Self-Improving Voronoi Construction for a Hidden Mixture of Product Distributions

Siu-Wing Cheng
Hong Kong University of Science and Technology, Hong Kong, China

Man Ting Wong
Hong Kong University of Science and Technology, Hong Kong, China

Abstract

We propose a self-improving algorithm for computing Voronoi diagrams under a given convex distance function with constant description complexity. The n input points are drawn from a hidden mixture of product distributions; we are only given an upper bound $m = o(\sqrt{n})$ on the number of distributions in the mixture, and the property that for each distribution, an input instance is drawn from it with a probability of $\Omega(1/n)$. For any $\varepsilon \in (0,1)$, after spending $O\big(mn\log^{O(1)}(mn) + m^{\varepsilon}n^{1+\varepsilon}\log(mn)\big)$ time in a training phase, our algorithm achieves an $O\big(\frac{1}{\varepsilon}n\log m + \frac{1}{\varepsilon}n2^{O(\log^* n)} + \frac{1}{\varepsilon}H\big)$ expected running time with probability at least $1 - O(1/n)$, where H is the entropy of the distribution of the Voronoi diagram output. The expectation is taken over the input distribution and the randomized decisions of the algorithm. For the Euclidean metric, the expected running time improves to $O\big(\frac{1}{\varepsilon}n\log m + \frac{1}{\varepsilon}H\big)$.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases entropy, Voronoi diagram, convex distance function, hidden mixture of product distributions

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2021.8

Related Version *Full Version*: `https://arxiv.org/abs/2109.13460`

Funding Research of Cheng and Wong are supported by Research Grants Council, Hong Kong, China (project no. 16200317).

1 Introduction

Self-improving algorithms, proposed by Ailon et al. [1], is a framework for studying algorithmic complexity beyond the worst case. There is a *training phase* that allows some auxiliary structures about the input distribution to be constructed. In the *operation phase*, these auxiliary structures help to achieve an expected running time, called the *limiting complexity*, that may surpass the worst-case optimal time complexity.

Self-improving algorithms have been designed for *product distributions* [1, 11]. Let n be the input size. A product distribution $\mathscr{D} = (D_1, \ldots, D_n)$ consists of n distributions D_i such that the ith input item is drawn independently from D_i. It is possible that $D_i = D_j$ for some $i \neq j$, but the draws of the ith and jth input items are independent. No further information about $\mathscr{D}$ is given. Sorting, Delaunay triangulation, 2D maxima, and 2D convex hull have been studied for product distributions. For all four problems, the training phase uses $O(n^{\varepsilon})$ input instances, and the space complexity is $O(n^{1+\varepsilon})$. The limiting complexities of sorting and Delaunay triangulation are $O\big(\frac{1}{\varepsilon}n + \frac{1}{\varepsilon}H_{\text{out}}\big)$ for any $\varepsilon \in (0,1)$, where H_{out} is the entropy of the output distribution [1]. The limiting complexities for 2D maxima and 2D convex hull are $O(\text{OptM} + n)$ and $O(\text{OptC} + n\log\log n)$ respectively, where OptM and OptC are the expected depths of the optimal linear decision trees for the two problems [11].

Extensions that allow dependence among input items have been developed. One extension is that there is a *hidden partition* of $[n]$ into groups. The input items with indices in the kth group follow some *hidden functions* of a common parameter u_k. The parameters $u_1, u_2, \cdots$ follow a product distribution. The partition of $[n]$ is not given though. If the hidden functions are known to be linear, sorting can be solved in a limiting complexity of $O\big(\frac{1}{\varepsilon}n + \frac{1}{\varepsilon}H_{\text{out}}\big)$ after a training phase that takes $O(n^2 \log^3 n)$ time [8]. If it is only known that each hidden function has $O(1)$ extrema and the graphs of two functions intersect in $O(1)$ places (without knowing any of the functions, or any of these extrema and intersections), sorting can be solved in a limiting complexity of $O(n + H_{\text{out}})$ after an $\tilde{O}(n^3)$-time training phase [7]. For the Delaunay triangulation problem, if it is known that the hidden functions are bivariate polynomials of $O(1)$ degree (without knowing the polynomials), a limiting complexity of $O(n\alpha(n) + H_{\text{out}})$ can be achieved after a polynomial-time training phase [7].

Another extension is that the input instance I is drawn from a *hidden mixture* of at most m product distributions. That is, there are at most m product distributions $\mathscr{D}_1, \mathscr{D}_2, \ldots$ such that $\Pr[I \sim \mathscr{D}_a] = \lambda_a$ for some fixed positive value λ_a. The upper bound m is given, but no information about the λ_a's and the $\mathscr{D}_a$'s is provided. Sorting can be solved in a limiting complexity of $O\big(\frac{1}{\varepsilon}n \log m + \frac{1}{\varepsilon}H_{\text{out}}\big)$ after a training phase that takes $O(mn \log^2(mn) + m^{\varepsilon}n^{1+\varepsilon} \log(mn))$ time [8].

In this paper, we present a self-improving algorithm for constructing Voronoi diagrams under a convex distance function d_Q in $\mathbb{R}^2$, assuming that the input distribution is a hidden mixture of at most m product distributions. The convex distance function d_Q is induced by a given convex polygon Q of $O(1)$ size. The upper bound m is given, and we assume that $m = o(\sqrt{n})$. We also assume that for each product distribution $\mathscr{D}_a$ in the mixture, $\lambda_a = \Omega(1/n)$. Let $\varepsilon \in (0, 1)$ be a parameter fixed beforehand. The training phase uses $O(mn \log(mn))$ input instances and takes $O\big(mn \log^{O(1)}(mn) + m^{\varepsilon}n^{1+\varepsilon} \log(mn)\big)$ time. In the operation phase, given an input instance I, we can construct its Voronoi diagram $\mathrm{Vor}_Q(I)$ under d_Q in a limiting complexity of $O\big(\frac{1}{\varepsilon}n \log m + \frac{1}{\varepsilon}n2^{O(\log^* n)} + \frac{1}{\varepsilon}H\big)$, where H denotes the entropy of the distribution of the Voronoi diagram output. Note that $\Omega(H)$ is a lower bound of the expected running time of any comparison-based algorithm. Our algorithm also works for the Euclidean case, and the limiting complexity improves to $O\big(\frac{1}{\varepsilon}n \log m + \frac{1}{\varepsilon}H\big)$.

For simplicity, we will assume throughout the rest of this paper that the hidden mixture has exactly m product distributions. We give an overview of our method in the following.

We follow the strategy in [1] for computing a Euclidean Delaunay triangulation. The idea is to form a set S of sample points and build $\mathrm{Del}(S)$ and some auxiliary structures in the training phase so that any future input instance I can be merged quickly into $\mathrm{Del}(S)$ to form $\mathrm{Del}(S \cup I)$, and then $\mathrm{Del}(I)$ can be split off in $O(n)$ expected time. Merging I into $\mathrm{Del}(S)$ requires locating the input points in $\mathrm{Del}(S)$. The location distribution is gathered in the training phase so that distribution-sensitive point location can be used to avoid the logarithmic query time as much as possible. Modifying $\mathrm{Del}(S)$ efficiently into $\mathrm{Del}(S \cup I)$ requires that only $O(1)$ points in I fall into the same neighborhood in $\mathrm{Del}(S)$ in expectation.

