 does not require the use of parametric search. Our algorithm for Problem 3 is fastest known solution asymptotically, and is indeed optimal as well as being conceptually simple.[1]

[1]Very recently Callahan [6] has claimed that his work with Kosaraju [7] provides an alternate solution to Problem 3 with the same running time $O(n\log n + kn)$

Preprocessing Phase Given a finite set S of n distinct points, construct a linear sized, bounded vertex degree Delaunay triangulation D of S' as described in [3].

Search Algorithm From each point p in the original set S, do a depth first search on D. Halt each branch of the search at the first point q such that $d(p,q) > \delta$. Report all pairs (p,q) such that $d(p,q) \leq \delta$ and $q \in S$. (Don't report (p,q) if $q \in S' - S$.)

Figure 1: Algorithm 1

2 Fixed-Radius Near Neighbors Search in Space

We begin with the Problem 1, the fixed-radius near neighbors search problem, as the proof of correctness and amortization arguments are slightly simpler than those for Problems 2 and 3. Throughout the paper, we let $d(p,q)$ be the Euclidean (or L_2) distance from p to q for $p, q \in \mathbf{R}^d$.

We present an algorithm for Problem 1 that requires $O(n \log n + k)$ time and $O(n)$ space. Specifically, we have a one-time preprocessing phase of $O(n \log n)$ time and $O(n)$ space to compute the higher dimensional Delaunay triangulation with added Steiner points. Given this data structure, we can solve the problem for any value of δ in $O(n+k)$ time and $O(n)$ space. Algorithm 1 is given in Figure 1. A proof of the algorithm's correctness and complexity will follow.

2.1 Proof of Correctness

To prove the correctness of Algorithm 1, we use Lemma 1 below. This is a similar version of a lemma for the planar case appearing as Lemma 1 in [11].

Lemma 1. *Let S be a set of distinct points in $\mathbf{R}^d$, and D the Delaunay triangulation of S. Let p and q be points in S. Then either (p,q) is an edge in D or there is a point $r \in S$ with edge $(r,q) \in D$ and $d(p,r) < d(p,q)$.*

Proof: Let H be the largest empty sphere contained in the diameter sphere H' of (p,q) and lying on H' at point q. If $H = H'$, then H' is an empty sphere containing edge (p,q), and (p,q) is in D. Otherwise there is some point r in H' tangent to H. Since H is empty, (r,q) is in D. And since r is in H', which is in turn contained in the sphere with radius (p,q) centered at p, $d(p,r)$ must be less than $d(p,q)$. $\square$

That Algorithm 1 correctly reports all pairs (p,q) with $d(p,q) \leq \delta$ follows from Lemma 1 by induction on $d(p,q)$. The base case for each $p \in S$ is the closest point $p^* \in S$. Since p^* is a nearest neighbor to p, the edge (p, p^*) must be in D , and thus p^* is found on the depth-first search from p. Now consider another point q with $d(p,q) \leq \delta$. If (p,q) is also in D, then it too is found directly in the depth-first search. But by Lemma 1, if (p,q) is not in D then there is a point $r \in S$ with (r,q) in D and $d(p,r) < d(p,q)$. By our inductive hypothesis, r will be reported in the depth-first search, and since (r,q) is in D, we know q will also be reported once r is found.

2.2 Steiner Point Density

Before we analyze our fixed radius search algorithm, we prove a key lemma about the density of Steiner points in the construction of [3]. Recall that the result of [3] was a set $S' \supset S$ with $|S'| = O(|S|)$, such that the Delaunay triangulation of S' has a constant degree bound. We extend the $|S'| = O(S)$ bound to a more local condition to say that, roughly, within any region of $\mathbf{R}^d$, the number of points in S' is proportional to that in S.

