New Algorithms for Minimum-Measure Simplices and One-Dimensional Weighted Voronoi Diagrams

David Eppstein and Jeff Erickson

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

Tech. Report 92-55

June 18, 1992

Abstract

We present two new algorithms for finding the minimum-measure simplex determined by a set of n points in $\mathcal{R}^d$, for arbitrary $d \geq 2$. The first algorithm runs in time $O(n^d \log n)$ using $O(n)$ space. The only data structure used by this algorithm is a stack. The second algorithm runs in time $O(n^d)$ using $O(n^2)$ space, which matches the best known time bounds for this problem in all dimensions and exceeds the previous best space bounds for all $d > 3$. We also present a new optimal algorithm for building one-dimensional multiplicatively weighted Voronoi diagrams that runs in linear time if the points are already sorted.

1 Introduction

The minimum-measure simplex problem can be stated as follows. Given a set of n points in $\mathcal{R}^d$, find a set of $d+1$ points whose convex hull has least measure. This problem is closely related to many other problems in computational geometry, such as detection of degeneracies.

The first non-trivial solution to the two-dimensional version of this problem is due to Dobkin and Munro [8], who discovered an $O(n^2 \log^2 n)$-time and $O(n^2 \log n)$-space algorithm, using a structure that allows rapid access to any projection of the point set. Edelsbrunner and Welzl [13] present an algorithm that uses time $O(n^2 \log n)$ and space $O(n)$. Their algorithm works by sweeping the arrangement of lines dual to the input points with a vertical line. Chazelle, Guibas, and Lee [5] and Edelsbrunner, O'Rourke, and Seidel [11] independently discovered an algorithm that uses time and space $O(n^2)$. Their algorithm builds the entire dual arrangement and then finds the closest edge to each vertex. This technique generalizes immediately into higher dimensions, with time and space bounds of $O(n^d)$ in any $\mathcal{R}^d$ (but see [12]). The best known result in the plane is due to Edelsbrunner and Guibas [10], who discovered an $O(n^2)$-time, $O(n)$-space algorithm which works by sweeping the dual arrangement with a pseudoline. Their technique was generalized to three dimensions by Anagnostu, Guibas, and Polimenis [2], resulting in an algorithm which uses time $O(n^3)$ and space $O(n^2)$.

We present an extremely simple algorithm that finds the minimum-measure simplex in time $O(n^d \log n)$ and space $O(n)$, for any $d \geq 2$. The only data structure this algorithm uses is a stack. A simple modification makes the algorithm run in time $O(n^d)$ and space $O(n^2)$. This matches the best known time bounds in all dimensions, up to constant factors, and exceeds the previous best known space bounds, for all $d > 3$. Our algorithm builds several two-dimensional arrangements rather than a single d-dimensional arrangement. Consequently, the constants hidden in the O-notation are smaller for our algorithm than for previous algorithms. To our knowledge, these are the first non-trivial minimum-measure simplex algorithms to use the same amount of space in all dimensions.

We also present a new algorithm for constructing the multiplicatively weighted Voronoi diagram of n points on a line in time $O(n \log n)$ and space $O(n)$, which is optimal. Two other optimal algorithms for this problem are known, both due to Aurenhammer [3, 4]. Unlike either of these algorithms, our algorithm requires only linear time if the points are already sorted. Similar results have been obtained in the plane for unweighted Voronoi diagrams when the points are sorted in various ways. Chew and Fortune [6] present an algorithm to build the orthogonal (L_1 metric) Voronoi diagram in time $O(n \log\log n)$, if the points are sorted separately by their x and y-coordinates. Eppstein [14] extended this result to solve the orthogonal post office problem for n sites and m queries in time $O((m+n)\log\log m)$. Aggarwal *et al.* [1] discovered a linear-time algorithm for building the (Euclidean) Voronoi diagram of the vertices of a convex n-gon given in, say, clockwise order. Most recently, Djidjev and Lingas [7] present an linear-time algorithm which builds the Voronoi diagram of a set of n points which have the same order sorted by either of their coordinates. Ours is the first result which shows that sorting can help in the construction of weighted Voronoi diagrams.