In our case, since there are m product distributions, we will need a larger set S of mn sample points in order to ensure that only $O(1)$ points in I fall into the same neighborhood in $\mathrm{Vor}_Q(S)$ in expectation. But then merging I into $\mathrm{Vor}_Q(S)$ in the operation phase would be too slow because scanning $\mathrm{Vor}_Q(S)$ already requires $\Theta(mn)$ time. We need to extract a subset $R \subseteq S$ such that R has $O(n)$ size and R contains all points in S whose Voronoi cells conflict with the input points.

Still, we cannot afford to construct $\mathrm{Vor}_Q(R)$ in $O(n \log n)$ time. In the training phase, we form a metric d related to d_Q and construct a *net-tree* T_S for S under d [15]. In the operation phase, after finding the appropriate $R \subseteq S$, we use nearest common ancestor queries [20] to compress T_S in $O(n \log \log m)$ time to a subtree T_R for R that has $O(n)$ size. Next, we use T_R to construct a well-separated pair decomposition of R under d in $O(n)$ time [15], use the decomposition to compute the nearest neighbor graph of R under d in $O(n)$ time, and then construct $\mathrm{Vor}_Q(R)$ from the nearest neighbor graph in $O(n)$ expected time. The merging of I into $\mathrm{Vor}_Q(R)$ to form $\mathrm{Vor}_Q(R \cup I)$, and the splitting of $\mathrm{Vor}_Q(R \cup I)$ into $\mathrm{Vor}_Q(I)$ and $\mathrm{Vor}_Q(R)$ are obtained by transferring their analogous results in the Euclidean case [1, 6].

We have left out the expected time to locate the input points in $\mathrm{Vor}_Q(S)$. It is bounded by $O(1/\varepsilon)$ times the sum of the entropies of the point location outcomes. We show that $\mathrm{Vor}_Q(I)$ allows us to locate the input points in $\mathrm{Vor}_Q(S)$ in $O(n \log m + n2^{O(\log^* n)})$ time. Then, a result in [1] implies that the sum of the entropies of the point location outcomes is $O(n \log m + n2^{O(\log^* n)} + H)$. The expected running time is thus $O(\frac{1}{\varepsilon} n \log m + \frac{1}{\varepsilon} n2^{O(\log^* n)} + \frac{1}{\varepsilon} H)$, which dominates the limiting complexity. In the Euclidean case, $\mathrm{Vor}(I)$ allows us to locate the input points in $O(n \log m)$ time, so the limiting complexity improves to $O(\frac{1}{\varepsilon} n \log m + \frac{1}{\varepsilon} H)$.

Details and proofs that are omitted due to space constraint can be found in the full version of this paper [9].

2 Preliminaries

Let Q be a convex polygon that has $O(1)$ complexity and contains the origin in its interior. Let ∂ and $\mathrm{int}(\cdot)$ be the boundary and interior operators, respectively. So Q's boundary is ∂Q and its interior is $\mathrm{int}(Q)$. Let d_Q be the distance function induced by Q: $\forall\, x, y \in \mathbb{R}^2$, $d_Q(x, y) = \min\{\lambda \in [0, \infty) : y \in \lambda Q + x\}$. As Q may not be centrally symmetric (i.e., $x \in Q \iff -x \in Q$), d_Q may not be a metric.

The bisector of two points p and q is $\{x \in \mathbb{R}^2 : d_Q(p, x) = d_Q(q, x)\}$, which is an open polygonal curve of $O(1)$ size. The Voronoi diagram of a set Σ of n points, $\mathrm{Vor}_Q(\Sigma)$, is a partition of $\mathbb{R}^2$ into interior-disjoint cells $V_p(\Sigma) = \{x \in \mathbb{R}^2 : \forall q \in \Sigma,\ d_Q(p, x) \leq d_Q(q, x)\}$ for all $p \in \Sigma$. There are algorithms for constructing $\mathrm{Vor}_Q(\Sigma)$ in $O(n \log n)$ time [10, 18].

$V_p(\Sigma)$ is simply connected and star-shaped with respect to p [10]. We use $N_p(\Sigma)$ to denote the set of Voronoi neighbors of p in $\mathrm{Vor}_Q(\Sigma)$. The Voronoi edges of $\mathrm{Vor}_Q(\Sigma)$ form a planar graph of $O(|\Sigma|)$ size. Each Voronoi edge is a polygonal line, and we call its internal vertices *Voronoi edge bends*. We use V_Σ to denote the set of Voronoi edge bends and Voronoi vertices in $\mathrm{Vor}_Q(\Sigma)$. For the infinite Voronoi edges, their endpoints at infinity are included in V_Σ.

Define $Q^* = \{-x : x \in Q\}$. For any points $x, y \in \mathbb{R}^2$, $d_{Q^*}(x, y) = d_Q(y, x)$. At any point x on a Voronoi edge of $\mathrm{Vor}_Q(\Sigma)$ defined by $p, q \in \Sigma$, there exists $\lambda \in (0, \infty)$ such that $d_{Q^*}(x, p) = d_Q(p, x) = d_Q(q, x) = d_{Q^*}(x, q) = \lambda$ and $d_{Q^*}(x, s) = d_Q(s, x) \geq \lambda$ for all $s \in \Sigma$. Hence, $\{p, q\} \subset \partial(\lambda Q^* + x)$ and $\mathrm{int}(\lambda Q^* + x) \cap \Sigma = \emptyset$, i.e., an "empty circle property".

Take a point x. Consider the largest homothetic[1] copy Q^*_x of Q^* centered at x such that $\mathrm{int}(Q^*_x) \cap \Sigma = \emptyset$. If we insert a new point q to Σ, we say that q *conflicts with* x if $q \in Q^*_x$. We say that q *conflicts with* a cell $V_p(\Sigma)$ if q conflicts with some point in $V_p(\Sigma)$. Clearly, $V_p(\Sigma)$ must be updated by the insertion of q. We use $V_\Sigma|_q$ to denote the subset of V_Σ that conflict with q. The Voronoi edge bends and Voronoi vertices in $V_\Sigma|_q$ will be destroyed by the insertion of q.

We make three general position assumptions. First, no two sides of Q are parallel. Second, for every pair of input points, their support line is not parallel to any side of Q. Third, no four input points lie on the boundary of any homothetic copy of Q^*, which implies that every Voronoi vertex has degree three.

It is much more convenient if all Voronoi cells of the input points are bounded. We assume that all possible input points appear in some fixed bounding square $\mathcal{B}$ centered at the origin. We place $O(1)$ dummy points outside $\mathcal{B}$ so that all Voronoi cells of the input

[1] A homothetic copy of a shape is a scaled and translated copy of it.

