Title
Iterated nearest neighbors and finding minimal polytopes

Authors
Eppstein, David
Erickson, Jeff

Publication Date
1992-06-30

Peer reviewed

Iterated Nearest Neighbors and Finding Minimal Polytopes

David Eppstein and Jeff Erickson

Department of Information and Computer Science
University of California, Irvine, CA 92717

Tech. Report 92-71

June 30, 1992

Abstract

We introduce a new method for finding several types of optimal k-point sets, minimizing perimeter, diameter, circumradius, and related measures, by testing sets of the $O(k)$ nearest neighbors to each point. We argue that this is better in a number of ways than previous algorithms, which were based on high order Voronoi diagrams. Our technique allows us for the first time to efficiently dynamize our algorithms, to generalize them to higher dimensions, to find minimal convex k-vertex polygons and polytopes, and to improve many previous results. We achieve many of our results via a new algorithm for finding rectilinear nearest neighbors in the plane in time $O(n \log n + kn)$. Finally, we demonstrate a related method for finding k-point sets with minimum boundary measure or volume in arbitrary dimensions, generalizing our results for minimizing perimeter and an earlier result of the first author for minimizing area.

1 Introduction

A number of recent papers have discussed problems of selecting, from a set of n points, the k points optimizing some particular criterion [2, 10, 13, 14]. Criteria that have been studied include diameter [2], variance [2], area of the convex hull [13, 14], convex hull perimeter [2, 10, 14], and rectilinear diameter and perimeter [2]. Such problems are useful in clustering, line detection, statistical data analysis, and other geometric applications.

We study and improve known algorithms for many of these problems. We also introduce dynamic versions of these problems, in which the optimum must be maintained as the point set is updated. Our methods further generalize to higher dimensional versions of these problems. Our techniques apply to "one-dimensional" measures including all of the problems cited above, except for the two-dimensional area measure, for which the best known time bound remains $O(n^2 \log n + k^3 n^2)$ [13].

Previous algorithms for these problems used the following method. An *ad hoc* algorithm was determined, with time bounded by a polynomial $O(n^c)$. Then, it was shown that the optimum k-point set is contained in the set of points labeling a single region of the order-$O(k)$ Voronoi diagram. Constructing the Voronoi diagram and searching the $O(kn)$ such regions takes a total time of $O(n \log n + k^{c+1} n)$. Aggarwal *et al.* [2] reduced the number of regions to be searched from $O(kn)$ to $O(n)$. Thus the time becomes $O(n \log n + k^c n)$. However, there remains an anomaly in these time bounds: if k is $\Theta(n)$, the time is worse than the original $O(n^c)$ by a factor of n. Thus at some point the device of higher order Voronoi diagrams becomes worthless, and one must use a simpler algorithm.

We argue that, in this formulation, Voronoi diagrams should be replaced by sets of the $O(k)$ nearest neighbors to each point. There are several reasons why we believe this. First, the reduction to $O(n)$ regions to be searched is immediate, and avoids the complicated analysis of Aggarwal *et al.* [2].

Second, by finding neighbors of neighbors, we show that the number of regions can be further reduced to $O(n/k)$, improving the time bounds by a factor of k and eliminating the anomaly described above.

Third, our time bounds can be improved in a different way. The k nearest neighbors can be found in time $O(kn \log n)$, using Vaidya's algorithm [22]. For the rectilinear (L_1 or L_∞) metric, we further improve this to $O(n \log n + kn)$. Thus we get faster time bounds in the plane, even for problems such as circumradius for which the reduction to Voronoi diagrams is immediate.

Fourth, our method lends itself well to dynamization. As points are inserted one at a time, the neighbors of each new point may be computed quickly using standard techniques. In contrast, the Voronoi diagram may change by as many as $\Omega(n)$ edges at each insertion. Dynamic algorithms have been studied for many important geometric optimization problems, such as the closest pair, diameter, minimum spanning tree, and convex hull, but this is the first time that dynamic algorithms have been described for minimum measure subset problems.

Fifth, our approach generalizes to higher dimensions in a way that does not work for Voronoi diagrams. In dimension d, even first order Voronoi diagrams can have complexity $\Omega(n^{\lceil d/2 \rceil})$; whereas, the nearest neighbors can still be found in time $O(n \log n)$ using Vaidya's algorithm [22].

Finally, by applying an old combinatorial result of Erdős and Szekeres [15], we can generalize our techniques to find minimum measure convex polygons and polytopes.

A slight variant of our approach provides a natural generalization of our new minimum perimeter algorithm and the first author's minimum area algorithm [13] into arbitrary dimensions. Instead of using the neighbors to each point, we let each set of r points in the set define a particular polytope, for some constant r defined by the measure we are trying to minimize, and we examine the nearest neighbors to each polytope thus defined.

2 New Results

We present algorithms for the following problems.

- We find the k nearest rectilinear neighbors to each of a set of n points in the plane, in time $O(n \log n + kn)$, improving the previous $O(kn \log n)$ bound [22].

- Given a set of n points in the plane, we find the k-point set minimizing perimeter, L_∞ perimeter, circumradius, diameter, L_∞ diameter, or variance. Our results are summarized in the first column of Table 1. We improve all previous results [2, 10, 14], except for variance, which we improve for certain values of k.

- We maintain minimal point sets in the plane as points are inserted, under a variety of "one-dimensional" measures. Our results are sum-

Measure	Static time bound	Dynamic time bound
perimeter	$O(n \log n + k^3 n)$	$O(k^4 + \log^2 n)$
L_∞ perimeter	$O(n \log n + k^2 n)$	$O(k^3 + \log^2 n)$
circumradius	$O(n \log n + kn \log k)$	$O(k^2 \log k + \log^2 n)$
diameter	$O(n \log n + k^2 n \log^2 k)$	$O(k^3 \log^2 k + \log^2 n)$
L_∞ diameter	$O(\min\{n \log n + kn, n \log^2 n\})$	$O(k \log^2 k + \log^2 n)$
variance	$O(k^{3/2} n \log n + k^{3/2+\epsilon} n)$	$O(k^{3+\epsilon} + \log^2 n)$

Table 1. New results for finding minimum measure k-point sets, given n points in the plane. (ϵ is an arbitrarily small positive constant.)

Measure	Time bounds
circumradius	$O(kn \log n + k^{d-1} n \log^2 k)$
diameter	$O(kn \log n + 2^{O(k)} n)$
L_∞ diameter	$O(kn \log n + k^{d/2-1} n \log^2 k)$
variance	$O(k^{(d+1)/2} n \log n + k^{O(d^2)} n \log k)$
boundary measure	$O(n^d + 2^{O(k)} n^{d-1})$
L_∞ boundary measure	$O(n^d + k^{2d-1} n^{d-1})$
volume	$O(kn^d \log^{d+1} n + 2^{O(k)} n^d)$

Table 2. New results for finding minimum measure k-point sets, given n points in $\mathcal{R}^d$, for all $d > 2$.

marized in the second column of Table 1. No previous bounds are known for any of these problems.

- Given a set of n points in $\mathcal{R}^d$, with $d > 2$, we find the k-point set minimizing circumradius, diameter, L_∞ diameter, variance, boundary measure, L_∞ boundary measure, or volume. Our results are summarized in Table 2. We improve previous algorithms for circumradius and variance based on Voronoi diagrams, which run in time $O(n^{d+1})$ [2]. No previous bounds were known for the other problems. Our minimum volume algorithm generalizes previous results of the first author on minimum area polygons [13].