We first describe in some detail the construction of S' from [3], as that paper was more concerned with two-dimensional triangulation algorithms and omitted some important details from the higher-dimensional construction. Let $x = O(\sqrt{d})$ be sufficiently large that any sphere of radius x in R^d contains at least one cube with vertices having integer coordinates. We construct a *quadtree* by starting with a hypercubical *root box* containing S, and then recursively subdividing each box into 2^d smaller boxes until each box contains at most one point and each input point is in the center of a grid of 3^d equal-size boxes. As we perform this subdivision, we maintain a *balance condition* that no two adjacent unsubdivided hypercubes can differ in size by more than a factor of two. If more than some fixed number c of subdivisions occurs without separating any points, we identify a *cluster* of points which we triangulate recursively, using a root box for the recursive cluster that is the center cube of a grid of 5^d cubes, that may then be subdivided to meet the same balance condition above. Note that none of the outer cubes in such a grid will be subdivided. In the outer quadtree, we treat this cluster as a single point, so it should end up in a grid of 3^d equal-size boxes.

We then form S' by replacing each box in the quadtree or surrounding a recursive cluster by a grid of x^d points, and taking the union of these grid points with our input set. These grids have the property that no sphere can pass through two nonadjacent boxes without having an interior grid point. As a consequence, the complexity of the Delaunay triangulation of S' is proportional to the number of boxes in the quadtree, that as we note in the proof below is $O(n)$.

Lemma 2. *Let D be a disk of radius r in R^d, and let $D(1+\epsilon)$ be a concentric disk of radius $r(1+\epsilon)$ for some fixed $\epsilon > 0$. Let m denote the number of input points within $D(1+\epsilon)$, and let m' denote the number of points of S' within D. Then $m' = O(m)$, with a constant of proportionality depending only on d and ϵ.*

Proof: Define the *level* of a Steiner point p, denoted $\ell(p)$, to be the size of the quadtree box containing p. If p was added as part of a box in the grid around a recursive cluster, then let the level of p be the size of that box.

In any quadtree, the boxes formed by the balance condition are within distance $O(\ell(p))$ of boxes formed by some other subdivision. And the limit on the number of subdivisions possible without forming a recursive cluster implies that for any Steiner point p, there is a pair of input points within distance $O(\ell(p))$ from p, separated by the subdivision of a hypercube of size $\Theta(\ell(p))$. Thus globally we can charge each Steiner point to the subdivision event in which that pair of points was separated, and each event is charged $O(1)$ times, giving the $O(n)$ bound of [3].

Locally, within D we can charge each Steiner point p for which $\ell(p) = O(\epsilon)$ to an event in which two input points in D' are separated. There can be $m-1$ such events, so there can be $O(m)$ such Steiner points. The remaining Steiner points, for which $\ell(p) = \Omega(\epsilon)$, are separated by a distance of $\Omega(\ell(p))$ from each other and hence there can be at most $O((1/\epsilon)^d) = O(1)$ such points. □

2.3 Analysis

We now provide an amortized analysis of the running time of our fixed radius search algorithm. The preprocessing phase takes time $O(n \log n)$, as shown in [3]. The search phase takes time proportional to the number of vertices searched, since the Delaunay triangulation has fixed degree. Some of those vertices form pairs that are part of the output, while others are Steiner points added in the preprocessing phase and do not contribute to the output. To analyze the algorithm, we must amortize the cost of looking at a pair of points (p, q) with q a Steiner point against the actual cost of all reported pairs. The argument follows.

Consider dividing R^d into a grid of hypercubes of fixed diameter δ. Let points lying on the boundary of more than one grid be considered as belonging to the adjacent grid in the direction of negative infinity. Within a grid cell g_i let m_i denote the number of Steiner points and n_i denote the number of input points. It is clear that all $\binom{n_i}{2}$ pairs of input points in the same grid cell are reported.

Now consider a Steiner point $s \in S' - S$, contained in cell g_i. The amount of work caused by this Steiner point is bounded asymptotically by the number m of original points within a distance δ from s. We charge m units to that one of the $O(1)$ grid cells within distance δ of g_i and having the largest value of n_i.