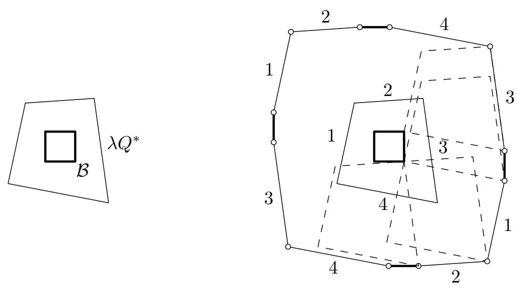

Figure 1 The left image shows the bounding square $\mathcal{B}$ and the large enclosing λQ^*. In the right image, we slide a copy of λQ^* around $\mathcal{B}$ to generate the outer convex polygon. The dashed polygon demonstrates the sliding of λQ^* around $\mathcal{B}$. The bold edges on this convex polygon are translates of the boundary edges of $\mathcal{B}$. Every edge of λQ^* has two translational copies too as labelled.

points are bounded, and their portions inside $\mathcal{B}$ remain the same as before. Refer to Figure 1. Take λQ^* for some large enough $\lambda \in \mathbb{R}$ such that for every point $x \in \mathcal{B}$, $\lambda Q^* + x$ contains $\mathcal{B}$. Refer to the left image in Figure 1. We slide a copy of λQ^* around $\mathcal{B}$ to generate the outer convex polygon. The dashed polygon demonstrates the sliding of λQ^* around $\mathcal{B}$. This outer polygon contains a translational copy of every edge of $\mathcal{B}$ and two translational copies of every edge of λQ^*. We add the vertices of this outer polygon as dummy points. Any homothetic copy of Q^* that intersects $\mathcal{B}$ cannot be expanded indefinitely without containing some of these dummy points. So all Voronoi cells of input points are bounded. For each point $x \in \mathcal{B}$, since the dummy points lie outside $\lambda Q^* + x$ and $\mathcal{B} \subseteq \lambda Q^* + x$ (i.e., $\lambda Q^* + x$ is not empty of the input points), the portion of the Voronoi diagram inside $\mathcal{B}$ is unaffected by the dummy points.

3 Training phase

Sample set S. Take $mn\ln(mn)$ instances $I_1, I_2, \ldots, I_{mn\ln(mn)}$. Define $x_1, \ldots, x_{mn\ln(mn)}$ by taking the p_1's in $I_1, \ldots, I_{m\ln(mn)}$ to be $x_1, \ldots, x_{m\ln(mn)}$, p_2's in $I_{m\ln(mn)+1}, \ldots, I_{2m\ln(mn)}$ to be $x_{m\ln(mn)+1}, \ldots, x_{2m\ln(mn)}$, and so on. The set S of sample points includes a $\frac{1}{mn}$-net of the x_i's with respect to the family of homothetic copies of Q^*, as well as the $O(1)$ dummy points. The set S has $O(mn)$ points and can be constructed in $O(mn\log^{O(1)}(mn))$ time as homothetic copies of Q^* are pseudo-disks [1, 19].

Point location. Compute $\mathrm{Vor}_Q(S)$ and triangulate it by connecting each $p \in S$ to $V_S \cap \partial V_p(S)$, i.e., the Voronoi edge bends and Voronoi vertices in $\partial V_p(S)$. For unbounded Voronoi cells, we view the infinite Voronoi edges as leading to some vertices at infinity; an extra triangulation edge that goes between two infinite Voronoi edges also leads to a vertex at infinity, giving rise to unbounded triangles. Figure 2 shows an example.

Construct a point location structure L_S for the triangulated $\mathrm{Vor}_Q(S)$ with $O(\log(mn))$ query time [13]. Take another $m^\varepsilon n^\varepsilon$ input instances and use L_S to locate the points in these input instances in the triangulated $\mathrm{Vor}_Q(S)$. For every $i \in [n]$ and every triangle t, we compute $\tilde{\pi}_{i,t}$ to be the ratio of the frequency of t hit by p_i to $m^\varepsilon n^\varepsilon$, which is an estimate of $\Pr[p_i \in t]$. For each $i \in [n]$, form a subdivision $\mathcal{S}_i$ that consists of triangles with positive $\tilde{\pi}_{i,t}$'s, triangulate the exterior of $\mathcal{S}_i$, and give these new triangles a zero estimated probability. Set the weight of each triangle in $\mathcal{S}_i$ to be the maximum of $(mn)^{-\varepsilon}$ and its estimated probability. Construct a distribution-sensitive point location structure L_i for $\mathcal{S}_i$ based on the triangle weights [2, 16]. Note that L_i has $O(m^\varepsilon n^\varepsilon)$ size, and locating a point in a triangle $t \in \mathcal{S}_i$ takes $O\big(\log \frac{W_i}{w_t}\big)$ time, where w_t is the weight of t and W_i is the total weight in $\mathcal{S}_i$.

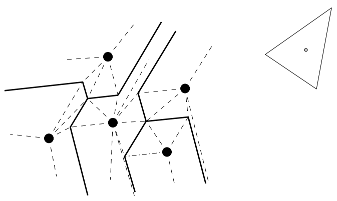

Figure 2 Part of the triangulation of a Voronoi diagram induced by the triangle shown with a gray center. The solid edges form the Voronoi diagram. The dashed edges refine it into a triangulation.

For any input instance $(p_1, \ldots, p_n)$ in the operation phase, we will query L_i to locate p_i in the triangulated $\mathrm{Vor}_Q(S)$, which may fail if p_i falls into a triangle with zero estimated probability. If the search fails, we query L_S to locate p_i.

Net-tree. We first define a metric that is induced by a centrally symmetric convex polygon. Define $\hat{Q} = \{x - y : x, y \in Q^*\}$, i.e., the Minkowski sum of Q^* and $-Q^*$, or equivalently Q^* and Q. It is centrally symmetric by definition. It can be visualized as the region covered by all possible placements of Q^* that has the origin in the polygon boundary. Since $\hat{Q}$ is a Minkowski sum, its number of vertices is within a constant factor of the total number of vertices of Q^* and $-Q^*$, which is $O(1)$.

Let d be the metric induced by the centrally symmetric convex polygon $\hat{Q}$, which is a *doubling metric* – there is a constant $\lambda > 0$ such that for any point $x \in \mathbb{R}^2$ and any positive number r, the ball with respect to d centered at x with radius r can be covered by λ balls with respect to d of radius $r/2$.

Given a set of points P, a *net-tree* for P with respect to d [15] is an analog of the well-separated pair decomposition for Euclidean spaces [4]. It is a rooted tree whose leaves are the points in P. For each node v, let *parent*(v) denote its parent, and let P_v denote the subset of P at the leaves that descend from v. Every tree node v is given a representative point p_v and an integer level ℓ_v. Let $\tau \geq 11$ be a fixed constant. Let $B(x, h)$ denote the ball $\{y \in \mathbb{R}^2 : d(x, y) \leq h\}$. By the results in [15] (Definition 2.1 and the remark that follows Proposition 2.2), the following properties are satisfied by a net-tree:

(a) $p_v \in P_v$.
(b) For every non-root node v, $\ell_v < \ell_{parent(v)}$, and if v is a leaf, then $\ell_v = -\infty$.
(c) Every internal node has at least two and at most a constant number of children.
(d) For every node v, $B\big(p_v, \frac{2\tau}{\tau-1} \cdot \tau^{\ell_v}\big)$ contains P_v.
(e) For every non-root node v, $B\big(p_v, \frac{\tau-5}{2\tau-2} \cdot \tau^{\ell_{parent(v)}-1}\big) \cap P \subset P_v$.
(f) For every internal node v, there is a child w of v such that $p_w = p_v$.

