Self-improving Algorithms for Delaunay Triangulations

Kenneth L. Clarkson
IBM Almaden Research Center

C. Seshadhri
Dept. of Computer Science
Princeton University

July 22, 2008

Abstract

We study the problem of two-dimensional Delaunay triangulation in the self-improving algorithms model [1]. We assume that the n points of the input each come from an independent, unknown, and arbitrary distribution. The first phase of our algorithm builds data structures that store relevant information about the input distribution. The second phase uses these data structures to efficiently compute the Delaunay triangulation of the input. The running time of our algorithm matches the information-theoretic lower bound for the given input distribution, implying that if the input distribution has low entropy, then our algorithm beats the standard $\Omega(n \log n)$ bound for computing Delaunay triangulations.

Our algorithm and analysis use a variety of techniques: ε-nets for disks, entropy-optimal point-location data structures, linear-time splitting of Delaunay triangulations, and information-theoretic arguments.

1 Introduction

Data in the real world often has some structure. Suppose the inputs to an algorithm are generated by a probability distribution. Even if the distribution cannot be represented by a closed form expression, it may have some structural properties which can be exploited to speed up the algorithm. A standard algorithm will not be able to exploit this extra structure.

The model of *self-improving algorithms* was defined by Ailon *et al.* [1] to capture these scenarios. Suppose we wish to compute a function f on a sequence of inputs $I_1, I_2, \cdots$ which are being generated from an unknown and arbitrary time-invariant distribution $\mathcal{D}$. A self-improving algorithm for computing f initially only gives standard worst-case guarantees. As it handles more and more inputs, it learns about the distribution. Eventually, it tunes itself to be more efficient for $\mathcal{D}$, and may beat the worst-case running time. Self-improving algorithms have two phases: the initial *learning phase*, where the algorithm learns about $\mathcal{D}$ and builds data structures storing this information, and the *limiting phase*, where the algorithm uses the information obtained to speed up the running time. The main parameters of a self-improving algorithm are the number of rounds (number of problem instances) in the learning phase, the space used to store information about the distribution, and the running time in the limiting phase.

The basic intuition is that if $\mathcal{D}$ has low entropy, then the self-improving algorithm should be able to make a significant improvement. Of course, it may take too long to learn $\mathcal{D}$ as a whole, or even to learn a reasonable approximation of it. The challenge is to learn as little as possible about $\mathcal{D}$ and still glean enough to improve the running time. If the entropy of the input distribution for a sorting problem is low, for example, then a self-improving algorithm for sorting the resulting input will do better than the standard $\Omega(n \log n)$ lower bound.

We take the concept of self-improving algorithms to the geometric realm, and the problem of computing Delaunay triangulations. The most relevant result is that of [1], where a self-improving sorter was constructed. We borrow some ideas from there, and use geometric techniques to design a self-improving algorithm for Delaunay triangulations. In this new mode of algorithmic analysis, our algorithm is optimal, as its running time in the limiting phase matches the information-theoretic lower bound for computing the output over inputs from a fixed distribution.

We now formally define the problem. Let

$$I := (x_1, \cdots, x_n)$$

denote an input instance, where each x_i is a point in the plane, generated by a point distribution $\mathcal{D}_i$. The distributions $\mathcal{D}_i$ are arbitrary, and may be continuous, although we never explicitly use such a condition. Each x_i is independent of the others, and so the input I is drawn from the product distribution $\mathcal{D} := \prod_i \mathcal{D}_i$. In each round, a new input I is drawn from $\mathcal{D}$, and we wish to compute the Delaunay triangulation $T(I)$ of I. We use the comparison model, so any operation consists of evaluating a polynomial at some point (more details about this are given in Section 3). Although it is not critical, for the sake of simplicity, we will assume that the points of I are in general position, which is true with probability one when all the $\mathcal{D}_i$'s are continuous.

The distribution $\mathcal{D}$ also induces a (discrete) distribution on the set of Delaunay triangulations, viewed as labeled graphs on the vertex set $[1, n]$. Consider the entropy of this distribution: for each graph G on $[1, n]$, let p_G be the probability that it represents the Delaunay triangulation of $I \in_R \mathcal{D}$. Let the output entropy $H(T(I)) := -\sum_G p_G \log p_G$. By information-theoretic arguments, this quantity is a lower bound on the expected time required by any comparison-based algorithm to compute the Delaunay triangulation of $I \in_R \mathcal{D}$. An *optimal* algorithm will be one that has an expected running time of $O(H(T(I)) + n)$.

Our main result is the following.

Theorem 1.1 *For inputs $I_1, I_2, \ldots$ drawn from the product distribution $\mathcal{D} = \prod_i \mathcal{D}_i$, and for any constant $\varepsilon > 0$, there is a self-improving algorithm for finding the Delaunay triangulations of the I_j that has a learning phase of $O(n^\varepsilon)$ rounds and uses $O(n^{1+\varepsilon})$ space* [1]*. The limiting-phase running time is $O(\varepsilon^{-1}(H(T(I)) + n))$, and therefore optimal.*

Why is the independence of the $\mathcal{D}_i$'s important? A lower bound from [1] shows that any optimal self-improving sorter that handles *all* possible distributions requires exponential space. From the reduction of sorting to Delaunay triangulations, the following is an immediate corollary of Lemma 2.1 in [1].

Corollary 1.2 *There is an input distribution $\mathcal{D}$ such that any self-improving algorithm computing the Delaunay triangulation of inputs from $\mathcal{D}$ in $O(H(D) + n)$ limiting running time requires $\Omega(2^n)$ space.*

Furthermore, by Lemma 2.5 of [1], the time-space tradeoff we provide is essentially optimal.

2 The algorithm

We describe the algorithm in two parts. The first part explains the learning phase and the data structures that are constructed. The second part explains how these data structures are used to speed up the computation in the limiting phase. The expected running time will be expressed in terms of certain parameters of the data structures obtained in the learning phase. In the next section, we will prove that these parameters are comparable to the output entropy $H(\mathcal{D})$. We will assume in this section that the distributions $\mathcal{D}_i$ are known to us. Furthermore, the data structures described here will use $O(n^2)$ space. Section 4 shows how to remove this assumption and give the space-time tradeoff bounds of Theorem 1.1.