Lemma 3. *Each cell g_i is charged at most $O(n_i^2)$ units.*

Proof: Let some Steiner point s be within distance δ of m input points. There are $O(1)$ grid cells within distance δ, so s must be within distance δ of a grid cell having $\Omega(m)$ points, and will only charge a cell with at least that many points. Equivalently, each cell g_i is charged only by Steiner points with $m = O(n_i)$.

To complete the proof we show that cell g_i is charged by at most $O(n_i)$ Steiner points. For suppose that some cell g_j within distance δ of g_i has more than cn_i Steiner points for some sufficiently large c. Then by Lemma 2 there must be $\Omega(cn_i)$ input points within distance δ of g_j, thus there would be a single cell with $\Omega(cn_i)$ points and if c is sufficiently large this would be more than n_i and g_i would not be charged by points in g_j. Thus g_i is charged only from cells with $O(n_i)$ Steiner points, and it can be charged only by the $O(1)$ cells within distance δ of g_i, completing the proof. $\square$

Thus the time spent searching Steiner vertices can be charged against the number of output pairs, and we have the following result.

Theorem 1. *Given a set of n points in $\boldsymbol{R}^d$, We can list all interpoint distances less than some given δ, in time $O(n \log n + k)$ where k denotes the number of such distances.*

3 Enumerating k Smallest Distances

We now present an algorithm for Problem 1. The basic idea of the algorithm is simply to interleave simultaneous breadth-first searches[2] from all vertices in the original set S, where *breadth* is defined by distance and not by number of edges. For those familiar with Algorithm 1 in [11], this algorithm is based on a similar strategy but with four main differences: in the *preprocessing* phase, we use the Steiner Delaunay triangulation as presented in [3], rather than the standard (possibly quadratic sized) Delaunay triangulation for points in $\boldsymbol{R}^d$; in the *enumeration* phase we allow the queue to grow beyond size k (though we show that the size will remain $O(k)$); the bounded degree of the vertices implies that we need not sort adjacent edges in increasing order by length; and finally we treat all points and pairs of points equally, including Steiner points, except that we only begin our breadth-first searches from original points, and we only *report* pairs where both points are from the original set S.

The algorithm requires $O(n \log n + k \log k)$ time and $O(n + k)$ space. Specifically, we have a one-time preprocessing phase of $O(n \log n)$ time and $O(n)$ space, followed by an algorithm requiring $O((n+k) \log(n+k))$ time and $O(n + k)$ space. A discussion of the algorithm's correctness and complexity will follow.

3.1 Correctness of Algorithm 2

Algorithm 2 is given in Figure 2. We let Q be a priority queue that supports **DeleteMin** and **Insert** operations. The **DeleteMin** operation deletes from Q the pair (v, w) with minimum distance and returns (v, w). The **Insert** (v, w) operation checks to see if (v, w) is already in Q, and inserts it only if it is not. As noted, what this algorithm does is essentially a simultaneous interleaved breadth-first search from all vertices in S.

The correctness of Algorithm 2 follows, using Lemma 1, by induction on the number of pairs deleted from Q. The inductive hypothesis is: At the jth iteration of the loop, a jth closest pair (v, w) of points from $S \times S'$ will be deleted from the priority queue Q. Furthermore, all pairs of points separated by distance $\leq d(v, w)$ will have been inserted into the queue before step j. A detailed proof for a slightly simpler planar version of the algorithm is given in [11].

[2]We might call these "best-first" or "closest-first" search. We cannot use the simpler depth-first search as we did in Algorithm 1 for two reasons: 1) we do not know the search distance δ in advance, and so we don't know the halting condition; and 2) we must enumerate the distance in non-decreasing order.

1. Preprocessing Phase Given a finite set S of n distinct points in R^d, construct a linear sized, bounded vertex degree Delaunay triangulation D of S' as described in [3].