Clusters. We construct a *net-tree* T_S for S in $O(mn \log(mn))$ expected time [15]. We define *clusters* as follows. Label all leaves of T_S as unclustered initially. Select the leftmost m unclustered leaves of T_S; if there are fewer than m such leaves, select them all. Find the subtree rooted at a node v of T_S that contains the selected unclustered leaves, but no child subtree of v contains them all. We call the subtree rooted at v a cluster and label all its leaves clustered. Then, we repeat the above until all leaves of T_S are clustered. By construction, the clusters are disjoint, each cluster has $O(m)$ nodes, and there are $O(n)$ clusters in T_S.

We assign nodes in each cluster a unique cluster index in the range $[1, O(n)]$. We also assign each node of a cluster three indices from the range $[1, O(m)]$ according to its rank in the preorder, inorder, and postorder traversals of that cluster. The preorder and postorder indices allow us to tell in $O(1)$ time whether two nodes are an ancestor-descendant pair.

We keep an initially empty van Emde Boas tree EB_c [21] with each cluster c. The universe for EB_c is the set of leaves in the cluster c, and the inorder of these leaves in c is the total order for EB_c. We also build a nearest common ancestor query data structure for each cluster [20]. The nearest common ancestor query of any two nodes can be reported in $O(\log\log m)$ time.

Planar separator. $\mathrm{Vor}_Q(S)$ is a planar graph of $O(mn)$ size with all Voronoi edge bends and Voronoi vertices as graph vertices. By a recursive application of the planar separator theorem, one can produce an m^2-*division* of $\mathrm{Vor}_Q(S)$: it is divided into $O(n/m)$ regions, each region contains $O(m^2)$ vertices, and the boundary of each region contains $O(m)$ vertices [14].

Extract the subset $B \subset S$ of points whose Voronoi cell boundaries contain some region boundary vertices in the m^2-division. So $|B| = O(m \cdot n/m) = O(n)$. Compute $\mathrm{Vor}_Q(B)$ and triangulate it as in triangulating $\mathrm{Vor}_Q(S)$. By our choice of B, the region boundaries in the m^2-division of $\mathrm{Vor}_Q(S)$ form a subgraph of $\mathrm{Vor}_Q(B)$. Label in $O(n)$ time the Voronoi edge bends and Voronoi vertices in $\mathrm{Vor}_Q(B)$ whether they exist in $\mathrm{Vor}_Q(S)$.

We construct point location data structures for every region Π in the m^2-division as follows. For every boundary vertex w of Π, let Q^*_w be the largest homothetic copy of Q^* centered at w such that $\mathrm{int}(Q^*_w) \cap B = \emptyset$. These Q^*_w's form an arrangement of $O(m^2)$ complexity, and we construct a point location data structure that allows a point to be located in this arrangement in $O(\log m)$ time. We also construct a point location data structure for the portion of the triangulated $\mathrm{Vor}_Q(S)$ inside Π. Since the region has $O(m^2)$ complexity, this point location data structure can return in $O(\log m)$ time the triangle in the triangulated $\mathrm{Vor}_Q(S)$ that contains a point inside Π.

Output and performance. The following result summarizes the output and performance of the training phase. The proof of Lemma 1(a) is similar to an analogous result for sorting in [8].

▶ **Lemma 1.** *Let $\mathscr{D}_a$, $a \in [m]$, be the distributions in the hidden mixture. The training phase computes the following structures in $O(mn \log^{O(1)}(mn) + m^\varepsilon n^{1+\varepsilon} \log(mn))$ time.*

(i) *A set S of $O(mn)$ points and $\mathrm{Vor}_Q(S)$. It holds with probability at least $1 - 1/n$ that for any $a \in [1, m]$ and any $v \in V_S$, $\sum_{i=1}^n \Pr[X_{iv} \,|\, I \sim \mathscr{D}_a] = O(1/m)$, where $X_{iv} = 1$ if $p_i \in I$ conflicts with v and $X_{iv} = 0$ otherwise.*

(ii) *Point location structures L_S and L_i for each $i \in [n]$ that allow us to locate p_i in the triangulated $\mathrm{Vor}_Q(S)$ in $O\big(\frac{1}{\varepsilon}H(t_i)\big)$ expected time, where t_i is the random variable that represents the point location outcome, and $H(t_i)$ is the entropy of the distribution of t_i.*

(iii) *A net-tree T_S for S, the $O(n)$ clusters in T_S, the initially empty van Emde Boas trees for the clusters, and the nearest common ancestor data structures for the clusters.*

(iv) *An m^2-division of $\mathrm{Vor}_Q(S)$, the subset $B \subseteq S$ of $O(n)$ points whose Voronoi cell boundaries contain some region boundary vertices in the m^2-division, $\mathrm{Vor}_Q(B)$, and the point location data structures for the regions in the m^2-division.*

Lemma 1(a) leads to Lemma 2 below, which implies that for any $v \in V_S$, if we feed the input points that conflict with v to a procedure that runs in quadratic time in the worst case, the expected running time of this procedure over all points in V_S is $O(n)$. The proof of Lemma 2 is just an algebraic manipulation of the probabilities.

▶ **Lemma 2.** *For every $v \in V_S$, let Z_v be the subset of input points that conflict with v. It holds with probability at least $1 - O(1/n)$ that $\sum_{v \in V_S} \mathrm{E}\big[|Z_v|^2\big] = O(n)$.*

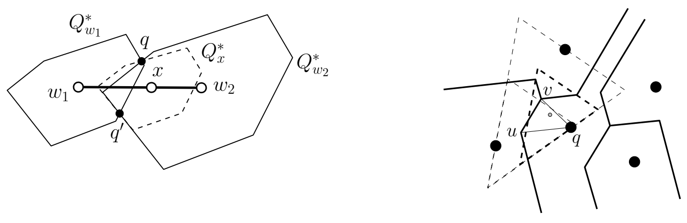

Figure 3 (a) The points q and q' define a Voronoi edge, and w_1 and w_2 are two adjacent Voronoi edge bends or Voronoi vertices on this edge. At any point x between w_1 and w_2, the polygon Q^*_x (shown dashed) is a subset of $Q^*_{w_1} \cup Q^*_{w_2}$. (b) A triangle quv in the triangulated Voronoi diagram in Figure 2 is shown. If a point p conflicts with the white dot (i.e., lies inside the bold dashed triangle), then p conflicts with u or v (i.e., lies inside one of the two light dashed circles.)

We state two technical results. Figure 3(a) and (b) illustrate these two lemmas.

▶ **Lemma 3.** *Consider* $\mathrm{Vor}_Q(Y)$ *for some point set* Y*. For any point* $x \in \mathbb{R}^2$*, let* Q^*_x *be the largest homothetic copy of* Q^* *centered at* x *such that* $\mathrm{int}(Q^*_x) \cap Y = \emptyset$*. Let* w_1 *and* w_2 *be two adjacent Voronoi edge bends or Voronoi vertices in* $\mathrm{Vor}_Q(Y)$*. For any point* $x \in w_1w_2$*,* $Q^*_x \subseteq Q^*_{w_1} \cup Q^*_{w_2}$*. The same property holds if* w_1 *and* w_2 *are Voronoi vertices connected by a Voronoi edge, and* x *lies on that Voronoi edge.*

▶ **Lemma 4.** *Let* q *be a point in some point set* Y*. Let* quv *be a triangle in the triangulated* $\mathrm{Vor}_Q(Y)$*. If a point* $p \notin Y$ *conflicts with a point in* quv*, then* p *conflicts with* u *or* v*. Hence, if* p *conflicts with* $V_q(Y)$*,* p *conflicts with a Voronoi edge bend or Voronoi vertex in* $\partial V_q(Y)$*.*

4 Operation phase

Given an instance $I = (p_1, \cdots, p_n)$, we construct $\mathrm{Vor}_Q(I)$ using the pseudocode below.