2 A Simple Minimum-Area Triangle Algorithm

We present a very simple algorithm that finds, given a set of n points and a fixed vertex p in the plane, the two points that form the smallest triangle with p. To find the smallest triangle formed by three input points, it suffices to run this algorithm n times, once for each input point. Our algorithm sweeps the plane with a rotating line through the fixed vertex.

Throughout this section, we make the following assumptions, without loss of generality. Any of these assumptions can be relaxed without significantly complicating our algorithm.

- The fixed vertex is the origin, which we denote o.
- Some input point is on the positive x-axis.
- Each pair of points, possibly including the origin, defines a line with a unique slope. In particular, no three points are colinear. Standard perturbation arguments apply.
- All input points are in the upper half-plane $y \geq 0$. Otherwise, we reflect any points in the lower half-plane through the origin. This operation does not affect the area of any triangle defined by the origin and two input points.

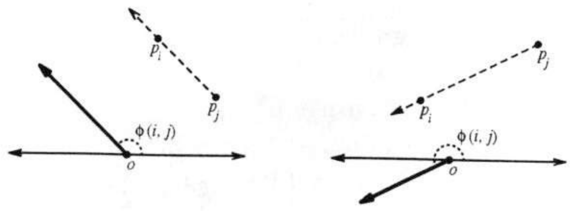

Figure 1. Defining the separator of p_i and p_j.

We will also assume that our input points $p_1, p_2, \ldots, p_n$ are sorted in the order in which a rotating sweep line would encounter them, starting at the positive x-axis and moving counterclockwise. We let θ_i denote the angle $\angle p_1 o p_i$, measured counterclockwise from the positive x-axis. For all $1 \leq i \leq n$, we have $0 \leq \theta_i < \pi$.

For each pair of input points p_i and p_j, such that $i > j$, we define their *separator* as the ray from the origin parallel to $\overrightarrow{p_j p_i}$. See Figure 1. We let $\phi(i,j)$ denote the angle of the separator, measured counterclockwise from the positive x-axis. Note that $\phi(i,j) > \theta_i > \theta_j$; that is, the rotating sweep line encounters each separator after the points that define it. In particular, $\phi(i,j)$ may be greater than π. For each point p_i, except p_1, we define $\phi(i) = \min_{j<i} \phi(i,j)$. For all $2 \leq i \leq n$, we have $\theta_i < \phi(i) < \pi$. For consistency, we define $\phi(1)$ to be π.

Lemma 2.1. *Let p_i, p_j, and p_k be input points such that $i > j > k$. Then $|\triangle p_i o p_j| < |\triangle p_i o p_k|$ if and only if $\theta_i < \phi(j,k)$.* □

For each input point p_i, except p_1, we define its *partner* to be the point $q_i \in \{p_1, \ldots, p_{i-1}\}$ such that $|\triangle p_i o q_i|$ is minimized. To find the pair of points which form the smallest triangle with the origin, it suffices to find each point's partner. The following lemma characterizes partners in terms of separators.

Lemma 2.2. *For each $2 \leq i \leq n$, the partner of p_i is the point p_j with the largest index less than i, such that $\theta_i < \phi(j)$.*

Proof: Let p_j be p_i's partner. Clearly, $j < i$. Suppose $\theta_i > \phi(j)$. Then for some $k < j$, $\theta_i > \phi(j,k)$, and by Lemma 2.1, $|\triangle p_i o p_k| < |\triangle p_i o p_j|$. On the other hand, suppose $\theta_i < \phi(l)$, for some l strictly between i and j. Then $\theta_i < \phi(l,j)$, and by Lemma 2.1, $|\triangle p_i o p_l| < |\triangle p_i o p_j|$. In either case, we have a contradiction. □

```
Algorithm FindPartners:
begin
      push $p_1$
      for $i \leftarrow 2$ to $n$ begin
(1)       while $(\phi(s_1, s_2) < \theta_i)$
              pop
(2)       $p_{s_1}$ is $p_i$'s partner
(3)       while $(\phi(s_1, s_2) < \phi(i, s_1))$
              pop
(4)       push $p_i$
      end
end
```

Figure 2. Finding each point's partner (See Lemma 2.3.)