2. Initialization Phase For each edge $(v, w) \in D$ with $v \in S$, **Insert**(v, w).

3. Enumeration Phase

```
Let $i := 1$.
WHILE $i <= k$ DO
        a) Let $(v, w) :=$ **DeleteMin**$(Q)$; mark $(v, w)$ "visited"
        b) Let $\delta := d(v, w)$.
        c) **if** $v, w \in S$ and $v <_{\text{lex}} w$ **then**
                report $(v, w)$ with $d_i = \delta$.
                Let $i := i + 1$.
        d) $\forall (w, x)$ **if** $(v, x)$ has not been visited **then Insert**$(v, x)$;
END
```

Figure 2: Algorithm 2

3.2 Analysis

We now analyze the complexity of Algorithm 2. As already discussed, Step 1 requires $O(n \log n)$ time and $O(n)$ space and results is the Delaunay triangulation D of a point set $S' \supseteq S$ with $|S'|$ is $O(|S|)$ and the maximum vertex degree in D bounded by a constant [3]. Using an appropriate balanced tree implementation for our queue, we can perform the **Insert** and **DeleteMin** operations in $O(K)$ space and $O(\log K)$ time where K is the current size of the queue. The time required by Step 2 is therefore $O(n \log n)$.

To analyze the enumeration phase, Step 3, let K be the number of iterations of the WHILE loop. Since the maximum degree of vertices in D is also constant, step 3.d) contributes only a constant number of **Insert** operations per iteration of the loop, so the queue remains of size $O(n + K)$. A dictionary also of size $O(n + K)$ can be used to keep track of the edges already visited. Thus the **Insert** and **DeleteMin** operations each require time $O(\log(n + K))$. Steps 3.b) and 3.c) can be done in constant time per iteration. It follows that K iterations of the WHILE loop require a total of $O(K \log(n + K))$ time.

To prove our bound of $O((n + k) \log(n + k))$, it remains only to prove that K is $O(n + k)$. However this follows directly from the analysis given for Algorithm 1. Let δ be the length of the kth longest distance in S. We showed in our earlier analysis that we examine at most $n + k$ pairs of points in a search of radius δ. Having shown that K is $O(n + k)$, we have completed the proof of the following result:

Theorem 2. *We can enumerate the k smallest distances determined by a set of points in R^d, in order by distance, in time $O(n \log n + k \log k)$.*

This is asymptotically equivalent to the running time of the previously known planar algorithm, but now holds in any fixed dimension.

4 Enumerating k Unordered Distances

We now describe how to modify the bound of Theorem 2, to eliminate the $O(\log k)$ term in the case that k is known in advance and the k distances to be enumerated need not be output in order.

The analysis of our previous algorithm tells us that there is some (large) constant c, such that if we enumerate the ck smallest distances in S' we will be guaranteed to find among them the k smallest distances in S. The actual k smallest distances can then be found by a linear time selection algorithm [4]. (An explicit bound on c is needed to implement the algorithm, but this can be determined by a more careful analysis in Lemmas 2 and 3; we omit the details here.)

Thus we can reduce the problem to one of finding the ck smallest distances in a collection of n bounded-degree breadth first search trees. We use an algorithm of Frederickson [15] for performing selection in heap-ordered trees.

Lemma 4 (Frederickson [15]). *Let T be a binary tree in which each vertex has a weight, and in which the weight of any vertex is less than the weight of its children. Then we can find the k smallest weights of vertices in T, in time $O(k)$.*

Note that the time bound of Lemma 4 does not depend on $|T|$, but only on k. For instance, Frederickson uses this lemma to find the k smallest spanning trees in a graph in time $O(m \log \beta(m,n) + k^{3/2})$ [14], even though in this application T has exponential size.