Algorithm 1 Operation Phase.

```
1. For each $i \in [n]$, query $L_i$ to find the triangle $t_i$ in the triangulated $\mathrm{Vor}_Q(S)$ that contains
   $p_i$, and if the search fails, query $L_S$ to find $t_i$.
2. For each $i \in [n]$, search $\mathrm{Vor}_Q(S)$ from $t_i$ to find $V_S|_{p_i}$, i.e., the subset of $V_S$ that conflict
   with $p_i$. This also gives the subset of $S$ whose Voronoi cells conflict with the input points.
   Let $R$ be the union of this subset of $S$ and the set of representative points of all cluster
   roots in $T_S$.
3. Compute the compression $T_R$ of $T_S$ to $R$.
4. Construct the nearest neighbor graph $1\text{-NN}_R$ under the metric $d$ from $T_R$.
5. Compute $\mathrm{Vor}_Q(R)$ from $1\text{-NN}_R$.
6. Modify $\mathrm{Vor}_Q(R)$ to produce $\mathrm{Vor}_Q(R \cup I)$.
7. Split $\mathrm{Vor}_Q(R \cup I)$ to produce $\mathrm{Vor}_Q(I)$ and $\mathrm{Vor}_Q(R)$. Return $\mathrm{Vor}_Q(I)$.
```

We analyze step 1 in Section 4.1, steps 2 and 3 in Section 4.2, steps 4 and 5 in Section 4.3, and steps 6 and 7 in Section 4.4. Step 1 is the most time-consuming; all other steps run in $O(n)$ expected time.

4.1 Point location

By Lemma 1(b), step 1 runs in $O\big(\sum_{i=1}^{n} \frac{1}{\varepsilon} H(t_i)\big)$ expected time, which is $O\big(\frac{1}{\varepsilon} n \log m + \frac{1}{\varepsilon} H(t_1, \ldots, t_n)\big)$ as we will show later. By Lemma 5 below, if there is an algorithm that can use $\mathrm{Vor}_Q(I)$ to determine $t_1, \ldots, t_n$ in $c(n)$ expected time, then $H(t_1, \ldots, t_n) = O(c(n) + H)$, implying that step 1 takes $O\big(\frac{1}{\varepsilon}(n \log m + c(n) + H)\big)$ expected time. Any preprocessing cost of S is excluded from $c(n)$. We present such an algorithm.

▶ **Lemma 5** (Lemma 2.3 in [1]). *Let $\mathscr{D}$ be a distribution on a universe $\mathcal{U}$. Let $X : \mathcal{U} \to \mathcal{X}$, and let $Y : \mathcal{U} \to \mathcal{Y}$ be two random variables. Suppose that there is a comparison-based algorithm that computes a function $f : (I, X(I)) \to Y(I)$ in C expected comparisons over $\mathscr{D}$ for every $I \in \mathcal{U}$. Then $H(Y) = C + O(H(X))$.*

Recall that we have computed in the training phase the subset $B \subseteq S$ whose Voronoi cell boundaries contain some region boundary vertices in the m^2-division of $\mathrm{Vor}_Q(S)$. Note that $|B| = O(n)$. We have also computed $\mathrm{Vor}_Q(B)$ and point location data structures associated with the regions in the m^2-division. We use $\mathrm{Vor}_Q(B)$ and these point location data structures determines $t_1, \ldots, t_n$ as follows.

- Task 1: Merge $\mathrm{Vor}_Q(B)$ with $\mathrm{Vor}_Q(I)$ to form the triangulated $\mathrm{Vor}_Q(B \cup I)$.
- Task 2: Use $\mathrm{Vor}_Q(S)$, $\mathrm{Vor}_Q(B)$, and $\mathrm{Vor}_Q(B \cup I)$ to find the triangles $t_1, \ldots, t_n$.

We discuss these two tasks in the following.

Task 1. For every point $p \in B$, define a polygonal cone surface $C_p = \big\{(a, b, d_Q(p, (a, b))) : (a, b) \in \mathbb{R}^2\big\}$. Each horizontal cross-section of C_p is a scaled copy of Q centered at p. The triangulated $\mathrm{Vor}_Q(B)$ is the vertical projection of the lower envelope of $\{C_p : p \in B\}$, denoted by $\mathcal{L}(B)$. Similarly, $\mathcal{L}(I)$ projects to $\mathrm{Vor}_Q(I)$. We take the lower envelope of $\mathcal{L}(B)$ and $\mathcal{L}(I)$ to form $\mathcal{L}(B \cup I)$ which projects to $\mathrm{Vor}_Q(B \cup I)$. We do so in $O(n 2^{O(\log^* n)})$ expected time with a randomized algorithm that is based on an approach proposed and analyzed by Chan [5, Section 4].

Task 2. Suppose that for an input point $p_i \in I$, we have determined some subset B_i that satisfies $B \subseteq B_i \subseteq S$, and we have computed a Voronoi edge bend or Voronoi vertex v_i in $\mathrm{Vor}_Q(B_i)$ that conflicts with p_i and is known to be in V_S or not.

If $v_i \in V_S$, we search $\mathrm{Vor}_Q(S)$ from v_i to find $V_S|_{p_i}$ (i.e., the subset of V_S that conflict with p_i), which by Lemma 3 also gives the triangle t_i in the triangulated $\mathrm{Vor}_Q(S)$ that contains p_i. By Lemma 2, the expected total running time of this procedure over all input points is $O(n)$.

Suppose that $v_i \notin V_S$. So v_i is not a region boundary vertex in the m^2-division of $\mathrm{Vor}_Q(S)$, i.e., v_i lies inside a region in the m^2-division of $\mathrm{Vor}_Q(S)$, say Π. For each boundary vertex w of Π, let Q^*_w be the largest homothetic copy of Q^* centered at w such that $\mathrm{int}(Q^*_w) \cap B = \emptyset$. These Q^*_w's form an arrangement of $O(m^2)$ complexity, and we locate p_i in this arrangement in $O(\log m)$ time. It tells us whether $p_i \in Q^*_w$ for some boundary vertex w of Π. If so, then p_i conflicts with w, which belongs to V_S, and we search $\mathrm{Vor}_Q(S)$ from w to find $V_S|_{p_i}$ and hence the triangle t_i in the triangulated $\mathrm{Vor}_Q(S)$ that contains p_i. Otherwise, p_i must lie inside Π in order to conflict with v_i inside Π without conflicting with any boundary vertex of Π. So we do a point location in $O(\log m)$ time to locate p_i in the portion of the triangulated $\mathrm{Vor}_Q(S)$ inside Π. This gives t_i.

How do we compute v_i for p_i? We discuss this computation and provide more details of Step 2 in the full version [9]. The following lemma summarizes the result that follows from the discussion above.