Finally, we present our algorithm, which we call FindPartners. See Figure 2. The algorithm maintains a stack of points, and we let p_{s_1} denote the top point on the stack, p_{s_2} the next point down, and so on. We start with just p_1 on the stack, and loop through the other points in counterclockwise order. At each iteration, we examine the stack contents in two passes, popping off points if they can no longer be partners. The point on top of the stack after the first pass is the current point's partner. After the second pass, we push the current point onto the stack and continue with the next iteration.

Lemma 2.3. *Given a set of n points in the plane, sorted by their angles about the origin, Algorithm* FindPartners *finds each point's partner in time and space $O(n)$.*

Proof: Since each point is pushed once and popped at most once, the algorithm clearly uses linear time and space.

Let p_j denote p_i's partner. By Lemmas 2.1 and 2.2, we have $\phi(p) > \theta_i > \phi(k, j)$, for all k strictly between i and j. Suppose that p_j is not on the stack during the ith iteration of step (2). It must have been popped off during the kth iteration of the loop, for some $i > k > j$. If p_j were popped in step (1), we would have $\phi(j) < \theta_k \leq \theta_i$. If p_j were popped in step (3), we would have $\phi(j) < \phi(k, j)$. In either case, we have a contradiction, so p_j must be on the stack.

We prove the a stronger property of the entire stack: at the ith iteration of step (2), the points on the stack are stored in increasing order of the areas of the triangles they form with p_i and the origin. Step (1) insures that the top two points on the stack are properly ordered. Assume inductively that the ordering property holds for the first k points on the stack. If $\triangle p_i o p_{s_k}$ were smaller than $\triangle p_i o p_{s_{k-1}}$, then we would have $\phi(s_{k-1}, s_k) < \theta_i < \phi(s_{k-2}, s_{k-1})$, and $p_{s_{k-1}}$ would have been popped off the stack in the s_{k-2}-th iteration of step (3). Therefore, all points on the stack are properly ordered, and in particular, the top point is p_i's partner. $\square$

Our algorithm is quite similar to one proposed by Galil and Giancarlo [15] to find row minima in triangular monotone matrices. In fact, we can solve our problem with their algorithm. If we define $A[i,j] = \log|\triangle p_i o p_j|$ for all $i < j$, the resulting triangular matrix A is monotone, and partners correspond exactly to row minima. However, since this matrix does not satisfy Galil and Giancarlo's "closest zero property", their algorithm would require time $O(n \log n)$. Klawe and Kleitman [16] present a triangular monotone matrix searching algorithm that runs in time $O(n\alpha(n))$, where $\alpha(n)$ is the functional inverse of Ackerman's function. Larmore [17] presents an optimal algorithm, whose time complexity is unknown, for the same problem. Our algorithm improves all these previous results for our particular class of matrices. In particular, even if Larmore's algorithm is linear, it is considerably more complicated than our algorithm.

To find the smallest triangle in general, we fix each input point in turn as the origin, sort the other points by their angles in time $O(n \log n)$, and call FINDPARTNERS.

Theorem 2.1. *Given a set of n points in the plane, the minimum-area triangle can be found in time $O(n^2 \log n)$ and space $O(n)$.* $\square$

It is possible to find all n sorted orders in $O(n^2)$ time and space by building a dual line arrangement [9]. The order of the points around a fixed point p is the same as the order in which their dual lines intersect the line dual to p. Thus, we can improve our running time at the expense of increased storage.

Theorem 2.2. *Given a set of n points in the plane, the minimum-area triangle can be found in time and space $O(n^2)$.* $\square$

3 Higher dimensions

Let S be a simplex in $\mathcal{R}^d$ with vertices p and q, and let F be one of the two facets of S that does not contain the edge $\overline{pq}$. Then the d-dimensional measure of S is $(|\overline{pq}||F| \sin\theta)/d$, where θ is the angle between the edge $\overline{pq}$ and the facet F. Let S' be the projection of S onto the hyperplane normal to $\overleftrightarrow{pq}$. The $(d-1)$-dimensional measure of S' is $|F| \sin\theta$. Thus, to find the smallest simplex with a particular fixed vertex p, we can fix each of the other points q in turn as another vertex and project the problem to the hyperplane normal to $\overleftrightarrow{pq}$. This projection immediately reduces the d-dimensional problem to a set of $O(n)$ $(d-1)$-dimensional subproblems. Applying the projection process to each of the subproblems recursively, we eventually get $O(n^{d-2})$ two-dimensional subproblems. Thus, to find minimum-measure simplices in higher dimensions, it suffices to fix all but two of the vertices and apply our two-dimensional algorithm.