2.1 Learning Phase

For each round in the learning phase, we use a standard algorithm to compute the output Delaunay triangulation. We also perform some extra computation to build some data structures that will allow speedup in the limiting phase. These data structures are easily described.

The learning phase is as follows. Take the first $k := c \log n$ input lists $I_1, I_2, \cdots, I_k$, where c is a sufficiently large constant. Merge them into one list S of $kn = cn \log n$ points. Setting $\varepsilon := 1/n$, find an ε-net V for the set of all open disks. In other words, find a set $V \subseteq S$ such that for any open disk C that contains more than $\varepsilon kn = c \log n$ points of S, C contains at least one point of V. Matousek, *et al.* show that [7] there exist ε-nets of size $O(1/\varepsilon)$ for disks, where here $O(1/\varepsilon) = O(n)$. Furthermore, a construction

[1]The total time required for the learning phase is also $O(n^{1+\varepsilon})$.

and analysis similar to that of Clarkson and Varadarajan [6] yields a randomized construction that takes $n(\log n)^{O(1)}$ expected time.

We construct the Delaunay triangulation of V, which we denote by $T(V)$. We build an optimal planar point location structure (called Γ) for $T(V)$: given a point, we can find the triangle of $T(V)$ that it lies in $O(\log n)$ time. Define the random variable t_i to be the triangle of $T(V)$ that x_i falls into. Now let the entropy of t_i be H_i^V. If the probability that x_i falls in triangle t of $T(V)$ is p_i^t, then $H_i^V = -\sum_t p_i^t \log p_i^t$. For each i, we construct a search structure Γ_i of size $O(n)$ that finds t_i in expected $O(H_i^V)$ time. These Γ_i's can be constructed using the results of Arya *et al.* [3], for which the number of primitive comparisons is $H_i^V + o(H_i^V))$.

We will show that the triangles of $T(V)$ do not contain many points of a new input $I \in_R \mathcal{D}$ on the average. Consider a triangle t of $T(V)$ and let C_t be its circumscribed Delaunay disk. Let $X_t := |I \cap C_t|$, the random variable that is the number of points of $I \in_R \mathcal{D}$ that fall inside C_t. Note that the randomness comes from the random distribution of S, and so V and $T(V)$, as well as the randomness of I. We are interested in the expectation $\mathbf{E}_I[X_t]$ over I of X_t.

Claim 2.1 *With probability at least $1 - 1/n^3$ over the construction of $T(V)$, for every triangle t of $T(V)$, $\mathbf{E}_I[X_t] = O(1)$.*

Proof: Let the list of points S be $s_1, \cdots, s_{kn}$, the concatenation of I_1 through I_k. Consider the triangle t with vertices s_1, s_2, s_3. Note that all the remaining $kn - 3$ points are chosen independently of these three, from some distribution $\mathcal{D}_\ell$. For each $j \in [4, kn]$, let $Y_j^{(t)}$ be the indicator variable for the event that s_j is inside C_t. Let $Y^{(t)} = \sum_j Y_j^{(t)}$. By the Chernoff bound, for any $\beta \in (0, 1]$,

$$\Pr[Y^{(t)} \leq (1-\beta)\mathbf{E}[Y^{(t)}]] \leq e^{-\beta^2 \mathbf{E}[Y^{(t)}]/2}$$

Setting $\beta = 1/2$, if $\mathbf{E}[Y^{(t)}] > 48 \log n$, then $Y^{(t)} > 24 \log n$ with probability at least $1 - n^{-6}$. We can now consider any triangle generated by some triple of points s_i, s_j, s_m, for $i, j, m \in [4, kn]$, and apply the same argument as above. Taking a union bound over all triples of the points in S, we obtain that with probability at least $1 - n^{-3}$, for any triangle t generated by the points of S, if $\mathbf{E}[Y^{(t)}] > 48 \log n$, then $Y^{(t)} > 24 \log n$. We henceforth assume that this event happens.

Consider a triangle t of $T(V)$ and its circumcircle C_t. Since $T(V)$ is Delaunay, C_t contains no point of V in its interior. Since V is a $(1/n)$-net for all disks with respect to S, C_t contains at most $c \log n$ points of S, that is, $Y^{(t)} \leq c \log n$. This implies that $\mathbf{E}[Y^{(t)}] = O(\log n)$, as in the previous paragraph. Since $\mathbf{E}[Y^{(t)}] > (\log n - 3)\mathbf{E}_I[X_t]$, we obtain $\mathbf{E}_I[X_t] = O(1)$, as claimed. □

2.2 Limiting Phase

Suppose that we are done with the learning phase, and have $T(V)$ with the property given in Claim 2.1: for every triangle $t \in T(V)$, $\mathbf{E}_I[X_t] = O(1)$. We have reached the limiting phase where the algorithm is expected to compute the Delaunay triangulation with the optimal running time. We will prove the following lemma in this section.

Lemma 2.2 *Using the data structures from the learning phase, and the properties of them that hold with probability $1 - O(1/n)$, in the limiting phase the Delaunay triangulation of input I can be generated in expected $O(n + \sum_{i=1}^n H_i^V)$ time.*

The algorithm, and the proof of this lemma, has two steps. In the first step, $T(V)$ is used to quickly compute $T(V \cup I)$, with the time bounds of the lemma. In the second step, $T(I)$ is computed from $T(V \cup I)$, using a randomized splitting algorithm proposed by Chazelle *et al* [5], whose Theorem 3 is as follows.

Theorem 2.3 *Given a set of n points P and its Delaunay triangulation, for any partition of P into two disjoint subsets P_1 and P_2, the Delaunay triangulations $T(P_1)$ and $T(P_2)$ can be computed in $O(n)$ expected time, using a randomized algorithm.*

The remainder of the proof of the lemma, and of this subsection, is devoted to showing that $T(V \cup I)$ can be computed in the time bound of the lemma. The algorithm is as follows. For each x_i, we use Γ_i to find the triangle t_i of $T(V)$ that contains it. By the arguments given in the previous section, this takes time $O(\sum_{i=1}^{n} H_i^V)$. We now need to argue that given the t_i's, the Delaunay triangulation $T(V \cup I)$ can be computed in expected linear time. For each x_i, we walk through $T(V)$ and find all the Delaunay disks of $T(V)$ that contain x_i, as in incremental constructions of Delaunay triangulations. This is done by breadth-first search of the dual graph of $T(V)$, starting from t_i. Let S_i denote the set of circumcircles containing x_i. The following claim implies that this procedure will work.