- We generalize all of our results to k-point convex polygons and polytopes. We derive time bounds with the same dependence on n as the corresponding k-point set algorithms, but with an exponential depen-

dence on k. We know of no previous results for these problems, except for a $O(kn^3)$ time bound on finding minimum perimeter k-gons [14], which we improve for small k.

3 Rectilinear Nearest Neighbors

We now describe a data structure for finding m rectilinear nearest neighbors in the plane. In the L_1 metric, above and to the right of any point p, points (x, y) are sorted by distance to p by the values of the function $x + y$. If we sort all points by these values, the nearest neighbors above and to the right of each point will be a subsequence of this sorted list. We combine neighbors from each of the four directions to find the nearest neighbors overall.

Our data structure is in the form of a balanced binary tree over the points, sorted by their y-coordinates. The tree root covers all n points, and for each tree node with i points we split the points into two *slabs*, consisting of the top $i/2$ and the bottom $i/2$ points. We build a data structure for each slab, and recursively subdivide slabs until we reach sets of a single point. Each input point will be in $O(\log n)$ slabs, and the points above and to the right of any query point p can be interpreted as the union of points to the right of p in $O(\log n)$ slabs above p.

We assume m is fixed. Without loss of generality $m > \log n$. In each slab, we wish to determine, for a query point p, the m nearest points to the right of p. If we did this for all slabs, we would generate $O(m \log n)$ neighbors, and queries would be slower than we wish. Instead, we partition the neighbors into *chunks* of $\Theta(m/\log n)$ points. Our data structure will enable us to find each succeeding chunk quickly, and we then combine chunks from different slabs to give the final set of m neighbors.

Within a single slab, we sweep from left to right, maintaining a list of points ordered by $x + y$. As we sweep across each point in the slab, we add it to the list. The positions to add new points into the list can be found in time $O(n)$ if the points are already sorted by x-coordinate. We would like our data structure to reconstruct the state of this list at each time in the sweep. This is a *persistent offline* data structure problem [12], in which we perform a number of updates (insertions into a linked list) and must then query different versions of the data structure (the list at different times in the sweep).

We maintain, at each point in the left to right sweep, a partition of the

sorted list of points into chunks of between $m/\log n$ and $2m/\log n$ points each. When a new point is inserted in the list, it is added to a chunk. When this addition causes a chunk to have too many points, it is split into smaller chunks. As we only need remember at most m neighbors to each query point, we only need keep $\log n$ chunks, so as one chunk is split another chunk may be removed from the end of the list.

To remember these manipulations we store the list of points in each chunk just before the chunk is split, and the list of all $\log n$ chunks at the same time. To find the neighbors for a query point p, we determine the next time t after our left-to-right sweep crosses p, at which some chunk is split. We then step through the sequence of chunks existing at time t. Each chunk contains between $m/\log n$ and $2m/\log n$ points, of which at least $m/\log n$ existed at the last time the chunk was split and hence are to the right of p. We eliminate the other points to the left of p. Thus in time $O(m/\log n)$ we can find each successive set of $\Omega(m/\log n)$ neighbors in the slab.

The time and storage for remembering the points in each chunk is $O(n)$. However if m is small there are $O(n)$ times at which a chunk may split, and hence $O(n \log n)$ storage for remembering the sequence of chunks at each time. We remove this unwanted logarithmic factor with a data structure for maintaining lists of $O(\log n)$ elements in a persistent offline manner.

Lemma 3.1. *Given a sequence of n insert and delete operations on a list, such that the list length is always $O(\log n)$, we can construct in time and space $O(n)$ a data structure such that, for any version of the list, we can step through the list in time $O(1)$ per step.*

Proof: Break the sequence into $O(n/\log n)$ subsequences of $O(\log n)$ operations each, and treat each subsequence separately. Within a subsequence, there are $O(\log n)$ items initially, and $O(\log n)$ items inserted. List all items by processing all the insert operations and none of the delete operations. Represent this list as an array of pointers to items, so that the item in a given position can be found in $O(1)$ time. Now represent each version of the list by an $O(\log n)$-bit integer, in which a one bit represents the presence of the item at that position. Each insert or delete can be performed with $O(1)$ steps of integer arithmetic, as can the operation of moving from one element to the next in a given version of the list. □

By analogy to the atomic heaps of Fredman and Willard [17] we call this data structure an *atomic list.* This completes the description of each slab, which we summarize below.

Lemma 3.2. *Given n points in the plane, sorted from left to right, we can in $O(n)$ time and space construct a data structure for which, given a value x, we can find the points with the smallest values of $x + y$, in chunks of $O(m/\log n)$ points at a time, in time $O(m/\log n)$ per chunk.* □

To finish the description of our data structure, we combine results from the $O(\log n)$ slabs into which space above the query is divided. We use a priority queue of one chunk from each slab. Each chunk's priority is the largest value of $x + y$ for any point in the chunk. We remove chunks one by one from the queue; when we remove a chunk we insert the next chunk from the same slab. Once we have removed chunks totalling at least m points, any remaining neighbors will have smaller values of $x + y$ than the priorities of the chunks left in the queue. Such points must be in chunks already in the queue, and we remove these chunks as well. This gives us $O(\log n)$ chunks and hence $O(m)$ potential neighbors. We reduce this to m neighbors using a linear time selection algorithm. Using a global list of all points, sorted by $x + y$, we can represent priorities as $O(\log n)$-bit integers, so we can perform priority queue operations in $O(1)$ time using atomic heaps [17].

Lemma 3.3. *For any fixed m, we can preprocess a set of n points in the plane, in time $O(n \log n)$, so that the m nearest rectilinear neighbors to any query point can be found in time $O(m + \log n)$.*

Proof: The query time is $O(m + \log n)$, once we have determined the version of the chunk list to use in each slab. For each slab, we maintain an index from the left to right order of points into this sequence of versions. We also index, for each slab, the relation between positions in the left to right order of points in the slab, and the same positions in the two smaller slabs into which it is divided. The position of the query point in the root slab can be found by binary search, after which we can follow the indices to find the list versions for all $O(\log n)$ slabs queried in $O(\log n)$ time. The time to construct each slab is $O(n)$ assuming the points are sorted from left to right. This sorted order can be maintained as slabs are split recursively, in $O(n \log n)$ total time. Thus all slabs can be constructed in time $O(n \log n)$. □

Theorem 3.1. *We can find the m nearest rectilinear neighbors to each of a set of n points in the plane, in time $O(n \log n + mn)$.* □

4 Iterated Neighbors

We now show that in any point set, in any dimensions, there is some point for which there are few neighbors of neighbors. We state the result more generally, in terms of spheres satisfying certain properties. Given two spheres A and B, we say that A is *entirely within* B if the closure of A is contained in the interior of B.