Theorem 3.1. *Given a set of n points in $\mathcal{R}^d$, $d \geq 2$, the minimum-measure simplex can be found in time $O(n^d \log n)$ and space $O(n)$.* □

We can combine the sorting phases from sets of $O(n)$ two-dimensional subproblems, even though each subproblem deals with a distinct set of points, by considering the action of the algorithm in dual space. The rotating line in each two-dimensional subproblem can be viewed as a hyperplane H rotating about a fixed $(d-2)$-flat F. In dual space, this corresponds to a point H^* moving along a fixed line F^* through the dual hyperplane arrangement. As with the two-dimensional problem, the order of the points around F is the same as the order in which their dual hyperplanes intersect F^*. We could find all these orders by constructing the entire dual arrangement, but that is far too space-intensive. The algorithm described by the previous theorem effectively builds all of the one-dimensional subarrangements, applying the FINDPARTNERS algorithm to each one in turn. A faster approach is to build each two-dimensional subarrangement in $O(n^2)$ time and space [5, 11], and apply the sweep algorithm $O(n)$ times to each one.

Theorem 3.2. *Given a set of n points in $\mathcal{R}^d$, $d \geq 2$, the minimum-measure simplex can be found in time $O(n^d)$ and space $O(n^2)$.* □

This approach is very similar to the one used by Edelsbrunner and Guibas [10] to detect degeneracies in configurations of points in arbitrary dimensions. Rather than explicitly constructing each two-dimensional subarrangement, their algorithm applies a topological sweep to each one, searching for sets of three or more concurrent lines.

4 One-Dimensional Weighted Voronoi Diagrams

The minimum-area fixed-vertex triangle problem can be thought of as a weighted nearest neighbor problem on the unit circle as follows. For each input point p_i, we associate a weighted point $\hat{p}_i$ on the unit circle, positioned at the angle θ_i and having weight $r_i = |\overline{op_i}|$. The "distance" between two points $\hat{p}_i$ and $\hat{p}_j$ is $|\triangle p_i o p_j| = (r_i r_j \sin|\theta_i - \theta_j|)/2$. We can find find the nearest neighbor of each point $\hat{p}_i$ under this distance function by running the FINDPARTNERS algorithm twice: once clockwise and once counterclockwise.

With an almost trivial modification, we have an algorithm which, given a set S of n multiplicatively weighted points on a line, sorted by their positions, finds each point's nearest weighted neighbor in linear time and space. In this section, we describe a further modification that produces the multiplicatively weighted Voronoi diagram of these points, which we call *sites* in this context, within the same resource bounds.

We make use of a two-dimensional interpretation of one-dimensional weighted Voronoi diagrams which was originally presented by Aurenhammer [3]. We assume without loss of generality that the sites are on the x-axis. Each site p_i has position x_i and weight w_i. For each p_i, we define a wedge $W_i = \{(x, y): y \geq |x - x_i|/w_i\}$. The boundary of the union of these wedges is a monotone polygonal chain which we denote $C(S)$. The vertical projection of the vertices of $C(S)$ onto the x-axis is essentially the multiplicatively weighted Voronoi diagram of S. Aurenhammer builds $C(S)$ by splitting S into two subsets S_1 and S_2, recursively building $C(S_1)$ and $C(S_2)$, and merging them in linear time using an intersection algorithm of Nievergelt and Preparata [18]. We refer the interested reader to Aurenhammer's paper for further details and relevant proofs.