Claim 2.4 *The set of $t \in T(V)$ with $C_t \in S_i$ is a connected set in the dual graph of $T(V)$.*

Proof: Omitted.

Claim 2.5 *Given all t_i's, all S_i sets can be found in expected $O(n)$ time.*

Proof: To find all circles containing x_i, do a breadth-first search from t_i. For any triangle t encountered, check if C_t contains x_i. If it does not, then we do not look at the neighbors of t. By Claim 2.4, we will visit all C_t's that contain x_i. The time taken to find S_i is $O(|S_i|)$. The total time taken to find all S_t's (once all the t_i's are found) is $O(\sum_{i=1}^{n} |S_i|)$. Define the indicator function $\chi(t, x_i)$ that takes value 1 if $x_i \in t$ and zero otherwise. We have

$$\sum_{i=1}^{n} |S_i| = \sum_{i=1}^{n} \sum_{t \in T(V)} \chi(t, x_i) = \sum_{t \in T(V)} \sum_{i=1}^{n} \chi(t, x_i) = \sum_{t} X_t.$$

Therefore, by Claim 2.1,

$$\mathbf{E}[\sum_{i=1}^{n} |S_i|] = \mathbf{E}[\sum_{t} X_t] = \sum_{t} \mathbf{E}[X_t] = O(n).$$

This implies that all S_i's can be found in expected linear time. □

Our aim is to build the Delaunay triangulation of $V \cup I$ in linear time using the S_i sets. This is done by a standard incremental construction where the x_i's are added in order $x_1, x_2, \cdots, x_n$. We will show how we can get the set of edges that each x_i will "kill" using the S_i sets. We will assume that given any triangle t, we can get all the S_i sets that t belongs to.

Let $V_i := V \cup \{x_1, \cdots, x_i\}$. When we add x_1, the edges of $T(V)$ that will be affected are the edges of triangles in S_1. Therefore, $T(V_1)$ can be obtained in $O(|S_1|)$ time. Now suppose we have $T(V_{i-1})$ and we add x_i. Again, we can show that if some edge from $T(V)$ is affected, it must be an edge of S_i.

Claim 2.6 *When x_i is added to $T(V_{i-1})$, suppose that edge e is removed. If the endpoints of e are both in V, then e is an edge of some triangle in S_i.*

This is proved in the appendix.

The claim above shows that only $O(|S_i|)$ time is required to find edges from $T(V)$ that are removed. But now, we have the additional problem of finding affected edges which may not have an endpoint in V, and therefore are not present in $T(V)$.

Claim 2.7 *Suppose e has an endpoint in I. There is an edge $f \in T(V)$ such that, for the two triangles t, t' incident on f, the point x_i and the endpoints of e lie in either C_t or $C_{t'}$.*

The proof is given in the appendix.

This now gives us a method of finding edges of $T(V_{i-1})$ affected by the addition of x_i. Take a triangle $t \in S_i$ and choose an edge e of t (for ease of notation, we will say $e \in t$). Let the neighbor of t incident to e be t'. Look at the points in $\{x_1, \cdots, x_{i-1}\}$ that are in C_t and $C_{t'}$, and take the edges of $T(V_{i-1})$ between them. These are the edges that need to be checked.

Claim 2.8 *Given all S_i sets and $T(V)$, $T(V_n)$ can be generated in expected linear time.*

Proof: The total time taken to handle all edges of $T(V)$ that get killed is $\mathbf{E}[\sum_{i=1}^{n} |S_i|] = O(n)$. Consider some $t \in T(V)$ and edge e of t. Let $t^e \in T(V)$ be incident to e. The random variable $Z_{t,e}$ is set to be $X_t X_{t^e}$. By Claim 2.7, the total time to find all (other) affected edges is bounded above by

$$\sum_{i=1}^{n} \sum_{t \in S_i} \sum_{e \in t} Z_{t,e}.$$

For a triangle t, we define the indicator random variable $\chi(t, i)$, as before, for the event that x_i falls in C_t. Thus, $X_t = \sum_{i=1}^{n} \chi(t, i)$.

$$\begin{aligned} \sum_{i=1}^{n} \sum_{t \in S_i} \sum_{e \in t} Z_{t,e} &= \sum_{i=1}^{n} \sum_{t} \chi(t, i) \sum_{e \in t} X_t X_{t^e} \\ &= \sum_{i=1}^{n} \sum_{t} \sum_{e \in t} \chi(t, i) X_t X_{t^e}, \end{aligned}$$

and

$$\begin{aligned} \mathbf{E}[\chi(t, i) X_t X_{t^e}] &= \mathbf{E}\big[\chi(t, i) \sum_{j=1}^{n} \chi(t, j) \sum_{k=1}^{n} \chi(t^e, k)\big] \\ &= \sum_{j=1}^{n} \mathbf{E}[\chi(t, i)\chi(t, j)\chi(t^e, j)] \\ &\quad + \sum_{j \neq k} \mathbf{E}[\chi(t, i)\chi(t, j)\chi(t^e, k)] \end{aligned}$$

Since $\chi(t,i)$ is an indicator, $\chi(t,j)\chi(t^e,j) \le \chi(t,j)$. For $j \neq k$, $\chi(t,j)$ and $\chi(t^e,k)$ are independent. For the second summation in the equation above, we can separate out the case $i = j$ and $i = k$.