Lemma 4.1. *Let $\mathcal{S}$ be a set of spheres, so that no sphere is entirely within another sphere, and so that no sphere contains more than m centers of spheres. Let S be the smallest sphere in $\mathcal{S}$, and let U be the union of spheres in $\mathcal{S}$ having centers in S. Then U contains $O(m)$ sphere centers.*

Proof: Let R denote the radius of S. Because no sphere in U contains S, it follows that U is contained in a sphere of radius $3R$. This larger sphere can be partitioned into $O(1)$ regions, each with diameter R. If any of these regions contained more than m centers, any sphere centered in such a region either would contain too many centers or would be smaller than S. □

This result applies more generally to any family of homothetic convex bodies, and hence to "spheres" in any metric. We apply this result to sets of m nearest neighbors as follows. Given a point set, put a sphere around each point at a radius determined by its mth nearest neighbor. This sphere will contain exactly the m nearest neighbors of the point, and the set of all such spheres will satisfy the conditions of the lemma. Therefore there is some point for which the m nearest neighbors have $O(m)$ neighbors altogether.

This suggests the following algorithm outline. Suppose we can prove that the optimal k-point set (according to some specified criterion), if it contains a point, is contained in the m nearest neighbors of that point. Sort the points by the size of their neighbor spheres. Collect the neighbors of the points in the smallest neighbor sphere, search for the optimal set among them, and throw out the $m+1$ points in the sphere. Repeat the preceding step until all points are gone, but if a smallest neighbor sphere ever contains less than k points, we throw out its center immediately, since that point cannot possibly be in the optimal set. The size of the sets tested increases by a constant factor, but the number of sets decreases from n to $\lceil n/k \rceil$. Thus we achieve a savings in time of $O(k)$ over the naïve algorithm.

In general, we will be able to use rectilinear nearest neighbors, even for problems defined in the Euclidean metric; by our results above these

neighbors can be found in time $O(n \log n + mn)$ in the plane. In higher dimensions, we use Vaidya's $O(mn \log n)$-time algorithm [22].

Lemma 4.2. *Let μ be a measure having the property that the minimum measure k-point set is contained in the m nearest neighbors of each of its points, and let $f(m)$ be the time required to find the optimal k-point set among m points. Then, given a set of n points in $\mathcal{R}^d$, we can find the k-point subset minimizing μ, in time $O(mn \log n + nf(m)/k)$, or in time $O(n \log n + mn + nf(m)/k)$ if $d = 2$.* □

5 Finding Minimum Measure Sets

5.1 Perimeter

We first demonstrate our technique on the minimum perimeter k-point set problem. The problem is to find, given a set of n points in the plane, a set of k points for which the perimeter of the convex hull is minimized. This was previously solved in $O(k^2 n \log n + k^5 n)$ time by Dobkin *et al.* [10]; this was improved by Aggarwal *et al.* [2] to $O(n \log n + k^4 n)$. Eppstein *et al.* [14] describe a dynamic programming algorithm that solves the problem in time $O(kn^3)$; we use this algorithm as a subroutine.

Lemma 5.1. *If a point p is in the minimum perimeter k-point set, then the set is contained in the $O(k)$ nearest rectilinear neighbors of p.*

Proof: Let q be the farthest point from p in the optimal set. Then the entire set fits in a circle around p, of radius $|pq|$, and the perimeter must be at least $2|pq|$. But we can partition the circle into 16 squares of perimeter $|pq|$; if q is not among the $16k$ nearest points then some square must contain at least k points, and would supply a k-point set with smaller perimeter. □

Theorem 5.1. *We can find the minimum perimeter k-point subset of a set of n points in the plane, in time $O(n \log n + k^3 n)$.* □

This algorithm generalizes to minimize perimeter in any metric, but we can do better in L_∞. The minimum L_∞ perimeter k-point set is the set enclosable in the minimum perimeter axis-aligned rectangle. Aggarwal *et al.* [2] solve this problem in time $O(k^2 n \log n)$. We use their $O(n^3)$-time brute force algorithm as a subroutine.

Theorem 5.2. *We can find the minimum L_∞ perimeter k-point subset of a set of n points in the plane, in time $O(n \log n + k^2 n)$.* □

5.2 Circumradius

We now describe our algorithms for finding the k-point set contained in the smallest sphere, given a set of n points in $\mathcal{R}^d$. We improve previous time bounds, due to Aggarwal *et al.* [2], of $O(n \log n + k^2 n)$ in the plane and $O(n^{d+1})$ in higher dimensions. Their algorithms are based on higher order Voronoi diagrams.

First we develop a new algorithm to use as a subroutine within each neighbor set. Consider the related problem of placing a fixed-size sphere so that it covers the maximum number of a given set of points. Once we have a solution to this problem, we can apply Megiddo's parametric search technique [19] to find the smallest sphere whose optimal placement covers k (or more) points.

Lemma 5.2. *We can find the minimum circumradius k-point subset of a set of n points in $\mathcal{R}^d$, in time $O(n^d \log^2 n)$, or in time $O(n^2 \log n)$ if $d = 2$.*

Proof: First consider the sphere placement problem. We fix each set of $d-1$ points and rotate a sphere around its affine hull, stopping whenever a point enters or leaves the sphere. Each sweep requires time $O(n \log n)$. Degenerate cases, where the optimal sphere cannot be forced to be tangent to d points, are handled in total time $O(n^{d-1})$. Thus, the entire sweep algorithm takes time $O(n^d \log n)$. In the plane, we can solve this problem in time $O(n^2)$, using a more complicated algorithm developed by Chazelle and Lee [7].

To find minimum circumradius sets, we apply parametric searching with Cole's trick [8]. Our sweep algorithm can be parallelized to run in $O(\log n)$ steps on $O(n^d)$ processors. Thus, the total time is $O(n^d \log^2 n)$ in general, and $O(n^2 \log n)$ in the plane. □

Lemma 5.3. *If a point p is in the minimum circumradius k-point set, then the set is contained in the $O(k)$ nearest neighbors of p.*

Proof: Let R be the optimal circumradius. The minimum circumradius set is contained in a sphere of radius $2R$, centered at p. This sphere can

be covered by a constant number of spheres of radius R, none of which can contain more than k points. □

Theorem 5.3. *We can find the minimum circumradius k-point subset of a set of n points in $\mathcal{R}^d$, in time $O(kn \log n + k^{d-1} n \log^2 k)$, or in time $O(n \log n + kn \log k)$ if $d = 2$.* □

5.3 Diameter

The diameter of a set is the largest distance between any two points in the set. In the plane, Aggarwal *et al.* [2] show how to find the minimum diameter k-point set, in time $O(n \log n + k^{2.5} n \log k)$. It is noteworthy that the problem can even be solved in polynomial time: the diameter must be one of only $O(n^2)$ point distances, but it is not clear how to find a large set of points having a given distance as diameter. Indeed, we know of no fully polynomial algorithm for this problem in dimensions greater than two, so we are forced to use a brute force approach.

Lemma 5.4. *If p is in the minimum diameter k-point set, the set is contained in the $O(k)$ nearest neighbors of p.*

Proof: Let D be the optimal diameter. The minimum circumradius set is contained in a sphere of radius D, centered at p. This sphere can be covered by a constant number of spheres of diameter D, none of which can contain more than k points. □

Theorem 5.4. *We can find the minimum diameter k-point subset of a set of n points in R^d, in time $O(kn \log n + 2^{O(k)} n)$, for all $d > 2$.* □

We can do considerably better than this in the plane. Aggarwal *et al.* solve this problem by reducing it to one of finding a maximum independent set in a certain bipartite graph. For bipartite graphs, the maximum independent set and maximum matching are closely related (their cardinalities add to the size of the point set) so matching techniques can be applied to this problem.