We split each wedge W_i into two halves along the line $x = x_i$, and let W_i^R and W_i^L denote the right and left halves, respectively. The boundary of the union of the right wedges is the *right chain*, denoted $C^R(S)$. See Figure 3. We define the left chain $C^L(S)$ symmetrically. Using Nievergelt

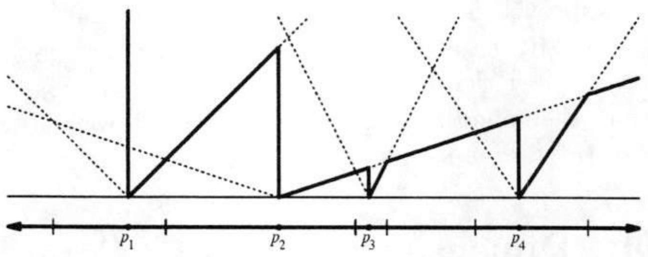

Figure 3. The right chain $C^R(S)$.

and Preparata's intersection algorithm, we can construct $C(S)$ from the left and right chains in linear time. The main result of this section is a linear-time algorithm that constructs the left and right chains. We explicitly describe the construction of the right chain; the left chain is built symmetrically.

Throughout this section, we make the following general position assumptions: no point on the x-axis is equidistant from more than a single pair of sites, no site is equidistant from any two other sites, and no two sites have the same weight. The algorithm we describe also assumes that the sites $S = \{p_1, p_2, \ldots, p_n\}$ are sorted from left to right.

For each pair of sites p_i and p_j, such that $i < j$, we define $X(i,j)$ to be the point on the x-axis to the right of both sites that has the same weighted distance from both sites. Specifically, if $w_i > w_j$, then $X(i,j) = (w_i x_j - w_j x_i)/(w_i - w_j)$. If $w_i < w_j$, then no such point exists, and we define $X(i,j) = \infty$. For each site p_i, we define $X(i) = \min_{j<i, w_j > w_i} X(i,j)$, where the minimum is taken over all sites to the left of p_i that have more weight. If there are no such sites, then we define $X(i) = \infty$.

Lemma 4.1. *Let p_i and p_j be two sites and let x be some arbitrary point on the x-axis such that $x > x_i > x_j$. Then $(x - x_i)/w_i < (x - x_j)/w_j$ if and only if $x < X(i,j)$.* □

Given an arbitrary point $x > x_1$, we define its *left neighbor* to be the site p_i to the left of x minimizing $(x - x_i)/w_i$. We can characterize the right chain $C^R(S)$ in terms of left neighbors as follows. For all $x > x_1$, p_i is the left neighbor of x if and only if $x_i < x$ and $(x, (x - x_i)/w_i) \in C^R(S)$.

Lemma 4.2. *For all $x > x_1$, the left neighbor of x is the site p_i with the largest index, such that $x_i < x < X(i)$.* □

```
Algorithm RightChain:
begin
        add $(x_1, \infty)$ to $C$
        add $(x_1, 0)$ to $C$
        push $p_1$
        for $i \leftarrow 2$ to $n$ begin
(1)         while $(x_i > X(s_1, s_2))$ begin
                add $(X(s_1, s_2), Y(s_1, s_2))$ to $C$
                pop
            end
(2)         add $(x_i, (x_i - x_{s_1})/w_{s_1})$ to $C$
            add $(x_i, 0)$ to $C$
(3)         while $(w_i > w_{s_1}$ or $X(i, s_1) > X(s_1, s_2))$
                pop
(4)         push $p_i$
        end
(5)     while stack has more than one point begin
            add $(X(s_1, s_2), Y(s_1, s_2))$ to $C$
            pop
        end
        add $(\infty, (\infty - x_{s_1})/w_{s_1})$ to $C$
end
```

Figure 4. Constructing the right chain $C^R(S)$.

Lemma 4.3. *Let p_i, p_j, and p_k be three sites such that $i < j < k$. Then $X(i,k)$ is between $X(i,j)$ and $X(j,k)$.* □

For each pair of sites p_i and p_j, such that $i < j$ and $w_i > w_j$, we define $Y(i,j)$ to be the the weighted distance from $X(i,j)$ to either p_i or p_j. Explicitly, $Y(i,j) = (x_j - x_i)/(w_i - w_j)$. We can characterize $(X(i,j), Y(i,j))$ as the intersection point of the two lines $y = (x - x_i)/w_i$ and $y = (x - x_j)/w_j$, when that point is above the x-axis.