$$
\begin{aligned}
&\mathbf{E}[\chi(t,i)X_tX_{t^e}] \\
&= \sum_{j=1}^{n} \mathbf{E}[\chi(t,i)\chi(t,j)\chi(t^e,j)] + \sum_{k\neq i} \mathbf{E}[\chi(t,i)^2\chi(t^e,k)] \\
&\quad + \sum_{j\neq i} \mathbf{E}[\chi(t,i)\chi(t^e,i)\chi(t,j)] \\
&\quad + \sum_{i\neq j\neq k} \mathbf{E}[\chi(t,i)\chi(t,j)\chi(t^e,k)] \\
&\le \sum_{j=1}^{n} \mathbf{E}[\chi(t,i)\chi(t,j)] + \mathbf{E}[\chi(t,i)] \sum_{k\neq i} \mathbf{E}[\chi(t^e,k)] \\
&\quad + \sum_{j\neq i} \mathbf{E}[\chi(t,i)\chi(t,j)] \\
&\quad + \mathbf{E}[\chi(t,i)] \sum_{i\neq j\neq k} \mathbf{E}[\chi(t,j)]\mathbf{E}[\chi(t^e,k)] \\
&= \mathbf{E}[\chi(t,i)] + \mathbf{E}[\chi(t,i)] \sum_{j\neq i} \mathbf{E}[\chi(t,j)] \\
&\quad + \mathbf{E}[\chi(t,i)] \sum_{k\neq i} \mathbf{E}[\chi(t^e,k)] \\
&\quad + \mathbf{E}[\chi(t,i)] \sum_{j\neq i} \mathbf{E}[\chi(t,j)] \\
&\quad + \mathbf{E}[\chi(t,i)] \sum_{i\neq j\neq k} \mathbf{E}[\chi(t,j)]\mathbf{E}[\chi(t^e,k)] \\
&\le \mathbf{E}[\chi(t,i)] + 2\mathbf{E}[\chi(t,i)] \sum_{j=1}^{n} \mathbf{E}[\chi(t,j)] \\
&\quad + \mathbf{E}[\chi(t,i)] \sum_{k=1}^{n} \mathbf{E}[\chi(t^e,k)] \\
&\quad + \mathbf{E}[\chi(t,i)] \big(\sum_{j=1}^{n} \mathbf{E}[\chi(t,j)]\big)\big(\sum_{k=1}^{n} \mathbf{E}[\chi(t^e,k)]\big) \\
&= \mathbf{E}[\chi(t,i)]\big(1 + 2\mathbf{E}[X_t] + \mathbf{E}[X_{t^e}] + \mathbf{E}[X_t]\mathbf{E}[X_{t^e}]\big)
\end{aligned}
$$

By Claim 2.1, we get that $\mathbf{E}[\chi(t,i)X_tX_{t^e}] \le \alpha\mathbf{E}[\chi(t,i)]$, for some fixed constant α. The expected running time is bounded by

$$
\begin{aligned}
\mathbf{E}[\sum_{i=1}^{n}\sum_{t\in S_i}\sum_{e\in t} Z_{t,e}] &= \sum_{i=1}^{n}\sum_{t}\sum_{e\in t} \mathbf{E}[\chi(t,i)X_tX_{t^e}] \\
&\le \alpha \sum_{i=1}^{n}\sum_{t}\sum_{e\in t} \mathbf{E}[\chi(t,i)] \\
&= 3\alpha \sum_{i=1}^{n} \mathbf{E}\big[\sum_{t} \chi(t,i)\big] \\
&= 3\alpha \sum_{i=1}^{n} \mathbf{E}[|S_i|] = O(n)
\end{aligned}
$$

□

With this claim, it follows that $T(V_n)$ can be computed in expected $O(n + \sum_{i=1}^n H_i^V)$ time, and hence, as discussed at the beginning of this subsection, Lemma 2.2 follows.

3 Limiting Phase Optimality

In this section, we prove that the running time bound in Lemma 2.2 is indeed optimal. Before we get into the analysis of the various entropies that represent the running time, it is important to clarify the model of computation. We are using comparison based algorithms, where a single step (or "comparison") involves evaluating a point $(z_1, z_2, \cdots, z_d) \in \mathbb{R}^d$ (for constant d) at some polynomial $f(z_1, z_2, \cdots, z_d) : \mathbb{R}^d \to \mathbb{R}$ and checking if the result is positive or negative. Based on this result, the algorithm chooses the next comparison to make. An algorithm can be completely represented by a decision tree, with each node representing some comparison. In this model, we get an information-theoretic lower bound of $H(T(I))$ for computing the Delaunay triangulation of input $I \in_R \mathcal{D}$.

Recall that by Lemma 2.2, the running time of the our algorithm is expected $O(n + \sum_i H_i^V)$. The aim of this section is to prove the optimality of the algorithm by the following theorem.

Theorem 3.1 *For H_i^V the entropy of the triangle t_i of $T(V)$ containing x_i, and $H(T(I))$ the entropy of the Delaunay triangulation of I, considered as labeled graph,*

$$\sum_i H_i^V = O(H(T(I)) + n).$$

This theorem will be proven through a chain of lemmas, which will eventually connect $\sum_{i=1}^n H_i^V$ to $H(T(I))$. Note that V is a fixed set and there is no randomness in $T(V)$. As a result, for the sake of information theory bounds, we can assume that $T(V)$ is known in advance: indeed, any computation whatsoever can be done in advance on the points in V and is not charged as a comparison.

The chain of lemmas begins with $H(T(V_n))$, which is bounded above by $O(H(T(I) + n)$ in the next lemma. The entropy $H(T(V_n))$ is used to bound $\sum_i \mathbf{E}[H_i]$ in the following lemma, where H_i is the entropy of w_i, the triangle of $T(V_{i-1})$ that contains x_i. After some preliminary lemmas, the final lemma in the chain uses $\sum_i \mathbf{E}[H_i]$ to bound $\sum_i H_i^V$, as needed for the theorem.

By analogy to $H(T(I))$, let $H(T(V_n))$ be the entropy of $T(V_n)$ as a labeled graph, under the distribution induced by that of I. (Recall that $V_n := V \cup I$.) The entropy $H(T(V_n))$ is a lower bound for the expected running time of any comparison-based algorithm that computes $T(V_n)$.

The first lemma in the chain is the following.