In our application, the breadth first search trees are not binary, but they have bounded degree, which is sufficient for the lemma (one can translate a degree-δ tree into a binary tree by expanding each vertex into a subtree of $\delta - 1$ vertices; this only changes by a factor of $\delta - 1$ the number of nodes that need to be enumerated). In our breadth first search trees, the weight of each vertex is the Euclidean distance to the tree root; these weights have the appropriate heap ordering by Lemma 1. There is only one further complication: Lemma 4 requires that the input be a single tree T, whereas we wish to perform global selection in a forest of n trees. We form a single tree T by hooking the forest together using a binary tree with n leaves; all the new nodes of the binary tree will have weight zero. We can then find the ck smallest distances in S' by selecting the $(\delta - 1)ck + 2n - 1$ smallest weight vertices in T, eliminating repetitions, and ignoring the $2n - 1$ zero-weight vertices in the binary tree connecting our breadth-first search trees and at the roots of the breadth-first search trees.

Theorem 3. *We can list the k smallest distances determined by a set of points in $\mathbf{R}^d$, as an unordered set, in time $O(n \log n + k)$.*

5 Enumerating k Nearest Neighbors

We now present our algorithm for Enumerating k nearest neighbors of each point in S. This algorithm uses the same approach as the previous two: searching the Steiner Delaunay triangulation. Here, however, the amortized analysis is slightly different. The output is actually of size nk rather than size k.

5.1 Correctness and Analysis

Algorithm 3 is given in Figure 3. Once again, the proof of correctness follows directly from Lemma 1. The analysis of the running time takes a slightly different turn, however. It is based on the following lemma.

Lemma 5. *Let S and S' be two sets of points in $\mathbf{R}^d$ for arbitrary but fixed dimension d. Let p be a given point in set S'. The number of points $q \in S$ such that p can be as close to q as q's kth nearest neighbor in S is $O(k)$.*

A generalization of the proof of Lemma 4 in [11] suffices to prove Lemma 5. We may now complete our analysis. The number of vertices visited in the breadth-first search from some point $p \in S$ is $O(k + k')$ where

1. Preprocessing Phase Given a finite set S of n distinct points in R^d, construct a linear sized, bounded vertex degree Delaunay triangulation D of S' as described in [3].

2. Enumeration Phase For each point p in S, do a breadth-first search (by distance) on D to find the k nearest neighbors in S.

Figure 3: Algorithm 3

k' is the number of Steiner points as close to p as its kth nearest neighbor in S. Though for a particular point p, k' may be as large as n, by Lemma 5 we know that a Steiner point is visited at most $O(k)$ times in all searches from all points in S. Since the number of Steiner points is $O(n)$, the number of visits to all Steiner points is $O(nk)$. Each queue operation used in the breadth-first search may require $O(\log n)$ time, since the queues can grow as big as $O(n)$ for a particular search. Our overall running time is therefore $O(nk \log n)$.

Theorem 4. *We can enumerate the k nearest neighbors to each of a set of points in R^d, in order by distance, in time $O(kn \log n)$.*

5.2 Unordered Variant of k Nearest Neighbors

We now describe how to speed up Algorithm 3 using the techniques of Lemma 4 in the case where we do not require the neighbors to be output in order by distance. As before our ordered enumeration algorithm consists of searching out to a certain distance δ in the breadth-first search tree. If we knew that distance, we could simply search the tree in depth first order as in Theorem 1. If we knew how many entries in the tree we were going to search, we could find all those entries using Lemma 4 and then select among them as we did in Theorem 3. However neither δ nor the number of entries $k + k'$ is available to us.

To get around these difficulties, we use the following strategy. Let κ be an estimate for the unknown value $k + k'$. Initially we choose $\kappa = O(k)$. We use Lemma 4 to find the κ smallest entries in the breadth first search tree, corresponding to the κ nearest neighbors to the root in S'. We scan that set of κ neighbors and count how many of them are in S. If k or more neighbors are in S, they must be a superset of the true k nearest neighbors, which we can find by a linear time selection algorithm [4]. If fewer than k neighbors are in S, we double κ and continue iteratively. The final value of κ will then be less than $2(k + k')$. The time per iteration is $O(\kappa)$ and the time for all iterations adds in a geometric series to $O(k + k')$. We have already seen that adding these quantities for all input points gives a total of $O(kn)$.