We improve on the algorithm of Aggarwal *et al.* by solving a *dynamic* matching problem. Given a point set S, and a distance D, let the graph $G_D(S)$ be defined as follows. The vertices of $G_D(S)$ are simply the points in S. An edge (p, q) will exist in the graph exactly when $|pq| > D$; *i.e.*, the graph connects points that are sufficiently far apart.

Lemma 5.5. *Given a set S of n points, and a maximum matching in $G_D(S)$, we can insert or delete a single point in S, and update the maximum matching, in time $O(n \log n)$.*

Proof: The update can only change the matching cardinality by one. If the update is a deletion of a matched point, we remove its edge from the matching and mark its mate as unmatched. Then whether the update is an insertion or a deletion, the remaining problem is to find a single *alternating path* connecting two unmatched points. If such a path is found, the matching size can be increased by changing the unmatched edges in it to matched edges, and vice versa.

We will go through a process of marking points as *odd* or *even*. A point is labeled odd (even) if it can be reached from an unmatched vertex by an alternating path of odd (even) length. In each case we remember the last edge on the path, so that the entire path can be reconstructed quickly. Once we have performed this labeling, the existence of an alternating path can be tested by testing if any two even points share an edge. This can be done in $O(n \log n)$ time by finding the farthest pair of even points.

We will maintain a data structure for a point set P with the following operations: (1) given point p, find some point in P farther than D from p, or report that no such point exists; (2) delete a given point from P. As noted by Aggarwal *et al.* [2], these operations can be performed in $O(\log n)$ amortized time using the *circular hull* data structure of Hershberger and Suri [18].

We start the labeling process by marking each unmatched point as even (it has a zero length path to an unmatched point). We build the data structure above, letting P consist of all unmarked points (initially, that is simply the matched points). We then *process* each marked point in turn, maintaining a queue of points that require processing. Processing an odd point consists simply of marking its match even, adding it to the queue, and removing it from P. We process the even points as follows. While an unmarked point adjacent to the even point exists, we mark it odd, add it to the queue, and remove it from P. Such a point can be found using the find operation described above.

Once the queue is empty, all points are either marked or unreachable via an alternating path. The number of data structure operations is $O(n)$, as each find operation either discovers a new point to be marked and removed from P, or it is the last such operation performed in processing a given

point. Therefore the total time used is $O(n \log n)$. □

Lemma 5.6. *We can find the minimum diameter k-point subset of a set of n points in the plane, in time $O(n^3 \log^2 n)$.*

Proof: There are $O(n^2)$ possible diameters; we select among them using binary search. To test a given diameter D, we test each point p separately to see whether there is some k-point set with diameter $|pq|$ shorter than D. If so, the set is contained in the *lune* formed by intersecting two circles of diameter D, one centered on p and one centered at distance D from p. We sweep a lune around p, covering in turn $O(n)$ different point sets; we must test if any of these sets contains a small diameter k-point subset.

As noted by Aggarwal *et al.* [2], if S is the point set contained in a given lune, then $G_D(S)$ is bipartite, and a subset of S with diameter less than D is exactly an independent set in $G_D(S)$. If M is the maximum matching in $G_D(S)$, the size of the maximum independent set is $|S| - |M|$. Thus to test if there is a large subset with small diameter, we may compute this matching. We do this for all $O(n)$ positions of the lune around p, in time $O(n^2 \log n)$, using the dynamic matching algorithm of Lemma 5.5.

There are $O(n)$ points for which this must be done, so the time to test a single distance D is $O(n^3 \log n)$. The binary search used to find the optimal distance adds a further logarithmic factor to this bound. □

Theorem 5.5. *We can find the minimum diameter k-point subset of a set of n points in the plane, in time $O(n \log n + k^2 n \log^2 k)$.* □

5.4 L_∞ Diameter

The algorithms in the previous two sections generalize to circumradius and diameter in any metric, but we can make a significant improvement in L_∞. The minimum L_∞ diameter (equivalently, minimum L_∞ circumradius) k-point set is the set enclosable in the smallest axis-aligned hypercube. In the plane, Aggarwal *et al.* [2] give an algorithm for this problem, based on higher order L_∞ Voronoi diagrams, that takes time $O(k^2 n \log n)$.

Our approach is almost identical to that used to find minimum circumradius sets. We start with the problem of placing a fixed-size axis-aligned hypercube so that it covers the maximum number of points. Once we solve

this problem, we can parameterize it to find the smallest axis-parallel hypercube that covers at least k points. Instead of parametric search, we use a much simpler binary search among the possible L_∞ diameters.

Lemma 5.7. *We can find the minimum L_∞ diameter k-point subset of a set of n points in $\mathcal{R}^d$, in time $O(n^{d/2}\log^2 n)$.*

Proof: Finding the optimal placement of a hypercube is equivalent to finding the deepest point in an arrangement of hypercubes. We can easily adapt an algorithm of Overmars and Yap [21], originally applied to Klee's measure problem, to find the deepest point in an arrangement of axis-aligned boxes in time $O(n^{d/2}\log n)$.

To find the optimal hypercube size, we search along each coordinate axis as follows. We sort the points by the appropriate coordinate, and define a triangular matrix M of coordinate differences. These differences are potential L_∞ diameters. We will not actually build M, since that would require time $\Omega(n^2)$, but we can access any entry in constant time. We binary search through M for the optimal diameter. Since the rows and columns of M are sorted, we can select any element in time $O(n\log n)$ [16]. Thus, each step of the search is dominated by Overmars and Yap's algorithm, and the entire search requires time $O(n^{d/2}\log^2 n)$. □

Theorem 5.6. *We can find the minimum L_∞ diameter k-point subset of a set of n points in $\mathcal{R}^d$, in time $O(kn\log n + k^{d/2-1}n\log^2 k)$, or in time $O(\min\{n\log n + kn, n\log^2 n\})$ if $d = 2$.* □

5.5 Variance

The variance of a set of points is defined as the sum of the squares of the distances between pairs of points, divided by the number of points in the set. Equivalently, the variance is the sum of the squares of the distances from each point to the centroid of the set [2].

Lemma 5.8. *If a point p is in the minimum variance k-point set, then the set is contained in the $O(k^{d/2+1})$ nearest neighbors of p.*

Proof: Let V and R be the variance and circumradius of the minimum variance set, and let p be any point in the set. We easily verify that $2R^2 <$

$V \leq kR^2$. The set is contained in a sphere centered at p with radius $2R$. We can cover the sphere with $O(k^{d/2})$ spheres of radius $R\sqrt{2/k}$. If any of these spheres contain k points, their variance is at most $2R^2$, which is less than V. □

Aggarwal *et al.* [2] prove that the minimum variance k-point set corresponds to one of the cells in the kth order Voronoi diagram of the original n points and derive an algorithm that uses time $O(n \log n + k^2 n)$ in the plane. Agarwal and Matoušek [1] recently discovered an algorithm for constructing planar order-k Voronoi diagrams in time $O(kn^{1+\epsilon})$.[1] Combining their algorithm with our techniques, we can find minimum variance sets in the plane in time $O(n \log n + k^{2+\epsilon} n)$, which is slightly worse than the existing bound.