Lemma 3.2

$$H(T(V_n)) = O(H(T(I)) + n).$$

Proof: Using Chazelle's linear-time algorithm to compute the intersection of two three-dimensional convex polyhedra [4], we can compute $T(V_n)$ in $O(n)$ time, given $T(V)$ and $T(I)$. Suppose we represent every graph induced by a Delaunay triangulation on n points by some string, denoted by $s(T)$. By information theory, there exists some string encoding such that $\mathbf{E}[|s(T(I))|] = O(H(T(I)))$. Suppose, for input I, we are given the string $s(T(I))$, so we can uniquely identify $T(I)$. Now, we use the linear-time algorithm to compute $T(V_n)$. Obviously, this algorithm only performs $O(n)$ comparisons. Therefore, the output $T(V \cup I)$ can be uniquely identified by $s(T(I))$ and cn more bits, for some constant c. By definition, $\mathbf{E}[|s(T(I))| + cn] \geq H(T(V_n))$. This completes the proof. □

Let us consider an incremental construction of $T(V_n)$. At the ith step, x_i is added to $T(V_{i-1})$. We can consider a random process associated with this step. The points $x_1, \cdots, x_{i-1}$ are already fixed, thereby fixing $T(V_{i-1})$. We can consider the entropy of the random variable w_i that is the triangle of $T(V_{i-1})$ in which x_i falls. More precisely, we define

$$H_i^{T(V_{i-1})} := H(w_i) = - \sum_{t \in T(V_{i-1})} p(i,t) \log p(i,t)$$

$p(i,t)$ is the probability that x_i lies in t. Note that this entropy itself is a random variable, since $T(V_{i-1})$ depends on $x_1, \cdots, x_{i-1}$ which are randomly chosen. But w_i is independent of this randomness (since the distributions $\mathcal{D}_i$ are all independent). Therefore, we can take the expectation over the random choices $\{x_1, \cdots, x_{i-1}\}$, $\mathbf{E}_{x_1,\cdots,x_{i-1}}[H_i^{T(V_{i-1})}]$. Again, let us explain what this means. Given any set of points $x_1, \cdots, x_{i-1}$, we can define the entropy $H_i^{T(V_{i-1})}$. Now, because the randomness of w_i only depends on the randomness of x_i, w_i is independent of $x_1, \cdots, x_{i-1}$. Obviously, $H_i^{T(V_{i-1})}$ is a function of $x_1, \cdots, x_{i-1}$. We take the expectation over the random choices of $x_1, \cdots, x_{i-1}$ to get $\mathbf{E}[H_i^{T(V_{i-1})}]$. For clarity, we drop the subscripts and denote this by $\mathbf{E}[H_i]$.

In the next lemma, we relate the entropy of this incremental procedure of constructing $T(V_n)$ to the actual entropy of the $T(V_n)$.

Lemma 3.3

$$\sum_i \mathbf{E}[H_i] = O(H(T(V_n)) + n)$$

To prove the lemma, we need a claim and a lemma. The claim follows from the proof of Claim 2.8, and the lemma is proven in the appendix.

Claim 3.4 *For all $j \leq i$, the expected degree of x_j in $T(V_i)$ is $O(\mathbf{E}[|S_j|])$.*

Lemma 3.5

$$H(w_1, \cdots, w_n) \geq \sum_{i=1}^{n} \mathbf{E}[H_i]$$

Here $H(w_1, \cdots, w_n)$ is the joint entropy of all $w_1, \cdots, w_n$, and a lower bound on the expected length of any string representation of $w_1, \cdots, w_n$.

Proof: (of Lemma 3.3) Before giving the details of the proof, let us first sketch out the main idea. Suppose all the random choices $x_1, \cdots, x_n$ have been made. We would like to argue that if we know $T(V_n)$, then in linear time we can determine the w_i's for all i. This will be done by a procedure that goes backwards: it first removes x_n, and then computes the Delaunay triangulation $T(V_{n-1})$. This can be done in time linear in the degree of x_n [2]. The triangle w_n can be determined in time linear in the degree of x_n. Now, we remove x_{n-1} and so on, thereby finding all w_i's. It seems that by a standard backwards analysis argument, we should remove the x_i's in random order. By a planarity argument, we should get that the expected degree (over the random order) is constant at every step. But because we remove only the points in I, which is a strict *subset* of V_n, this argument will not hold.

However, using the properties of V and the randomness of I, we can still argue that these degrees will be expected constant. From Claim 3.4, it is easy to see that we can get the w_i's in $O(\sum_i \mathbf{E}[|S_i|])$ time. Let us now apply an argument similar to that in Lemma 3.2. Let there be a string representation $s(T(V_n))$ for each possible $T(V_n)$. By definition of entropy, we can assume that $\mathbf{E}[|s(T(V_n))|] = O(H(T(V_n)))$. Using the procedure described above, we can uniquely identify $w_1, \cdots, w_n$ by a string of expected length $\mathbf{E}[|s(T(V_n))|] + O(\mathbf{E}[\sum_i |S_i|]) = O(H(T(V_n)) + n)$.

The proof now follows by Lemma 3.5, since $H(w_1, \cdots, w_n)$ is no more than $\mathbf{E}_I[|s(T(V_n))|$. □

We now come to the final lemma in our chain of entropy inequalities.

Lemma 3.6

$$\sum_i H_i^V = O(\sum_i \mathbf{E}[H_i] + n)$$

Proof: Consider $x_1, \cdots, x_{i-1}$ to be chosen, fixing the triangulation $T(V_{i-1})$. The entropy H_i is now well defined. As before, $w_i \in T(V_{i-1})$ and $t_i \in T(V)$ are the triangles that x_i falls into. We will describe a procedure that given w_i finds t_i using $O(|S_i|)$ comparisons. First, we look at the Delaunay triangulations as 3-dimensional polytopes. By projecting onto the paraboloid $z = x^2 + y^2$, each point of the Delaunay triangulation is represented by a halfspace in 3-dimensions. Every vertex of the polytope corresponds to a Delaunay triangle (or disk). Abusing notation, $T(V)$ and $T(V_{i-1})$ are going to be the respective polytopes. We start by tetrahedralizing $T(V)$. Since $T(V_{i-1})$ is completely contained in $T(V)$, for every vertex of $T(V_{i-1})$, we can determine a tetrahedron of $T(V)$ that contains it (maybe on the boundary). Note that all of this can be done *before* we look at point x_k. Given w_i, we can determine the tetrahedron that it lies in without any comparisons. Since the triangle w_i will certainly be destroyed on the addition of x_i, the vertex corresponding to w_i (in the polytope) will be removed by the addition of the plane x_i. Obviously, there is some vertex of the tetrahedron would also be removed by the addition of x_i to $T(V)$. In a constant number of queries, this vertex can be determined. Now, let us go back to the Delaunay triangulations. This vertex corresponds to some Delaunay disk of $T(V)$ killed by x_i. By doing a walk through $T(V)$, we can find t_i in $O(|S_i|)$ time. This implies that

$$H_i^V \leq H_i + O(|S_i| + 1).$$

Taking expectations over I and summing,

$$\sum_i H_i^V \leq \sum_i \mathbf{E}[H_i] + O(\mathbf{E}[\sum_i |S_i|] + n) \leq \sum_i \mathbf{E}[H_i] + O(n).$$

□

As discussed above, Theorem 3.1 now follows by combining Lemmas 3.2, 3.3, and 3.6.

