


A Generalization of Self-Improving Algorithms

Siu-Wing Cheng 

HKUST, Hong Kong, China
scheng@cse.ust.hk

Man-Kwun Chiu

Institut für Informatik, Freie Universität Berlin, Germany
chiunk@zedat.fu-berlin.de

Kai Jin 

HKUST, Hong Kong, China
cscjkk@gmail.com

Man Ting Wong

HKUST, Hong Kong, China
mtwongaf@connect.ust.hk

Abstract

Ailon et al. [SICOMP'11] proposed self-improving algorithms for sorting and Delaunay triangulation (DT) when the input instances x_1, \dots, x_n follow some unknown *product distribution*. That is, x_i comes from a fixed unknown distribution \mathcal{D}_i , and the x_i 's are drawn independently. After spending $O(n^{1+\varepsilon})$ time in a learning phase, the subsequent expected running time is $O((n + H)/\varepsilon)$, where $H \in \{H_S, H_{DT}\}$, and H_S and H_{DT} are the entropies of the distributions of the sorting and DT output, respectively. In this paper, we allow dependence among the x_i 's under the *group product distribution*. There is a hidden partition of $[1, n]$ into groups; the x_i 's in the k -th group are fixed unknown functions of the same hidden variable u_k ; and the u_k 's are drawn from an unknown product distribution. We describe self-improving algorithms for sorting and DT under this model when the functions that map u_k to x_i 's are well-behaved. After an $O(\text{poly}(n))$ -time training phase, we achieve $O(n + H_S)$ and $O(n\alpha(n) + H_{DT})$ expected running times for sorting and DT, respectively, where $\alpha(\cdot)$ is the inverse Ackermann function.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases expected running time, entropy, sorting, Delaunay triangulation

Digital Object Identifier 10.4230/LIPIcs.SoCG.2020.29

Related Version A full version of this paper is available at <http://arxiv.org/abs/2003.08329>.

Funding *Siu-Wing Cheng*: Research of Cheng is supported by Research Grants Council, Hong Kong, China (project no. 16200317).

Man-Kwun Chiu: Research of Chiu is supported by ERC StG 757609.

Kai Jin: Research of Jin is supported by Research Grants Council, Hong Kong, China (project no. 16200317).

Man Ting Wong: Research of Wong is supported by Research Grants Council, Hong Kong, China (project no. 16200317).

1 Introduction

Ailon et al. [1] proposed *self-improving algorithms* for sorting and Delaunay triangulation (DT). The setting is that the input is drawn from an unknown but fixed distribution \mathcal{D} . The goal is to automatically compute some auxiliary structures in a *training phase*, so that these auxiliary structures allow an algorithm to achieve an expected running time better than the worst-case optimum in the subsequent *operation phase*. The expected running time in the operation phase is known as the *limiting complexity*.



© Siu-Wing Cheng, Man-Kwun Chiu, Kai Jin, and Man Ting Wong;
licensed under Creative Commons License CC-BY

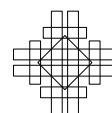
36th International Symposium on Computational Geometry (SoCG 2020).

Editors: Sergio Cabello and Danny Z. Chen; Article No. 29; pp. 29:1–29:13

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



This model is attractive for two reasons. First, it addresses the criticism that worst-case time complexity alone may not be relevant because worst-case input may occur rarely, if at all. Second, it is more general than some previous average-case analyses that deal with distributions that have simple, compact formulations such as the uniform, Poisson, and Gaussian distributions. There is still a constraint in the work of Ailon et al. [1], that is, \mathcal{D} must be a *product distribution*, meaning that the i -th input item follows a particular distribution and two distinct input items are independently drawn from their respective distributions.