Finally, we present our algorithm, which we call RightChain. See Figure 4. We sweep the x-axis from left to right, and maintain a stack as before. As each site is pushed onto or popped from the stack, a point is added to a polygonal chain C.

Lemma 4.4. *For all sites p_i, if the stack is empty just before p_i is pushed, then $X(i) = \infty$; otherwise, $X(i) = X(i, s_1)$.*

Proof: By induction. The lemma clearly holds when $i = 1$.

Suppose the stack is not empty. Since $w_i < w_{s_1}$, we know that $X(i, s_1)$ is finite. Suppose also that $X(i) = X(i, j) < X(i, s_1)$ for some $j \neq s_1$. By Lemma 4.3, we can easily show that $X(i) < X(j)$.

If $j < s_i$, then $X(s_1, j) < X(i, s_1)$ by Lemma 4.3. By the induction hypothesis, we have $X(s_1, s_2) = X(s_1)$. Then $X(s_1, s_2) < X(i, s_1)$, which implies that p_{s_1} was just removed from the stack. This is clearly a contradiction.

If $j > s_1$, then p_j cannot be on the stack. Since $x_i < X(i) < X(j)$, it must have been popped in the kth iteration of step (3), for some $k > j$. Thus, either $w_k > w_j$ or $X(j) < X(k, j)$. In either case, we have $X(i) = X(i, j) < X(k, j)$. Applying Lemma 4.3, we have $X(i, k) < X(i)$, which is another contradiction. Using a similar argument, we can show that if $X(i)$ is finite, then the stack cannot be empty. □

Lemma 4.5. *Given a set S of n weighted points on a line, sorted by their positions, Algorithm* RIGHTCHAIN *constructs the right chain $C^R(S)$ time and space $O(n)$.*

Proof: The time and space bounds are obvious.

Following the proof of Lemma 2.3, we know that after the ith iteration of the main loop, the points on the stack are ordered by increasing weighted distance from x_i. We can also show, using induction and Lemma 4.4, that the points p_{s_j} on the stack are ordered by increasing values of $X(s_j)$. Thus, for all p_{s_j} on the stack, we have $X(s_j) > x_i$.

We can easily conclude from these stack properties that C is monotone; that is, its vertices are added in order from left to right. Following the proof of Lemma 2.3, we conclude that the vertical edges of C are exactly the vertical edges of $C^R(S)$. All that is left to prove is that the non-vertical edges of C are correct.

Consider the non-vertical edges with endpoints between x_i and x_{i+1}, for some arbitrary $1 \leq i \leq n$. (For consistency, we define $x_{n+1} = \infty$.) These are exactly the edges added to C during the ith iteration of step (1), or in step (5) when $i = n$. We prove by induction that these edges are correct. The first edge starts at x_i and ends at either $X(i)$ or x_{i+1}. In either case, x_i is clearly the left neighbor of all points in the range covered by that edge.

Every edge except the first one starts at $X(j) = X(j,k)$ for some $k < j < i$, and ends at either $X(k)$ or x_{i+1}. Suppose for some x in that range, and for some $l \neq k$, that p_l is x's left neighbor. Since $X(j) < x < X(k)$, we have $l > k$ and $l \neq j$ by Lemma 4.2. If $j > l$, then $X(j,k) < x < X(l,k)$, and by Lemma 4.3, $X(j,l) < X(j)$, which is a contradiction. If $j < l$, then by Lemma 4.3, $X(j,k) < X(l,j)$, so p_j was popped off the stack before p_l was pushed on. By the inductive hypothesis, this is another contradiction. Thus, we can conclude that the edges of C are exactly the edges of $C^R(S)$, and the algorithm is correct. □

Theorem 4.1. *Given a set of n weighted points on a line, sorted by their positions, their multiplicatively weighted Voronoi diagram can be constructed in time and space $O(n)$, which is optimal.* □