4 The time-space tradeoff

We show how to remove the assumption that we have prior knowledge of the $\mathcal{D}_i$'s (to build the search trees Γ_i) and prove the time-space tradeoff given in Theorem 1.1. These techniques are identical to those used in [1] for their self-improving sorter. Let $\varepsilon > 0$ be any constant. The first $O(\log n)$ rounds of the learning phase are used as before to construct the Delaunay triangulation $T(V)$. To construct the tree Γ_i, we would need to know the exact probability with which x_i falls in every triangle of $T(V)$. Since these probabilities cannot be determined in a sublinear number of learning rounds, we build some suitable approximations for these search structures. We first build a standard search structure Γ over the triangles of $T(V)$. Given a point x, we can find the triangle of $T(V)$ that contains x in $O(\log n)$ time.

The learning phase goes on for $O(n^\varepsilon \log n)$ rounds. The main trick is to observe that (up to constant factors), the only probabilities that are relevant are those that are $> n^{-\varepsilon}$. In each round, for each x_i, we record the triangle of $T(V)$ that x_i falls into. At the end of $O(n^\varepsilon \log n)$ rounds, we take the set R_i of triangles such that for $t \in R_i$, x_i was in t for at least $\Omega(\log n)$ rounds. We remind the reader that p_i^t is the probability that x_i lies in triangle t. For every triangle in R_i, we have an estimate of the probability $\hat{p}_i^t$ (obtained by simply taking the total number of times that x_i lay in t, divided by the total number of rounds). By a standard Chernoff bound argument, for all $t \in R_i$, $\hat{p}_i^t = \Theta(p_i^t)$. Furthermore, for any triangle t, if $p_i^t = \Omega(n^{-\varepsilon})$, then $t \in R_i$.

For each x_i, we build the approximate search structure Γ_i. Consider the following probability distribution $\bar{p}_i$ over the triangles of $T(V)$: if $t \in R_i$, set $\bar{p}_i^t := \hat{p}_i^t / N_i$, where $N_i := \sum_{t \in R_i} \hat{p}_i^t$, and otherwise $\bar{p}_i^t := 0$. Using the construction of [3], we can build the optimal planar point location structure Γ_i according to the distribution $\bar{p}_i$. The limiting phase uses these structures to find t_i for every x_i: given x_i, we use Γ_i to search for it. If the search does not terminate in $\log n$ steps or Γ_i fails to find t_i (since $t_i \notin R_i$), then we use the standard search structure, Γ, to find t_i. Therefore, we are guaranteed to find t_i in $O(\log n)$ time. Without loss of generality, we can assume that each Γ_i deals with only n^ε triangles (and therefore, a planar subdivision of size n^ε). By the bounds given in [3], each Γ_i can be constructed with size n^ε in $n^\varepsilon \log n$ time. The total space is bounded by $n^{1+\varepsilon}$ and the time required to build them is at most $n^{1+\varepsilon} \log n$.

Let s_i^t denote the time to search for x_i given that $x_i \in t$. By the properties of Γ_i, and noting that

$N_i \le 1$,

$$\begin{aligned}
\sum_{t\in R_i} \bar{p}_i^t s_i^t &= \sum_{t\in R_i} \bar{p}_i^t \log(1/\bar{p}_i^t) \\
&= N_i^{-1} \sum_{t\in R_i} \hat{p}_i^t \log(N_i/\hat{p}_i^t) \\
&= N_i^{-1} \big[\sum_{t\in R_i} \hat{p}_i^t \log N_i - \sum_{t\in R_i} \hat{p}_i^t \log \hat{p}_i^t \big] \\
&\le -N_i^{-1} \sum_{t\in R_i} \hat{p}_i^t \log \hat{p}_i^t \\
&= O\big(N_i^{-1}(-\sum_{t\in R_i} p_i^t \log p_i^t + 1)\big)
\end{aligned}$$

We now bound the expected search time for x_i.

$$\begin{aligned}
\sum_t p_i^t s_i^t &= \sum_{t\in R_i} p_i^t s_i^t + \sum_{t\notin R_i} p_i^t s_i^t \\
&= O(\sum_{t\in R_i} \hat{p}_i^t s_i^t + \sum_{t\notin R_i} p_i^t \log n) \\
&= O\big(N_i \sum_{t\in R_i} \bar{p}_i^t s_i^t + \sum_{t\notin R_i} p_i^t \log n\big)
\end{aligned}$$

Noting that for $t \notin R_i$, $p_i^t = O(n^{-\varepsilon})$ and therefore $\log p_i^t \le -\varepsilon \log n + O(1)$, and so

$$\begin{aligned}
&\sum_t p_i^t s_i^t \\
&= O\big((-\sum_{t\in R_i} p_i^t \log p_i^t + 1) + \sum_{t\notin R_i} p_i^t \varepsilon^{-1}(-\log p_i^t + 1)\big) \\
&= O\big(\varepsilon^{-1}(-\sum_t p_i^t \log p_i^t + 1)\big) \\
&= O\big(\varepsilon^{-1}(H_i^V + 1)\big)
\end{aligned}$$

The total expected search time is $O(\varepsilon^{-1}(\sum_i H_i^V + n))$. By the analysis of Section 2 and Theorem 3.1, we have that the expected running time in the limiting phase is $O(\varepsilon^{-1}(H(\mathcal{D}) + n))$. This completes the proof of Theorem 1.1.