Theorem 5. *We can list the k nearest neighbors to each of a set of points in R^d, as an unordered set, in time $O(n \log n + kn)$.*

Finally, we note that the bound of Theorem 4 can be improved from $O(kn \log n)$ to $O(n \log n + kn \log k)$ time, matching the planar results of Dickerson et al.[11], by applying Theorem 5 and then using a sorting algorithm within each set of neighbors, but the resulting algorithm is considerably more complicated than that presented above.

6 Summary

We have given an $O(n \log n + k \log k)$ time and $O(n + k)$ space algorithm for the solution to Problem 2, reporting the k smallest interpoint distances of a set S of n points in a plane; we improved this to $O(n \log n + k)$ to list the distances without requiring them to be in order. We have also given an $O(n \log n + k)$ time algorithm for Problem 1, and $O(nk \log n)$ and $O(n \log n + kn)$ time algorithms for Problem 3. These algorithms are based on a modification of a common data structure, the Delaunay triangulation. This shows that the Delaunay triangulation, which because of its high complexity was previous thought to be of little use for solving proximity problems in higher dimensions, can be used as a theoretically efficient underlying data structure. Our algorithms for enumerating points in order are simple given the Steiner Delaunay triangulation. Our improved algorithms for listing the points out of order make use of some complicated subroutines, in particular the linear time selection algorithm of Blum et al. [4], but they demonstrate optimal asymptotic efficiency.

In all algorithms given in this paper, the initial $O(n \log n)$ time step is a true preprocessing phase used only to construct the Steiner Delaunay triangulation. The same structure remains for use in other applications. In particular, with the fixed radius search algorithm we have a running time of $O(n + k)$ for any search radius δ once the triangulation is computed.

6.1 Open Problems

There are a number of interesting open questions related to the work presented here.

(i) Can the result of Agarwal et al. [1] be improved for large k, for instance when k is $\Omega(n^2)$? How good a deterministic algorithm is possible? In particular, how quickly can the median problem be solved?

(ii) Our algorithm for enumerating the k smallest interpoint distances requires $O(n \log n + k \log k)$ time. The algorithms of Salowe and of Lenhof and Smid require $O(n \log n + k)$ time but does not give the distances in nondecreasing order and requires that k be at most n. Is there an algorithm that can enumerate the distances in non-decreasing order in $O(n \log n + k)$ time? A solution to this problem would give an $O(n^2)$ algorithm for listing all of the interpoint distances in increasing order, solving an old open problem.

(iii) We present asymptotically optimal methods for solving a number of proximity problems. These problems have also been solved optimally by Salowe [22], Lenhof and Smid [17], and Callahan and Kosaraju [6, 7] using different algorithms. Can we compare the constant factors in these asymptotic bounds, and the dependence of those bounds on the dimension, to determine which of the various algorithms would be best in practice? [3]

7 Acknowledgements

Work on this problem was begun at the Workshop on Computational Geometry (October 1993, Raleigh, North Carolina) sponsored by the Army Research Office and by the MSI at SUNY Stony Brook. This study was supported in part by the funds of the National Science Foundation, CCR-9301714 and CCR-9258355.

References

[1] P. K. Agarwal, B. Aronov, M. Sharir, and S. Suri, "Selecting distances in the plane", *Proc. 6th Annual ACM Symposium on Computational Geometry*, June 1990, 321– 331.

[2] A. Aggarwal, L. Guibas, J. Saxe, and P. Shor, "A linear-time algorithm for computing the Voronoi diagram of a convex polygon," *Discrete and Computational Geometry* **4** (1989) 591–604.

[3] In an updated version of [17], Lenhof and Smid have some preliminary results on actual running times for two dimensions comparing their algorithm to that of [11], and also a report on running times of only their algorithm in three dimensions.

[3] M. Bern, D. Eppstein, and J. Gilbert "Provably good mesh generation", *J. Computer and Systems Sciences* **48** (1994) 384–409.

[4] 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.

[5] J. Bentley, D. Stanat, and E. Williams Jr, "The Complexity of finding fixed-radius near neighbors," *Information Processing Letters* **6** (1977) 209-213.