▶ **Lemma 6.** *Given* $\mathrm{Vor}_Q(I)$*, the triangles* $t_1, \ldots, t_n$ *in the triangulated* $\mathrm{Vor}_Q(S)$ *that contain* $p_1, \ldots, p_n \in I$ *can be computed in* $O\left(n \log m + n2^{O(\log^* n)}\right)$ *expected time.*

▶ **Lemma 7.** *Step* 1 *of the operation phase takes* $O\big(\frac{1}{\varepsilon}(n \log m + n2^{O(\log^* n)} + H)\big)$ *expected time, where* H *is the entropy of the distribution of* $\mathrm{Vor}_Q(I)$*.*

Proof. Let $A \in [1, m]$ be a random variable that indicates which distribution in the mixture generates the input instance. By the chain rule for conditional entropy [22, Proposition 2.23], $H(t_i) \leq H(t_i) + H(A|t_i) = H(t_i, A) = H(A) + H(t_i|A)$. It is known that $H(A) \leq \log_2(\text{domain size of } A) = \log_2 m$ [22, Theorem 2.43]. Thus, $\sum_{i=1}^n H(t_i) \leq n \log_2 m + \sum_{i=1}^n H(t_i|A)$. The variables $t_1|A, \ldots, t_n|A$ are mutually independent. So $\sum_{i=1}^n H(t_i|A) = H(t_1, \ldots, t_n|A)$. Since entropy is not increased by conditioning [22, Theorem 2.38], we get $\sum_{i=1}^n H(t_i|A) = H(t_1, \ldots, t_n|A) \leq H(t_1, \ldots, t_n)$. By Lemma 6, we can determine $t_1, \ldots, t_n$ using $\mathrm{Vor}_Q(I)$ in $O(n \log m + n2^{O(\log^* n)})$ expected time. So $H(t_1, \ldots, t_n) = O(n \log m + n2^{O(\log^* n)} + H)$ by Lemma 5, where H is the entropy of the distribution of $\mathrm{Vor}_Q(I)$. ◀

In the Euclidean metric, merging $\mathrm{Vor}(B)$ and $\mathrm{Vor}(I)$ into $\mathrm{Vor}(B \cup I)$ can be reduced to finding the intersection of two convex polyhedra of $O(n)$ size in $\mathbb{R}^3$, which can be solved in $O(n)$ time [5]. So the expected running time of step 1 improves to $O\big(\frac{1}{\varepsilon}(n \log m + H)\big)$.

4.2 Construction of R

Step 1 determines the triangle t_i in the triangulated $\mathrm{Vor}_Q(S)$ that contains $p_i \in I$. We search $\mathrm{Vor}_Q(S)$ from t_i to find $V_S|_{p_i}$, which takes $O\big(|V_S|_{p_i}|\big)$ time [18]. This search also gives the Voronoi cells that conflict with p_i. The total time over all $i \in [n]$ is $O\big(\sum_{v \in V_S} |Z_v|\big)$, where Z_v is the subset of input points that conflict with v. Since R includes all sites whose cells conflict with the input points and the representative points of all cluster roots in T_S, we have $|R| \leq \sum_{v \in V_S} |Z_v| + O(n)$. The following result follows from Lemma 2.

▶ **Lemma 8.** *The set* R *has* $O(n)$ *expected size. Step 2 of the operation phase constructs* R *in* $O(n)$ *expected time.*

4.3 Extraction of $\mathrm{Vor}_Q(R)$

4.3.1 Construction of T_R

We define a *compression* of a net-tree T. Select a subset U of leaves in T. Let $T' \subseteq T$ be the minimal subtree that spans U. Bypass all internal nodes in T' that have only one child. *The resulting tree is the compression of* T *to* U. The following result is an easy observation.

▶ **Lemma 9.** *Let* T *be a net-tree. Let* T_1 *be the compression of* T *to a subset* U_1 *of leaves. The compression of* T_1 *to any subset* U_2 *of leaves in* T_1 *can also be obtained by a compression of* T *to* U_2*.*

Conceptually, T_R is defined as follows. Select all leaves of T_S that are points in R, and T_R is the compression of T_S to these selected leaves. Since R includes the representative points of all cluster roots, all ancestors of the cluster roots in T_S will survive the compression and exist as nodes in T_R. The compression affects the clusters only. More precisely, for each

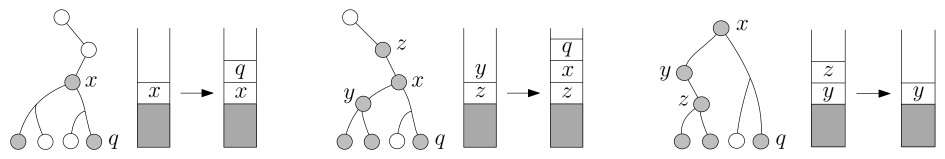

Figure 4 Three different cases in the manipulation of the stack. The tree shown is a part of T_S. The gray nodes are nodes in T_c. The gray leaves are leaves in R_c.

cluster c in T_S, we select its leaves that are points in R and compute the compression T_c of the cluster c to these selected leaves. Substituting every cluster c in T_S by T_c gives the desired T_R. It remains to discuss how to compute the T_c's.

We divide R in $O(n)$ expected time into sublists $R_1, R_2, \ldots$ such that R_c consists of the points that are leaves in cluster c. Recall that every cluster c has an initially empty van Emde Boas tree EB_c for its leaves in left-to-right order. For each R_c, we insert all leaves in R_c into EB_c and then repeatedly perform extract-min on EB_c. This gives in $O(|R_c| \log\log m)$ time a sorted list R'_c of the leaves in R_c according to their left-to-right order in the cluster c.

If $|R'_c| = 1$, then T_c consists of the single leaf in R_c. Suppose that $|R'_c| \geq 2$. We construct T_c using a stack. Initially, T_c is a single node which is the first leaf in R'_c. The stack stores the nodes on the rightmost root-to-leaf path in the current T_c, with the root at the stack bottom and the leaf at the stack top. When we scan the next leaf q in R'_c, we find in cluster c the nearest common ancestor x of q and q's predecessor in R'_c. This takes $O(\log\log m)$ time [20]. If we see x at the stack top, we add q as a new leaf to T_c with x as its parent, and then we push q onto the stack. Refer to the left image in Figure 4. If we see an ancestor z of x at the stack top, let y be the node that was immediately above z in the stack and was just popped, we make x the rightmost child of z in T_c (which was y previously), we also make y and q the left and right children of x respectively, and then we push x and q in this order onto the stack. Refer to the middle image in Figure 4. If neither of the two conditions above happens and the stack is not empty, we pop the stack and repeat. Refer to the right image in Figure 4. If the stack becomes empty, we make x the new root of T_c, we also make the old root of T_c and q the left and right children of x respectively, and then we push x and q in this order onto the stack. The construction of T_c takes $O(|R_c| \log\log m)$ time.

▶ **Lemma 10.** *The compression T_R of T_S to R can be computed in $O(n \log\log m)$ time.*

4.3.2 Construction of the k-nearest neighbor graph

Let X be any subset of S. Assume that the compression T_X of T_S to X is available. We show how to use T_X to construct in $O(k|X|)$ time the k-nearest neighbor graph of X under the metric d. We denote this graph by k-NN$_X$. We will use the *well-separated pair decomposition* or WSPD for short. For any $c \geq 1$, a set $\big\{\{A_1, B_1\}, \ldots, \{A_s, B_s\}\big\}$ is a c-WSPD of X under d if the following properties are satisfied:

- $\forall\, i,\ \ A_i, B_i \subseteq X$.
- $\forall$ distinct $x, y \in X$, $\exists\, i$ such that $\{x, y\} \in \big\{\{a, b\} : a \in A_i \wedge b \in B_i\big\}$.
- $\forall\, i$, the maximum of the diameters of A_i and B_i under d is less than $\frac{1}{c} \cdot d(A_i, B_i)$. It implies that $A_i \cap B_i = \emptyset$.