References

[1] Ailon, N., Chazelle, B., Comandur, S., Liu, D., *Self-Improving Algorithms*, Proc. 17th SODA (2006), 261–270.

[2] Aggarwal, A., Guibas, L., Saxe, J., Shor, P., *A Linear Time Algorithm for Computing the Voronoi Diagram of a Convex Polygon*, Discrete and Computational Geometry 4, 1989, 591–604.

[3] Arya, S., Malamatos, T., Mount, D. M., Wong, K.-C. *Optimal Expected-Case Planar Point Location*, SIAM J. Comput. 37, 2007, 584–610

[4] Chazelle, B., *An Optimal Algorithm for Intersecting Three-Dimensional Convex Polyhedra*, SIAM J. Computing 21, 1992, 671–696.

[5] Chazelle, B., Devillers, O.,Hurtado, F., Mora, M., Sacristan, V., Teillaud, M. *Splitting a Delaunay Triangulation in Linear Time*, Algorithmica 34, 2002, 39–46.

[6] Clarkson, K., Varadarajan, K. *Improved Approximation Algorithms for Geometric Set Cover* Discrete and Computation Geometry 37, 2007, 43–58.

[7] Matousek, J., Seidel, R., Welzl, E. *How to Net a Lot with Little: Small epsilon-Nets for Disks and Halfspaces*, Proc. 6th SOCG (1990), 16–22.

A Proofs for Section 2.2

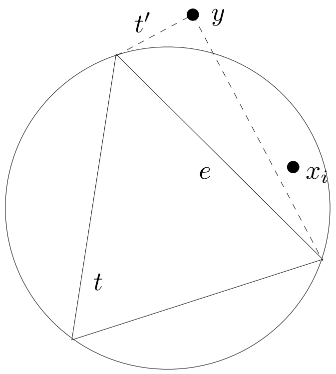

Figure 1:

Proof: (Claim 2.6) The edge e must be an edge in $T(V)$. Also, e is an edge of triangle t in $T(V_{i-1})$. Refer to Figure 1. The point x_i is in the sector bounded by C_t and e. Since $e \in T(V)$, there must be a point $y \in V$ such that e and y form a triangle t' of $T(V)$ and x_i and y are on the same side of e. The point y cannot be inside C_t, since t is a Delaunay triangle of $T(V_{i-1})$. Therefore, the angle subtended by y at e is smaller than that of x_i. The circle $C_{t'}$ must contain x_i and $t' \in S_i$. □

Proof: (Claim 2.7) Suppose some edge e in triangle $t \in T(V_{i-1})$ is killed by x_i. We will denote the vertices of t by u_1, u_2, and u_3, and for ease of notation, we will denote x_i by u_4. The point x_i is inside C_t. Consider the set of edges of $T(V)$ that intersect C_t. We can impose a natural linear ordering on these edges. If none of these edges separate the set points $U = \{u_1, \cdots, u_4\}$, then they all lie in a triangle t of $T(V)$.

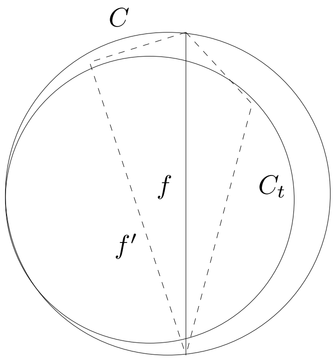

Figure 2:

Consider the first edge f that separates U into U_ℓ and U_r. Refer to Figure 2. Assume that the next edge after f to the right (in the ordering) that intersects C_t does not separate U. Let the two triangles of $T(V)$ that share f as an edge be t_ℓ, t_r (the left and right triangles). The triangle t_r contains U_r. Let C be the circle tangential to C at the left of f and having f as a chord. (Note that if f is actually an edge of t, then C_t and C are just the same, and we will end up with Claim 2.6.) If the vertex of t_ℓ not on f lies outside C, then C_{t_ℓ} will contain all of C_t which lies to the left of f. This implies that C_{t_ℓ} contains U_ℓ. The situation is as follows: triangles t_ℓ and t_r in $T(V)$ share edge f. Edge f divides U into U_ℓ and U_r and they are contained in C_{t_ℓ} and C_{t_r}, respectively, proving the claim (for this case).

Suppose the third vertex of t_ℓ is inside C. The triangle t_ℓ is shown by the dashed triangle in Figure 2 to the left of f. Let the angle to f be θ_ℓ. Let the angle subtended by any point in the right part of C_t be θ_r. Note that $\theta_\ell + \theta_r > \pi$. Therefore, the circle C_{t_ℓ} will contain the right part of C_t (and, as a result, U_r).

The edge f' is in $T(V)$ and intersects C_t. Also, f' is larger than f in the ordering of edges. If f' does not separate U, then C_{t_ℓ} must contain U_ℓ and we are done. If not, then suppose f' divides U into U'_ℓ and U'_r. The triangle t_ℓ is actually to the right of f' and C_{t_ℓ} contains U'_r. This leaves us in a situation analogous to f: the circumcircle of triangle to the right of f' contains all points in U to the right of f'. Therefore, we can apply the same argument as above: either we will stop, getting our desired triangles, or we will move to the edge to the right of f'. □

B Proof for Lemma 3.5

Proof: (Lemma 3.5) We will prove by induction on k that $H(w_1, \cdots, w_k) \geq \sum_{i=1}^{k} \mathbf{E}[H_i]$. Recall that w_k is the triangle of $T(V_{k-1})$ that contains x_k. The claim is a consequence of the independence of the x_i's. The reason why we cannot immediately use independence is that the random variable w_i *does* depend on the choices of $x_1, \cdots, x_{k-1}$, because w_k depends on $T(V_{k-1})$, which depends on $x_1, \cdots, x_{k-1}$. In some sense, we are just stating a well known fact about conditional entropies, but this small technical problem forces us to reprove it for our setting. We proceed with a proof by induction.

base case: For $k = 1$, $H(w_1) = H_1$ (note that H_1 is not a random variable).