Lemma 5.9. *Let p be a point in the minimum variance k-point set, and let V be the set's variance. Suppose for some constant $c > 0$, the distance between p and the set's centroid is $c\sqrt{V/k}$. Then the set is contained in the $O(ck^{(d+1)/2})$ nearest Euclidean neighbors of p.*

Proof: Let S be the sphere, centered at the optimal set's centroid, which just contains the set, and let R be the radius of S. S contains exactly k points [2]. Then S is contained in a sphere centered at p with radius $R + 2c\sqrt{V/k}$. The space between the two spheres can be covered by $O(ck^{(d-1)/2})$ spheres of radius $\sqrt{V/k}$, none of which can contain k points. □

The two previous bounds are tight in the worst case. Consider a sphere S_1 of radius $\sqrt{k}$, containing a smaller sphere S_2 of radius $\sqrt{k}/2$ tangent to S_1. There is a cluster of $k-2$ points with arbitrarily small variance around the center of S_2, but excluding the center itself. The surface of S_2 and the space between the two spheres are both filled with as many clusters of $k/2$ points as possible, such that every two clusters have at least unit distance between them. One of these clusters contains the center of S_1; another contains the tangent point of the two spheres. The minimum variance set consists of the large cluster, the center of S_1, and the tangent point. For each point p in this set, every sphere centered at p that contains the set also contains $\Omega(k^{(d+1)/2})$ other points, and the set contains the $\Theta(k^{d/2+1})$th neighbor of the center of S_1.

[1] Throughout this paper, ϵ represents an arbitrarily small positive constant. Multiplicative constants hidden by the O-notation may depend on ϵ.

To find the minimum variance set quickly, we need to find a *center point* within radius $c\sqrt{V/k}$ of the optimal set's centroid, for some constant $c > 0$, so that we can search for the optimal set among its $O(ck^{(d+1)/2})$ nearest neighbors. We describe an algorithm for finding a set of $O(n/k)$ points which contains at least one center point.

Theorem 5.7. *We can find the minimum variance k-point subset of a set of n points in the plane, in time $O(k^{3/2}n\log n + k^{3/2+\epsilon}n)$.*

Proof: We begin by finding the $k/2$ neighbors to every point, in time $O(n\log n + kn)$. Repeatedly find the point p with the smallest neighbor sphere. If neither p nor any of the neighbors of p are already marked noncentral, mark p as a potential center point, and mark its neighbors as noncentral. Each central point marks $k/2$ noncentral points, so this process gives us $O(n/k)$ potential center points. The entire marking process requires time $O(n\log n + kn)$.

Most of the points in the minimal set are within $\sqrt{2V/k}$ of the centroid, so every point within this radius has at least $k/2$ neighbors within $2\sqrt{2V/k}$. Let p be one of these points. When the marking algorithm reaches p, one of two things happens. (1) We could mark p as a potential center point. (2) We could ignore p because p or one of its neighbors is marked noncentral, in which case some point within $5\sqrt{2V/k}$ of the centroid is already marked central. Thus, at least one of the potential center points is an actual center point.

After we find the potential center points, we find the $O(k^{3/2})$ nearest Euclidean neighbors of each potential center point in time $O(k^{3/2}n\log n)$. We then test each of the $O(n/k)$ neighbor sets in time $O(k^{5/2+\epsilon})$ using Agarwal and Matoušek's Voronoi diagram algorithm [1]. □

This matches or improves previous time bounds for all k in $O(n^\epsilon) \cap \Omega(\log^2 n)$. For smaller values of k, the $O(n\log n + k^2 n)$-time algorithm of Aggarwal *et al.* is faster. For larger values of k, Agarwal and Matoušek's Voronoi diagram algorithm is faster. Finally, for $k = \Omega(n^{1-\epsilon})$, the fastest algorithm is based on another Voronoi algorithm of Chazelle and Edelsbrunner [5] and runs in time $O(n^2\log^2 n)$.

Mulmuley describes an algorithm that constructs the kth order Voronoi diagram of a set of n points in $\mathcal{R}^d$, in time $O(k^{\lceil\frac{d+1}{2}\rceil}n^{\lfloor\frac{d+1}{2}\rfloor}\log n + k^d n^2)$ [20]. To find minimum variance sets in higher dimensions, we use Mulmuley's

algorithm as a subroutine within each neighbor set. We improve the previous time bound of $O(n^{d+1})$ [2].

Theorem 5.8. *We can find the minimum variance k-point subset of a set of n points in $\mathcal{R}^d$, in time $O(k^{(d+1)/2} n \log n + k^{v(d)} n \log k)$, where $v(d) = \frac{d^2+3d}{4}$ if d is even, and $\frac{d^2+4d-1}{4}$ if d is odd.* □

6 Dynamization

We now show how to turn our planar algorithms into dynamic data structures, that can maintain the minimum measure k-point set as points are inserted. Our algorithm is simply to maintain a data structure that can determine, for each new point, its $O(k)$ nearest neighbors. Then if that point is part of a set improving the previous optimum, that set will be a subset of these neighbors, and can be found using the methods already described.

Lemma 6.1. *Let μ be a measure having the property that the minimum measure k-point set is contained in the m nearest neighbors of each of its points, and let $f(m)$ be the time required to find the optimal k-point set among m points. Then in the plane, we can maintain the k-point set minimizing μ as points are inserted in time $O(f(m) + \log^2 n + m \log n)$ per insertion.*

Proof: We apply a standard dynamic-to-static reduction technique for decomposable searching problems [3] to the rectilinear nearest neighbor data structure of Lemma 3.3. □

Theorem 6.1. *We can maintain the minimum measure k-point set in the plane as points are inserted, with the following insertion times: $O(k^4 + \log^2 n)$ for perimeter, $O(k^3 + \log^2 n)$ for L_∞ perimeter, $O(k^2 \log k + \log^2 n)$ for circumradius, $O(k^3 \log^2 k + \log^2 n)$ for diameter, $O(k \log^2 k + log^2 n)$ for L_∞ diameter, and $O(k^{3+\epsilon} + \log^2 n)$ for variance.* □

We can dynamize our higher dimensional results in a similar manner, using a dynamic nearest neighbor data structure of Agarwal and Matoušek [1], with results that are just slightly better than brute force.

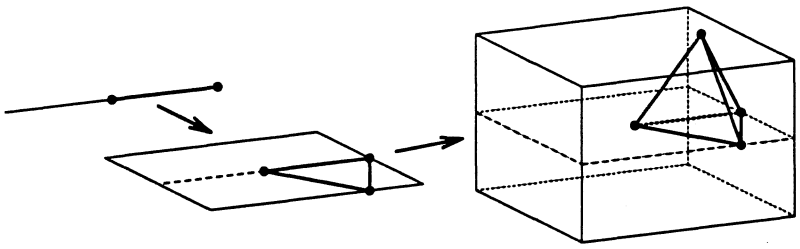

Figure 1. Extremal points, extremal simplices, and bounding boxes in $\mathcal{R}^3$

7 Minimizing Volume and Boundary Measure

Eppstein [13] proves that the minimum area k-point subset of a set of points in the plane is contained in the $O(k)$ nearest neighbors to the line segment connecting its two farthest points. In this section, we demonstrate a natural generalization of this result to arbitrary dimensions. We also generalize our results for finding minimum perimeter sets in the plane.