It is known that a c-WSPD has $O(c^{O(1)}|X|)$ size and can be constructed in $O(c^{(O(1)}|X|)$ time from a net-tree for X [15]. The same method works for a compression T_X of T_S to X, giving a c-WPSD of $O((c+1)^{O(1)}|X|)$ size in $O((c+1)^{O(1)}|X|)$ time. To compute k-NN$_X$, we transfer a strategy in [4] for constructing a Euclidean k-nearest neighbor graph using a WSPD.

▶ **Lemma 11.** *Given the compression T_X of T_S to any subset $X \subseteq S$, the k-NN_X can be constructed in $O(k|X|)$ time.*

The next result shows that the vertex degree of 1-NN_X is $O(1)$.

▶ **Lemma 12.** *For any subset $X \subseteq S$, every vertex in* 1-NN_X *has $O(1)$ degree, and adjacent vertices in* 1-NN_X *are Voronoi neighbors in* $\mathrm{Vor}_Q(X)$.

4.3.3 $\mathrm{Vor}_Q(R)$ from the nearest neighbor graph

We show how to construct $\mathrm{Vor}_Q(R)$ in $O(n)$ expected time using 1-NN_R. We use the following recursive routine which is similar to the one in [3] for constructing an Euclidean Delaunay triangulation from the Euclidean nearest neighbor graph. The top-level call is $\mathrm{VorNN}(R, T_R)$.

Algorithm 2 $\mathrm{VorNN}(Y, T_Y)$.

```
1. If $|Y| = O(1)$, compute $\mathrm{Vor}_Q(Y)$ directly and return.
2. Compute 1-$\mathrm{NN}_Y$ under the metric $d$ using $T_Y$.
3. Let $X \subseteq Y$ be a random sample such that $X$ meets every connected component of 1-$\mathrm{NN}_Y$,
   and $\Pr[p \in X] = 1/2$ for every $p \in Y$.
4. Compute the compression $T_X$ of $T_Y$ to $X$.
5. Call $\mathrm{VorNN}(X, T_X)$ to compute $\mathrm{Vor}_Q(X)$.
6. Using 1-$\mathrm{NN}_Y$ as a guide, insert the points in $Y \setminus X$ into $\mathrm{Vor}_Q(X)$ to form $\mathrm{Vor}_Q(Y)$.
```

There are two differences from [3]. First, we use a compression T_Y of T_S to compute 1-NN_Y in step 2, which takes $O(|Y|)$ time by Lemma 11. Second, we need to compress T_Y to T_X in step 4. This compression works in almost the same way as described in Section 4.3.1 except that we can afford to traverse T_Y in $O(|Y|)$ time to answer all nearest common ancestor queries required for constructing T_X. Thus, step 4 runs in $O(|Y|)$ time.

Step 3 is implemented as follows [3]. Form an arbitrary maximal matching of 1-NN_Y. By the definition of 1-NN_Y, each connected component of 1-NN_Y contains at least one matched pair. Randomly select one point from every matched pair. Then, among those unmatched points in 1-NN_Y, select each one with probability 1/2 uniformly at random. The selected points form the subset X required in step 3. The time needed is $O(|Y|)$.

In step 6, for each $p \in Y \setminus X$ that is connected to some point $q \in X$ in 1-NN_Y, p and q are Voronoi neighbors in $\mathrm{Vor}_Q(Y)$ by Lemma 12. So p conflicts with a point in $V_q(X)$. By Lemma 4, p conflicts with a Voronoi edge bend or Voronoi vertex in $\partial V_q(X)$, which can be found in $O\big(|\partial V_q(X)|\big)$ time. After finding a Voronoi edge bend or Voronoi vertex v in $\partial V_q(X)$ that conflicts with p, we search $\mathrm{Vor}_Q(X)$ from v to find all Voronoi edge bends and Voronoi vertices that conflict with p. In the same search of $\mathrm{Vor}_Q(X)$, we modify $\mathrm{Vor}_Q(X)$ into $\mathrm{Vor}_Q\big(X \cup \{p\}\big)$ as in a randomized incremental construction [18]. By the Clarkson-Shor analysis [12], the expected running time of the search of $\mathrm{Vor}_Q(X)$ and the Voronoi diagram modification over the insertions of all points in $Y \setminus X$ is $O(|Y|)$. We spend $O\big(|\partial V_q(X)|\big)$ time to find v. It translates to an $O(1)$ charge at each vertex of $V_q(X)$. This charging happens only for q's neighbors in 1-NN_Y. By Lemma 12, there are $O(1)$ such neighbors of q, so the charge at each vertex of $V_q(X)$ is $O(1)$. Moreover, if a vertex of $V_q(X)$ is destroyed by the insertion of a point from $Y \setminus X$, that vertex will not reappear. So the $O\big(|\partial V_q(X)|\big)$ cost is absorbed by the structural changes which is already taken care of by the Clarkson-Shor analysis. Unwinding the recursion gives a total expected running time of $O(|R| + |R|/2 + |R|/4 + \cdots) = O(|R|)$.

▶ **Lemma 13.** $\mathrm{VorNN}(R, T_R)$ *computes* $\mathrm{Vor}_Q(R)$ *in $O(|R|)$ expected time.*

4.4 Computing $\mathrm{Vor}_Q(I)$ from $\mathrm{Vor}_Q(R)$ and I

Let q be a point in R. Let $v_1, v_2, \ldots$ be the vertices of $V_q(R)$, in clockwise order, which may be Voronoi edge bends or Voronoi vertices. Let $Q^*_{v_i}$ denote the largest homothetic copy of Q^* centered at v_i such that $\mathrm{int}(Q^*_{v_i}) \cap R = \emptyset$. Let $Z_{v_i} = Q^*_{v_i} \cap I$ where I is an input instance.

▶ **Lemma 14.** *The portions of* $\mathrm{Vor}_Q(R \cup I)$ *and* $\mathrm{Vor}_Q\big(\{q\} \cup Z_{v_i} \cup Z_{v_{i+1}}\big)$ *inside the triangle* qv_iv_{i+1} *are identical.*

Proof. Let p be a point in $(R \cup I) \setminus \{q\}$ that contributes to $\mathrm{Vor}_Q(R \cup I)$ inside qv_iv_{i+1}. As $qv_iv_{i+1} \subseteq V_q(R)$, $p \notin R$. So $p \in I$. By Lemma 4, p conflicts with v_i or v_{i+1}. ◀

Step 2 of the operation phase has found $V_S|_{p_i}$ for each $p_i \in I$. $V_S|_{p_i}$ and the portions of the Voronoi edges of $\mathrm{Vor}_Q(S)$ among the points in $V_S|_{p_i}$ are preserved in $\mathrm{Vor}_Q(R)$ because R includes the subset of S whose Voronoi cells conflict with the input points. Hence, $\bigcup_{i=1}^n V_S|_{p_i}$ is the set U_R of Voronoi edge bends and Voronoi vertices in $\mathrm{Vor}_Q(R)$ that conflict with the input points. y Lemma 14, we locally compute pieces of $\mathrm{Vor}_Q(R \cup I)$ and stitch them together. The running time is $O\big(\sum_{u,v}(|Z_u| + |Z_v|)\log(|Z_u| + |Z_v|)\big)$, where the sum is over all pairs $\{u, v\}$ of adjacent Voronoi edge bends and Voronoi vertices in $\mathrm{Vor}_Q(R)$ such that $\{u, v\} \cap U_R \neq \emptyset$. Since the degrees of Voronoi edge bends and Voronoi vertices are two and three respectively, this running time can be bounded by $O\big(\sum_{v \in U_R} |Z_v| \log |Z_v|\big)$. Since $U_R \subseteq V_S$, by Lemma 2, step 6 of the operation phase computes $\mathrm{Vor}_Q(R \cup I)$ in $O(n)$ expected time.