induction step: Assume the claim is true up to $k-1$. For any triangle t (which is specified by a triple of vertex labels), let $p(i,t)$ be the probability that x_i falls in t. Suppose that $x_1, \cdots, x_{k-1}$ are fixed. We have

$$H_{k+1} = -\sum_{t \in T(V_{k-1})} p(k,t) \log p(k,t).$$

Let $\gamma(k-1,t)$ be the indicator variable of the event that $T(V_{k-1})$ has triangle t. This is a random variable, depending on $x_1, \cdots, x_{k-1}$. Removing the assumption that $x_1, \cdots, x_{k-1}$ are fixed, we take the expectation of H_k:

$$\begin{aligned} \mathbf{E}[H_k] &= -\mathbf{E}[\sum_t \gamma(k-1,t) p(k,t) \log p(k,t)] \\ &= -\sum_t \mathbf{E}[\gamma(k-1,t)] p(k,t) \log p(k,t) \end{aligned}$$

Consider some sequence of triangles $\Delta_k = \langle t_1, \cdots, t_k \rangle$. For $i \leq k$, let $\mathcal{E}_i(\Delta)$ denote the event that $w_1 = t_1, w_2 = t_2, \cdots, w_{i-1} = t_{i-1}$.

$$\begin{aligned} \Pr[\mathcal{E}_k(\Delta_k)] &= \Pr[\mathcal{E}_{k-1}(\Delta_k)] \times \Pr[w_k = t_k \big| \mathcal{E}_{k-1}(\Delta_k)] \\ &= \Pr[\mathcal{E}_{k-1}(\Delta_k)] \\ &\qquad \times p(k,t_k) \Pr[\gamma(k-1,t_k) = 1 \big| \mathcal{E}_{k-1}(\Delta_k)]. \end{aligned}$$

This is just a consequence of the independence of x_k from $x_1, \cdots, x_{k-1}$. As a result, the probability $p(k,t_k)$ is not affected by the values of $w_1, \cdots, w_{k-1}$. Note also that $\Pr[w_k = t_k \big| \mathcal{E}_{k-1}(\Delta_k)] = \Pr[w_k = t_k \big| \mathcal{E}_{k-1}(\Delta_{k-1})]$. Taking the convention that $0 \log 0 = 0$, we can now express as a sum the entropy $H(w_1, \cdots, w_k)$.

$$\begin{aligned} &H(w_1, \cdots, w_k) \\ &\quad = -\sum_{\Delta_k} \Pr[\mathcal{E}_k(\Delta_k)] \log \Pr[\mathcal{E}_k(\Delta_k)] \\ &\quad = -\sum_{\Delta_k} \Pr[\mathcal{E}_k(\Delta_k)] \log(\Pr[\mathcal{E}_{k-1}(\Delta_k)] \\ &\qquad\qquad \times p(k,t_k) \Pr[\gamma(k-1,t_k) = 1 \big| \mathcal{E}_{k-1}(\Delta)]) \\ &\quad = -\sum_{\Delta_k} \Pr[\mathcal{E}_k(\Delta_k)](\\ &\qquad\qquad \log(\Pr[\mathcal{E}_{k-1}(\Delta_k)]) \\ &\qquad\qquad + \log p(k,t_k) \\ &\qquad\qquad + \log(\Pr[\gamma(k-1,t_k) = 1 \big| \mathcal{E}_{k-1}(\Delta)])) \end{aligned}$$

We now open the parentheses and consider each sum separately.

$$
\begin{aligned}
&-\sum_{\Delta_k} \Pr[\mathcal{E}_k(\Delta_k)] \log(\Pr[\mathcal{E}_{k-1}(\Delta_k)]) \\
&= -\sum_{\Delta_k} \log(\Pr[\mathcal{E}_{k-1}(\Delta_k)])\Pr[\mathcal{E}_{k-1}(\Delta_k)] \\
&\qquad\qquad \times \Pr[w_k = t_k \big| \mathcal{E}_{k-1}(\Delta_{k-1})] \\
&= -\sum_{\Delta_{k-1}, t_k} \log(\Pr[\mathcal{E}_{k-1}(\Delta_{k-1})])\Pr[\mathcal{E}_{k-1}(\Delta_{k-1})] \\
&\qquad\qquad \times \Pr[w_k = t_k \big| \mathcal{E}_{k-1}(\Delta_{k-1})] \\
&= -\sum_{\Delta_{k-1}} \Pr[\mathcal{E}_{k-1}(\Delta_{k-1})] \log(\Pr[\mathcal{E}_{k-1}(\Delta_{k-1})]) \\
&\qquad\qquad \times \sum_{t_k} \Pr[w_k = t_k \big| \mathcal{E}_{k-1}(\Delta_{k-1})] \\
&= -\sum_{\Delta_{k-1}} \Pr[\mathcal{E}_{k-1}(\Delta_{k-1})] \log(\Pr[\mathcal{E}_{k-1}(\Delta_{k-1})]) \times 1 \\
&= H(w_1, \cdots, w_{k-1}).
\end{aligned}
$$

Now consider the next sum in a similar manner:

$$
\begin{aligned}
&-\sum_{\Delta_k} \log p(k,t_k) \Pr[\mathcal{E}_{k-1}(\Delta_k)] \\
&\qquad \times p(k,t_k) \Pr[\gamma(k-1,t_k)=1 \big| \mathcal{E}_{k-1}(\Delta_k)] \\
&= -\sum_{t_k} p(k,t_k) \log p(k,t_k) \sum_{\Delta_{k-1}} \Pr[\mathcal{E}_{k-1}(\Delta_{k-1})] \\
&\qquad \times \Pr[\gamma(k-1,t_k)=1 \big| \mathcal{E}_{k-1}(\Delta_{k-1})] \\
&= -\sum_{t_k} p(k,t_k) \log p(k,t_k) \sum_{\Delta_{k-1}} \Pr[\gamma(k-1,t_k)=1] \\
&= -\sum_{t_k} \mathbf{E}[\gamma(k-1,t_k)] p(k,t_k) \log p(k,t_k) = \mathbf{E}[H_k]
\end{aligned}
$$

The third sum is always positive. This implies that

$$H(w_1,\cdots,w_k) \geq H(w_1,\cdots,w_{k-1}) + \mathbf{E}[H_k] \geq \sum_{i=1}^{k} \mathbf{E}[H_i].$$

$\square$