Corollary 4.1. *Given a set of n weighted points on a line, their multiplicatively weighted Voronoi diagram can be constructed in time $O(n \log n)$ and space $O(n)$, which is optimal.* □

5 Conclusions

We have presented two new algorithms for finding minimum-measure simplices in spaces of dimension two or more. One algorithm uses linear space and is only a factor of $O(\log n)$ away from the best known time bound in all dimensions. The other algorithm matches the best known time bound and uses only $O(n^2)$ space in all dimensions. As a subroutine to these algorithms, we solve a particular triangular monotone matrix searching problem in linear time. We have also described a new optimal algorithm for constructing one-dimensional multiplicatively weighted Voronoi diagrams, which uses only linear time of the points are sorted.

The closely related problem of detecting degeneracies can be solved in time $O(n^d)$ using only linear space [10]. A minimum-measure simplex algorithm that runs within these bounds would be interesting. More importantly, we have no reason to believe that a time bound of $O(n^d)$ is even close to optimal for finding either minimum-measure simplices or degeneracies, since the tightest known lower bound is $\Omega(n \log n)$.

References

[1] A. Aggarwal, L. J. Guibas, J. Saxe, and P. W. Shor. A linear time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete Comput. Geom.*, 4:591–604, 1989.

[2] E. G. Anagnostu, L. J. Guibas, and V. G. Polimenis. Topological sweeping in three dimensions. In *Internat. Symp. SIGAL '90*, volume 450 of *Lecture Notes in Computer Science*, pages 310–317. Springer-Verlag, 1990.

[3] F. Aurenhammer. The one-dimensional weighted Voronoi diagram. *Inform. Process. Lett.*, 22:119–123, 1986.

[4] F. Aurenhammer. Power diagrams: Properties, algorithms, and applications. *SIAM J. Comput.*, 16:78–96, 1987.

[5] B. Chazelle, L. J. Guibas, and D. T. Lee. The power of geometric duality. *BIT*, 25:76–90, 1985.

[6] L. P. Chew and S. Fortune. Sorting helps for Voronoi diagrams. In *13th Symp. Mathematical Progr.*, Japan, 1988.

[7] H. Djidjev and A. Lingas. On computing the Voronoi diagram for restricted planar figures. In *2nd Worksh. Algorithms and Data Structures*, volume 519 of *Lecture Notes in Computer Science*, pages 54–64. Springer-Verlag, 1991.

[8] D. P. Dobkin and J. I. Munro. Efficient uses of the past. *J. Algorithms*, 6:455–465, 1985.

[9] H. Edelsbrunner. Geometric structures in computational complexity. In *15th Internat. Colloq. Automat. Lang. Program.*, volume 317 of *Lecture Notes in Computer Science*, pages 201–213. Springer-Verlag, 1988.

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

[11] H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM J. Comput.*, 15:341–363, 1986.

[12] H. Edelsbrunner, R. Seidel, and M. Sharir. On the zone theorem for hyperplane arrangments. In H. Maurer, editor, *New Results and New Trends in Computer Science*, volume 555 of *Lecture Notes in Computer Science*, pages 108–123. Springer-Verlag, 1991.

[13] H. Edelsbrunner and E. Welzl. Constructing belts in two-dimensional arrangements with applications. *SIAM J. Comput.*, 15:271–284, 1986.

[14] D. Eppstein. Finding the k smallest spanning trees. In *2nd Scand. Worksh. Algorithm Theory*, volume 447 of *Lecture Notes in Computer Science*, pages 38–47. Springer-Verlag, 1990.

[15] Z. Galil and R. Giancarlo. Speeding up dynamic programming with applications to molecular biology. *Theor. Comput. Sci.*, 64:107–118, 1989.

[16] M. M. Klawe and D. J. Kleitman. An almost linear time algorithm for generalized matrix searching. *SIAM J. Disc. Math.*, 3:81–97, 1990.

[17] L. L. Larmore. An optimal algorithm with unknown time complexity for convex matrix searching. *Inform. Process. Lett.*, 36:147–151, 1990.

[18] J. Nievergelt and F. P. Preparata. Plane-sweep algorithms for intersecting geometric figures. *Comm. ACM*, 25:739–747, 1982.