Let T be some r-dimensional polytope in $\mathcal{R}^d$, with $r < d$. Given a point p, we define the *orthogonal distance* from p to T to be the Euclidean distance from p to its orthogonal projection onto aff(T), which we denote p'. We call p an *orthogonal neighbor* of T if and only if $p' \in T$. We can compute the nearest orthogonal neighbors to any polytope with fixed complexity in linear time.

Given a set of points A in $\mathcal{R}^d$, and an arbitrary point $p_0 \in A$, we define the series of *extremal points*, *extremal simplices*, and *bounding boxes* of A with respect to p_0, denoted p_i, S_i, and B_i, respectively. While these sequences depend on the initial point p_0, the properties we derive hold for all initial points.

We define $B_0 = S_0 = p_0$. For each $i \leq d$, p_i is the point in A farthest from the affine hull of S_{i-1}. $S_i = \text{conv}(S_{i-1}, p_i)$. B_i is the convex hull of two copies of B_{i-1}, one containing p_i and one and equal distance from B_{i-1} in the opposite direction, situated so that $B_{i-1} \subset B_i$, and adjacent facets of B_i meet at right angles. For any set A, we have $S_d(A) \subset \text{conv}(A) \subset B_d(A)$. See Figure 1.

Volume and boundary measure share the following property. For some constant r, the minimum measure k-point set is contained in the m nearest orthogonal neighbors to the bounding box of its first r extremal points (with respect to any point in the set). For measures with this property, we have

the following algorithm outline for finding minimum measure sets. For each set of r points, there are r possible bounding boxes. For each box, we find its m nearest orthogonal neighbors, and search for the minimum measure set among them.

Lemma 7.1. *Let μ be a measure having the property that the minimum measure k-point set A is contained in the m nearest orthogonal neighbors of $B_r(A)$, and let $f(m)$ be the time required to find the optimal k-point set among m points. Then, given a set of n points in $\mathcal{R}^d$, we can find the k-point subset minimizing μ, in time $O(n^{r+1} + n^r f(m))$.* □

We know of no fully polynomial time algorithm to find minimum volume or boundary measure sets, except in the plane [14, 13]. A naïve algorithm runs in time $O(\binom{n}{k} k^{\lfloor d/2 \rfloor})$, by explicitly computing the convex hull of every k-point subset [4]. We use this algorithm as a subroutine.

Throughout this section, we let $|A|$ and $|\partial A|$ denote the volume and boundary measure of the convex hull of A. The following lemma relates the volumes of bounding boxes and extremal simplices.

Lemma 7.2. *For all $A \subset \mathcal{R}^d$, $|B_d(A)| = 2^d d! |S_d(A)|$.*

Proof: The volume of a d-dimensional cone is bh/d, where b is the $(d-1)$-dimensional measure of the base and h is the distance between the apex and the affine hull of the base. The volume of a d-dimensional box with the same base measure and height is bh.

We prove the lemma by induction. The lemma holds (trivially) when $d = 0$. Let h_d denote the distance between p_d and $\text{aff}(S_{d-1})$. Using the volume formulae above, we have $|S_d| = h_d |S_{d-1}|/d$ and $|B_d| = 2h_d |B_{d-1}|$. Therefore, $|B_d|/|S_d| = 2d|B_{d-1}|/|S_{d-1}|$. The closed form follows directly from the inductive hypothesis. □

7.1 Boundary Measure

Given a set A in $\mathcal{R}^d$, we define its *bounding cylinder* $C(A)$ as the set of points no farther orthogonally from $B_{d-2}(A)$ than $p_{d-1}(A)$, and we define $B'(A)$ as the smallest box containing $C(A)$. We have $S_{d-1}(A) \subset \text{conv}(A) \subset C(A) \subset B'(A)$. See Figure 2.

The following lemma relates the boundary measure of any set A with the boundary measure of $B'(A)$.

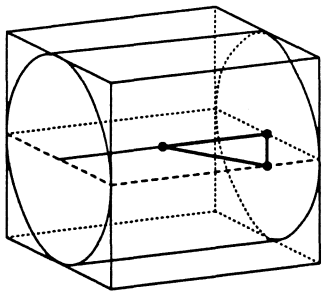

Figure 2. A bounding cylinder and its box in $\mathcal{R}^3$

Lemma 7.3. *For all $A \subset \mathcal{R}^d$, $|\partial B'(A)| < 2^{d-1}d!|\partial A|$.*

Proof: B' has $2d$ facets. Four of the facets have measure equal to $|B_{d-1}|$. Let h_i denote the distance between p_i and $\text{aff}(S_{i-1})$. Since $h_{d-1} < h_i$ for all $i < d-1$, the rest of the facets of B' have measure smaller than $|B_{d-1}|$. Therefore, $|\partial B'| < 2d|B_{d-1}| = 2^d d|S_{d-1}|$, by Lemma 7.2. The lemma follows from the observation that $|\partial A| > 2|S_{d-1}(A)|$. □

Lemma 7.4. *The minimum boundary measure set A is contained in the $O(k)$ nearest orthogonal neighbors to $B_{d-2}(A)$.*

Proof: Let $s(d)$ denote $\lceil 2 \cdot d!^{\frac{1}{d-1}} \rceil$. We divide $B'(A)$ into $s(d)^d$ congruent pieces by slicing parallel to each opposite pair of facets $s(d)$ times. Each piece has boundary measure $|\partial B'(A)|/s(d)^{d-1} < |\partial B'(A)|/2^{d-1}d!$. By the previous lemma, this is less than $|\partial A|$, so no piece can contain more than $k-1$ points. Thus, $B'(A)$ contains at most $s(d)^d(k-1) = O(k)$ points. Since $C(A) \subset B'(A)$, $C(A)$ also contains $O(k)$ points. The points in $C(A)$ are the nearest orthogonal neighbors of $B_{d-2}(A)$. □

Theorem 7.1. *We can find the minimum boundary measure k-point subset of a set of n points in $\mathcal{R}^d$, in time $O(n^d + 2^{O(k)}n^{d-1})$.* □

We can generalize L_∞ perimeter into higher dimensions as follows. We define the L_∞ *boundary measure* of a set A as the boundary measure of the smallest axis-parallel hyperrectangle enclosing A. Using techniques similar to those used to prove the previous theorem, we have the following result.

Theorem 7.2. *We can find the minimum L_∞ boundary measure k-point subset of a set of n points in $\mathcal{R}^d$, in time $O(n^d + k^{2d-1}n^{d-1})$.* □

7.2 Volume

While it is possible to derive a relatively efficient minimum volume algorithm using orthogonal neighbors, we can do better if we use *vertical* neighbors, as Eppstein [13] does in his minimum-area algorithm. We say that a point p is a vertical neighbor of a polytope T if the line through p parallel to the dth coordinate axis intersects T.