[6] P. B. Callahan, "Optimal parallel all-nearest-neighbors using the well-separated pair decomposition", *34th IEEE Symp. Foundations of Computer Science*, 1993, 332– 340.

[7] P. B. Callahan and S. R. Kosaraju, "A decomposition of multi-dimensional point-sets with applications to k-nearest-neighbors and n-body potential fields", *Proc. 24th ACM Sym. Theory of Computing*, 1992, 546–556.

[8] B. Chazelle, "New techniques for computing order statistics in Euclidean space", *Proc. 1st Annual ACM Symposium on Computational Geometry*, June 1985, 125–134.

based Symposium

[9] M. Dickerson and R. L. Drysdale, "Fixed radius search problem for points and segments", *Information Processing Letters* **35** (1990) 269–273.

[10] M. Dickerson and R. L. Drysdale, "Enumerating k distances for n points in the plane", *Proc. 7th Annual ACM Symposium on Computational Geometry*, June 1991, 159–168.

[11] M. Dickerson, R. L. Drysdale and J.-R. Sack "Simple algorithms for enumerating interpoint distances and finding k nearest neighbors", *International Journal of Computational Geometry and Applications* **2:3** (1992) 221–239.

[12] M. Dickerson, R. L. Drysdale, S. Mcelfresh, and E Welzl "Fast Algorithms for the Greedy Triangulation", *Proc. 10th Annual ACM Symposium on Computational Geometry*, June 1994, 211–220.

[13] D. Eppstein and J. Erickson, "Iterated nearest neighbors and finding minimal polytopes", *Proc. 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, Jan. 1993, 64– 73.

[14] G. N. Frederickson, "Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees", *Proc. 32nd Annual Symp. on Foundations of Computer Science*, 1991, 632–641.

[15] G. N. Frederickson, "An optimal algorithm for selection in a min-heap", *Information and Computation* **104** (1993) 197–214.

[16] D.T. Lee, "On k-nearest neighbor Voronoi diagrams in the plane," *IEEE Trans. Comput.* **31** (1982) 478–487.

[17] H-P Lenhof and M. Smid, "Enumerating the k closest pairs optimally," em Max-Planck-Institut für Informatik MPI-I-92-118 May 1992.

[18] G. L. Miller, S.-H. Teng, and S. A. Vavasis, "An unified geometric approach to graph separators", *32nd Annual Symposium on Foundations of Computer Science*, 1991, 538–547.

[19] M.S. Paterson and F.F. Yao. On nearest-neighbor graphs. 19th Int. Colloq. Automata, Languages, and Programming (1992) to appear.

[20] J. S. Salowe, "L-infinity interdistance selection by parametric search", *Information Processing Letters* **30** (1989) 9–14.

[21] J. S. Salowe, “Shallow interdistance selection and interdistance enumeration”, *Proc. WADS ’91, Lecture Notes in Computer Science*, **519** (Springer-Verlag, Berlin, 1991) 117–128.

[22] J. S. Salowe, “Enumerating distances in space”, *International Journal of Computational Geometry and Applications*, **2:1** (1992) 49–59.

[23] M. Shamos and D. Hoey “Closest point problems.” *Proc. 16th Annual IEEE Symposium on Foundations of Computer Science*, 1975, 151–162.

[24] M. Smid, “Maintaining the minimal distance of a point set in less than linear time”, *Algorithms Review* **2** (1991) 33–44.

[25] M. Smid, “Maintaining the minimal distance of a point set in polylogarithmic time”, *Proc. 2nd Annual ACM-SIAM Symposium on Discrete Algorithms*, 1991, 1–6.

[26] P. M. Vaidya, “An $O(n \log n)$ algorithm for the all-nearest-neighbors problem”, *Discrete and Computational Geometry* **4** (1989) 101–115.

[27] A. Yao, “On constructing minimum spanning trees in k-dimensional space and related problems”, *SIAM J. Computing* **11** (1982) 721–736.