In step 7, the splitting of $\mathrm{Vor}_Q(R \cup I)$ into $\mathrm{Vor}_Q(R)$ and $\mathrm{Vor}_Q(I)$ can be performed in $O(n)$ expected time by using the algorithm in [6] for splitting a Euclidean Delaunay triangulation. That algorithm is combinatorial in nature. It relies on the Voronoi diagram being planar and of $O(n)$ size, all points having $O(1)$ degrees in the nearest neighbor graph, and that one can delete a site from a Voronoi diagram in time proportional to its number of Voronoi neighbors. The first two properties hold in our case, and it is known how to delete a site from an abstract Voronoi diagram so that the expected running time is proportional to its number of Voronoi neighbors [17].

▶ **Lemma 15.** *Step 6 of the operation phase computes* $\mathrm{Vor}_Q(R \cup I)$ *in* $O(n)$ *expected time, and step 7 splits* $\mathrm{Vor}_Q(R \cup I)$ *into* $\mathrm{Vor}_Q(I)$ *and* $\mathrm{Vor}_Q(R)$ *in* $O(n)$ *expected time.*

In summary, since steps 2-7 of the operation phase take $O(n)$ expected time, the limiting complexity is dominated by the $O\big(\frac{1}{\varepsilon} n \log m + \frac{1}{\varepsilon} n 2^{O(\log^* n)} + \frac{1}{\varepsilon} H\big)$ expected running time of step 1. In the Euclidean case, step 1 runs faster in $O\big(\frac{1}{\varepsilon} n \log m + \frac{1}{\varepsilon} H\big)$ time.

▶ **Theorem 16.** *Let Q be a convex polygon with $O(1)$ complexity. Let n be the input size. For any $\varepsilon \in (0, 1)$ and any hidden mixture of at most $m = o(\sqrt{n})$ product distributions such that each distribution contributes an instance with a probability of $\Omega(1/n)$, there is a self-improving algorithm for constructing a Voronoi diagram under d_Q with a limiting complexity of $O(\frac{1}{\varepsilon} n \log m + \frac{1}{\varepsilon} n 2^{O(\log^* n)} + \frac{1}{\varepsilon} H)$. For the Euclidean metric, the limiting complexity is $O(\frac{1}{\varepsilon} n \log m + \frac{1}{\varepsilon} H)$. The training phase runs in $O(mn \log^2(mn) + m^\varepsilon n^{1+\varepsilon} \log(mn))$ time. The success probability is at least $1 - O(1/n)$.*

5 Conclusion

It is open whether one can get rid of the requirement that each distribution in the mixture contributes an instance with a probability of $\Omega(1/n)$, which is not needed for self-improving sorting [8]. Eliminating the $n 2^{O(\log^* n)}$ term from the limiting complexity might require solving the question raised in [5] that whether there is an $O(n)$-time algorithm for computing the lower envelope of pseudo-planes. As a Voronoi diagram can be interpreted as the lower envelope of some appropriate surfaces, a natural question is what surfaces admit a self-improving lower envelope algorithm.

References

1 N. Ailon, B. Chazelle, K. Clarkson, D. Liu, W. Mulzer, and C. Seshadhri. Self-improving algorithms. *SIAM Journal on Computing*, 40:350–375, 2011.

2 S. Arya, T. Malamatos, D.M. Mount, and K.C. Wong. Optimal expected-case planar point location. *SIAM Journal on Computing*, 37:584–610, 2007.

3 K. Buchin and W. Mulzer. Delaunay triangulations in $o(\mathrm{sort}(n))$ time and more. *Journal of the ACM*, 58:6:1–6:27, 2011.

4 P.B. Callahan and S.R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM*, 42:67–90, 1995.

5 T.M. Chan. A simpler linear-time algorithm for intersecting two convex polyhedra in three dimensions. *Discrete and Computational Geometry*, 56:860–865, 2016.

6 B. Chazelle, O. Devillers, F. Hurtado, M. Mora, V. Sacristan, and M. Teillaud. Splitting a delaunay triangulation in linear time. *Algorithmica*, 34:39–46, 2002.

7 S.-W. Cheng, M.-K. Chiu, K. Jin, and M.T. Wong. A generalization of self-improving algorithms. In *Proceedings of the International Symposium on Computational Geometry*, pages 29:1–29:13. Full version: arXiv:2003.08329v2 [cs.CG], 2020, 2020.

8 S.-W. Cheng, K. Jin, and L. Yan. Extensions of self-improving sorters. *Algorithmica*, 82:88–106, 2020.

9 S.-W. Cheng and M.T. Wong. Self-improving voronoi construction for a hidden mixture of product distributions. arXiv:2109.13460 [cs.CG], 2021.

10 L. Paul Chew and R.L. Scot Drysdale. Voronoi diagrams based on convex distance functions. In *Proceedings of the 1st Annual Symposium on Computational Geometry*, pages 235–244, 1985.

11 K.L. Clarkson, W. Mulzer, and C. Seshadhri. Self-improving algorithms for coordinatewise maxima and convex hulls. *SIAM Journal on Computing*, 43:617–653, 2014.

12 K.L. Clarkson and P.W. Shor. Applications of random sampling in computational geometry, II. *Discrete and Computational Geometry*, 4:387–421, 1989.

13 H. Edelsbrunner, L.J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15:317–340, 1986.

14 G.N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16:1004–1022, 1987.

15 S. Har-Peled and M. Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM Journal on Computing*, 35:1148–1184, 2006.

16 J. Iacono. Expected asymptotically optimal planar point location. *Computational Geometry: Theory and Applications*, 29:19–22, 2004.

17 K. Junginer and E. Papadopoulou. Deletion in abstract voronoi diagram in expected linear time. In *Proceedings of the 34th International Symposium on Computational Geometry*, pages 50:1–50:14, 2018.

18 R. Klein, K. Mehlhorn, and S. Meiser. Randomized incremental construction of abstract voronoi diagrams. *Computational Geometry: Theory and Applications*, 3:157–184, 1993.

19 E. Pyrga and S. Ray. New existence proofs for ϵ-nets. In *Proceedings of the 24th Annual Symposium on Computational Geometry*, pages 199–207, 2008.

20 A.K. Tsakalides and J. van Leeuwen. An optimal pointer machine algorithm for finding nearest common ancestors. Technical Report RUU-CS-88-17, Department of Computer Science, University of Utrecht, 1988.

21 P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10:99–127, 1977.

22 R.W. Yeung. *A First Course in Information Theory*. Kluwer Academic/Plenum Publishers, 2002.