Given a set A in $\mathcal{R}^d$ and an arbitrary point $p_0^{\mathrm{v}} \in A$, we define a series of *vertical* extremal points, extremal simplices, and bounding boxes, which we denote p_i^{v}, S_i^{v}, and B_i^{v}, respectively. As before, we define $S_0^{\mathrm{v}} = B_0^{\mathrm{v}} = p_0^{\mathrm{v}}$. For all $1 \leq i \leq d$, p_i^{v} is the point in A farthest along the ith coordinate axis from $\mathrm{aff}(S_{i-1}^{\mathrm{v}})$. $S_i^{\mathrm{v}} = \mathrm{conv}(S_{i-1}^{\mathrm{v}}, p_i^{\mathrm{v}})$. B_i^{v} is the convex hull of two copies of B_{i-1}^{v}, displaced equal distances in opposite directions along the ith coordinate axis, one containing p_i^{v}. For any set A, we have $S_d^{\mathrm{v}}(A) \subset A \subset B_d^{\mathrm{v}}(A)$. Clearly, Lemma 7.1 still holds if we consider vertical neighbors to $B_r^{\mathrm{v}}(A)$ instead of orthogonal neighbors to $B_r(A)$, and Lemma 7.2 also applies to vertical bounding boxes and extremal simplices.

Lemma 7.5. *The minimum volume set A is contained in the $O(k)$ nearest vertical neighbors of $B_{d-1}^{\mathrm{v}}(A)$.*

Proof: We divide $B_d^{\mathrm{v}}(A)$ into $2^d d!$ congruent convex pieces. By Lemma 7.2, each piece has the same volume as $S_d^{\mathrm{v}}(A)$. Since $|A| \geq |S_d^{\mathrm{v}}(A)|$, no piece can contain more than k points. The points in $B_d^{\mathrm{v}}(A)$ are the nearest vertical neighbors of $B_{d-1}^{\mathrm{v}}(A)$. □

We now describe an efficient algorithm for finding nearest vertical neighbors to $(d-1)$-dimensional boxes. First consider the simpler problem of finding nearest neighbors to hyperplanes. We use geometric duality to transform the problem into finding, in an arrangement of hyperplanes, the k closest hyperplanes above some query point. Vertical point-hyperplane distances in the dual space are the same as the corresponding vertical hyperplane-point distances in the primal space. Thus, we can solve this problem by *vertical ray-shooting* in the dual space. We will use the following result of Agarwal and Matoušek [1].

Lemma 7.6 (Agarwal and Matoušek [1]). *We can preprocess a set of n points in $\mathcal{R}^d$, in time $O(n^{\lfloor d/2 \rfloor + \varepsilon})$, so that the k nearest neighbors to a query hyperplane can be found in time $O(k \log n)$.* □

We make use of a technique developed by Chazelle *et al.* [6] for answering simplex range queries. Given a data structure to solve some arbitrary geometric problem, they build on top of it another structure that limits the problem to the points within an arbitrary halfspace. The resulting data structure can be built in time $O(n^{d+\epsilon} + P(n))$, where $P(n)$ is the preprocessing time required for the original structure; and queries are answered in time $O(Q(n)\log n)$, where $Q(n)$ is the original query time.

Lemma 7.7. *We can preprocess a set of n points in $\mathcal{R}^d$, in time $O(n^{d-1+\epsilon})$, so that the k nearest vertical neighbors to a query $(d-1)$-dimensional box can be found in time $O(k\log^{d+1} n)$.*

Proof: It suffices to find vertical neighbors to simplices, since every box can be split into a constant number of simplices, and neighbors can be merged in time $O(k)$. We build d levels of the halfspace data structure of Chazelle *et al.* one for each $(d-2)$-face of the query simplex, on top of Agarwal and Matoušek's vertical ray shooting data structure. Since all the hyperplanes are vertical, we actually apply the halfspace construction in $\mathcal{R}^{d-1}$, by ignoring the dth coordinate of every point. □

Theorem 7.3. *We can find the minimum volume k-point subset of a set of n points in $\mathcal{R}^d$, in time $O(kn^d\log^{d+1} n + 2^{O(k)}n^d)$.* □

8 Finding Minimal Convex Sets

We achieve results for finding minimal k-vertex convex polygons and polytopes by applying one of the oldest results in combinatorial geometry.

Lemma 8.1 (Erdős and Szekeres [15]). *Given $ES_2(k) \le \binom{2k-4}{k-2}+1$ points in general position in the plane, some k points form the vertices of a convex polygon.* □

Lemma 8.2. *Given $ES_d(k) \le \binom{2k-4}{k-2} + 1$ points in general position in $\mathcal{R}^d$, some k points form the vertices of a convex polytope.*

Proof: Project a set of $ES_2(k)$ points in $\mathcal{R}^d$ down to any plane. By Lemma 8.1, some k points in the projection form a convex polygon. The preimage of those k points forms a convex polytope in $\mathcal{R}^d$. □

This gives us an upper bound of $ES_d(k) = O(4^k)$. Erdős and Szekeres also conjecture that $ES_2(k) = 2^{k-2} + 1$ and prove that this is a lower bound. Tightening the bounds on this function remains one of the outstanding open problems in combinatorial geometry [9]. We know of no bounds on $ES_d(k)$ other than those stated here, but it is clear that the function decreases with increasing d. Clearly, any reduction of the upper bound on $ES_d(k)$ would speed up our algorithms.

Using the previous lemma, we can generalize all of our results, both static and dynamic, to find minimum measure convex sets. The resulting time bounds have the same dependence on n as the corresponding k-point set results, but with an exponential dependence on k.

For each of the measures we consider, if the minimum measure set is contained in the m nearest neighbors to each of its points, then the minimum measure *convex* set is contained in the $O(m4^k/k)$ nearest neighbors to each of its points. Our proof technique is identical to the one used for our earlier neighbor counting lemmas. We describe a convex body, typically a sphere, that contains the minimum measure set. We then divide the body into small pieces, such that if any piece contains $O(4^k)$ pieces, then it necessarily contains a k-point convex set with smaller measure then the original minimum measure set.

Theorem 8.1. *We can find the convex k-gon with minimum perimeter or L_∞ perimeter, in time $O(n \log n + 2^{6k} n)$. We can maintain the convex k-gon with minimum perimeter or L_∞ perimeter as points are inserted, in time $O(2^{6k} k + \log^2 n)$ per insertion.*

Proof: The minimum perimeter convex k-gon is contained in the $O(4^k)$ nearest neighbors to each of its points. Eppstein *et al.* [14] describe a dynamic programming algorithm to find minimum perimeter k-gons in time $O(kn^3)$. Using their algorithm as a subroutine, we achieve a static time bound of $O(n \log n + k(4^k)^3 n/k) = O(n \log n + 2^{6k} n)$. The dynamic time bound follows directly from Lemma 6.1. Our algorithms work under any metric. □

Theorem 8.2. *We can find the convex k-gon with minimum circumradius or L_∞ diameter, in time $O(n \log n + 2^{10k} n/k)$. We can maintain the convex k-gon with minimum circumradius or L_∞ diameter as points are inserted, in time $O(2^{10k} + \log^2 n)$ per insertion.*

Measure	Static time bound	Dynamic time bound
perimeter	$O(n \log n + 2^{6k} n)$	$O(2^{6k} k + \log^2 n)$
L_∞ perimeter	$O(n \log n + 2^{6k} n)$	$O(2^{6k} k + \log^2 n)$
circumradius	$O(n \log n + 2^{10k} n/k)$	$O(2^{10k} + \log^2 n)$
L_∞ diameter	$O(n \log n + 2^{10k} n/k)$	$O(2^{10k} + \log^2 n)$
diameter	$O(n \log n + 2^{2k^2+O(k)} n)$	$O(2^{2k^2+O(k)} + \log^2 n)$
variance	$O(n \log n + 2^{2k^2+k \lg k+O(k)} n)$	$O(2^{2k^2+k \lg k+O(k)} + \log^2 n)$

Table 3. New results for finding minimum measure convex k-gons, given n points in the plane. (Compare Table 1.)

Proof: The minimum circumradius convex k-point set is contained in the $O(4^k)$ nearest neighbors to each of its points. Edelsbrunner and Guibas [11] describe an algorithm that finds, given a set of n points, the largest (cardinality) convex subset that includes a given leftmost point, in time $O(n^2)$. For each point p and each circumcircle containing it, rotate the points within the circle so that p is leftmost, and find the largest convex subset containing p. Since each point is on $O(n^2)$ circumcircles, the resulting algorithm finds the minimum circumradius convex k-gon in time $O(n^5)$. We use this algorithm as a subroutine. □

We are unable to generalize our planar diameter and variance algorithms, or any of our algorithms in higher dimensions, to find minimal convex sets. Consequently, we must use brute force within the neighbor sets, and our resulting time bounds are heavily exponential in k. Nevertheless, for sufficiently small k, our algorithms are faster than brute force. We summarize our planar results in Table 3, and our higher dimensional results in Table 4.

9 Conclusions and Open Problems

We have presented several algorithms for finding minimum measure k-point sets under a variety of measures, both in the plane and in higher dimensions. Our results are based on a common method. Given a set of points, we compute the nearest neighbors to each subset of r points, where r is a small constant determined by the relevant measure, and then search within each neighbor set using another algorithm. For most of the measures we have examined, $r = 1$. For these measures, we can reduce the number of neighbor

Measure	Time bound
circumradius	$O(2^{2k}n\log n + 2^{2k^2+O(k)}n)$
diameter	$O(2^{2k}n\log n + 2^{2k^2+O(k)}n)$
L_∞ diameter	$O(2^{2k}n\log n + 2^{2k^2+O(k)}n)$
variance	$O(4^k k^{(d-1)/2}n\log n + 2^{2k^2+\frac{d-1}{2}k\lg k+O(k)}n)$
boundary measure	$O(n^d + 2^{2k^2+O(k)}n^{d-1})$
L_∞ boundary measure	$O(n^d + 2^{2k^2+O(k)}n^{d-1})$
volume	$O(2^{2k}n^d\log^{d+1} n + 2^{2k^2+O(k)}n^d)$

Table 4. New results for finding minimum measure k-vertex convex polytopes, given n points in $\mathcal{R}^d$, for all $d > 2$. (Compare Table 2.)

sets to search down to $O(n/k)$ by finding neighbors of neighbors. Our planar results were achieved through the use of a new algorithm that finds the m nearest rectilinear neighbors to n points, in time $O(n\log n + mn)$. We have also presented versions of our algorithms which maintain minimum measure sets as points are inserted and versions which find, or dynamically maintain, minimum measure convex sets.

Our results suggest several open problems. None of our results is known to be optimal. Faster algorithms, or matching lower bounds, would be interesting. In particular, is it possible to find higher-dimensional k-point sets with minimum diameter, volume, or boundary measure without resorting to brute force? Eppstein *et al.* [14] present a dynamic programming algorithm for solving a variety of minimum and maximum measure problems in the plane, but it seems highly unlikely that their approach can be adapted to higher dimensional problems. Similarly, we have been unable to generalize our minimum diameter algorithm, or the earlier algorithms of Aggarwal *et al.* [2], into higher dimensions.

Are there faster algorithms for finding nearest neighbors? An efficient technique for finding neighbors to $(d-2)$-flats might also lead to a faster minimum boundary measure algorithm. Finally, is it possible to find rectilinear neighbors to points in higher dimensions in $o(mn\log n)$ time?

References

[1] P. K. Agarwal and J. Matoušek. Ray shooting and parametric search.

In *24th ACM Symp. Theory Comput.*, pages 517–526, 1992.

[2] A. Aggarwal, H. Imai, N. Katoh, and S. Suri. Finding k points with minimum diameter and related problems. *J. Algorithms*, 12:38–56, 1991.

[3] J. L. Bentley and J. B. Saxe. Decomposable searching problems I: Static-to-dynamic transformation. *J. Algorithms*, 1:301–358, 1980.

[4] B. Chazelle. An optimal convex hull algorithm and new results on cuttings. In *32nd IEEE Symp. Found. Comput. Sci.*, pages 29–38, 1991.

[5] B. Chazelle and H. Edelsbrunner. An improved algorithm for constructing k^{th}-order Voronoi diagrams. *IEEE Trans. Comput.*, C-36:1349–1354, 1987.

[6] B. Chazelle, M. Sharir, and E. Welzl. Quasi-optimal upper bounds for simplex range searching and new zone theorems. In *6th ACM Symp. Comput. Geom.*, pages 23–33, 1990.

[7] B. M. Chazelle and D. T. Lee. On a circle placement problem. *Computing*, 36:1–16, 1986.

[8] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. ACM*, 34:200–208, 1987.

[9] H. P. Croft, K. J. Falconer, and R. K. Guy. *Unsolved Problems in Geometry.* Springer-Verlag, 1990.

[10] D. P. Dobkin, R. L. Drysdale, and L. J. Guibas. Finding smallest polygons. In *Advances in Computing Research, Vol. 1*, pages 181–214. JAI Press, 1983.

[11] H. Edelsbrunner and L. J. Guibas. Topologically sweeping an arrangement. *J. Comput. Syst. Sci.*, 38:165–194, 1989.

[12] D. Eppstein. Persistence, offline algorithms, and space compaction. Technical Report 91-54, Dept. Information and Comput. Sci., U. C. Irvine, 1991.

[13] D. Eppstein. New algorithms for minimum area k-gons. In *3rd ACM-SIAM Symp. Discrete Algorithms*, pages 83–88, 1992.

[14] D. Eppstein, M. Overmars, G. Rote, and G. Woeginger. Finding minimum area k-gons. *Discrete Comput. Geom.*, 7:45–58, 1992.

[15] P. Erdős and G. Szekeres. A combinatorial problem in geometry. *Compositio Math.*, 2:463–470, 1935.

[16] G. N. Frederickson and D. B. Johnson. The complexity of selection and ranking in $X + Y$ and matrices with sorted rows and columns. *J. Comput. Syst. Sci.*, 24:197–208, 1982.

[17] F. W. Fredman and D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. In *31st IEEE Symp. Found. Comput. Sci.*, pages 719–725, 1990.

[18] J. Hershberger and S. Suri. Finding tailored partitions. In *5th ACM Symp. Comput. Geom.*, pages 255–265, 1989.

[19] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30:852–865, 1983.

[20] K. Mulmuley. Output sensitive construction of levels and Voronoi diagrams in R^d of order 1 to k. In *22nd ACM Symp. Theory Comput.*, pages 322–330, 1990.

[21] M. H. Overmars and C.-K. Yap. New upper bounds in Klee's measure problem. *SIAM J. Comput.*, 20:1034–1045, 1991.

[22] P. M. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete Comput. Geom.*, 4:101–115, 1989.