Self-improving algorithms under product distributions have been proposed for sorting, DT, 2D coordinatewise maxima, and 2D convex hull. For sorting, Ailon et al. showed that a limiting complexity of $O((n + H_S)/\varepsilon)$ can be achieved for any $\varepsilon \in (0, 1)$, where H_S denotes the entropy of the output permutations. This limiting complexity is optimal in the comparison-based model by Shannon's theory [11]. The training phase uses $O(n^\varepsilon)$ input instances and runs in $O(n^{1+\varepsilon})$ time. The probability of achieving the stated limiting complexity is at least $1 - 1/n$. Ailon et al. [1] also proposed a self-improving algorithm for DT. The performance of the training phase is the same. The limiting complexity is $O((n + H_{DT})/\varepsilon)$, where H_{DT} denotes the entropy of the output Delaunay triangulations. Self-improving algorithms for 2D coordinatewise maxima and convex hulls have been developed by Clarkson et al. [9]. The limiting complexities for the 2D maxima and 2D convex hull problems are $O(\text{OptM} + n)$ and $O(\text{OptC} + n \log \log n)$, where OptM and OptC are the expected depths of optimal linear decision trees for the maxima and convex hull problems, respectively.

It is natural to allow dependence among input items. However, some restriction is necessary because Ailon et al. showed that $\Omega(2^{n \log n})$ bits of storage are necessary for optimally sorting n numbers if there is no restriction on the input distribution. In [8], two extensions are considered for sorting. The first extension assumes that there is a hidden partition of $[1, n]$ into groups G_k 's. The input items with indices in G_k are unknown linear functions of a common parameter u_k . The parameters u_1, u_2, \dots follow a product distribution. A limiting complexity of $O((n + H_S)/\varepsilon)$ can be achieved after a training phase that processes $O(n^\varepsilon)$ instances in $O(n^2 \log^3 n)$ time and $O(n^2)$ space. The second extension assumes that the input is a hidden mixture of product distributions, and that an upper bound m is given on the number of distributions in the mixture. A limiting complexity of $O((n \log m + H_S)/\varepsilon)$ can be achieved after a training phase that processes $O(mn \log(mn))$ instances in $O(mn \log^2(mn) + m^\varepsilon n^{1+\varepsilon} \log(mn))$ time and $O(mn \log(mn) + m^\varepsilon n^{1+\varepsilon})$ space.

In this paper, we revisit the problems of sorting and DT when there is a hidden partition of $[1, n]$ into G_1, G_2, \dots such that for each k , there is a hidden parameter u_k such that for all $i \in G_k$, $x_i = h_{i,k}(u_k)$ for some unknown function $h_{i,k}$. For sorting, u_k belongs to \mathbb{R} ; for DT, u_k belongs to \mathbb{R}^2 ; and u_1, u_2, \dots follow a product distribution. We call such an input distribution a *group product distribution*. The groups G_k 's are not given and they have to be learned in the training phase. Our generalization have the following features.

- For sorting, we do not assume any specific formulation of the functions $h_{i,k}$'s. Neither is any oracle given for evaluating them. We only assume that the graph of each $h_{i,k}$ has at most c_0 extrema, where c_0 is a known value, and the graphs of two distinct $h_{i,k}$ and $h_{j,k}$ intersect in $O(1)$ points. Our algorithm does not reconstruct or approximate the $h_{i,k}$'s.
- For DT, we assume that there are bivariate polynomials $h_{i,k}^x$ and $h_{i,k}^y$ in u_k for $i \in G_k$ that give the x - and y -coordinates of the i -th input point. The degrees of the $h_{i,k}^x$'s and $h_{i,k}^y$'s are no more than a fixed constant. No further information about these bivariate polynomials are given. Depending on the distribution of u_k , it may be impossible to reconstruct the equations of $h_{i,k}^x$ and $h_{i,k}^y$ using the input data.

Let $\alpha(\cdot)$ be the inverse Ackermann function. We prove that an optimal $O(n + H_S)$ limiting complexity for sorting and a nearly optimal $O(n\alpha(n) + H_{DT})$ limiting complexity for DT can be achieved with probability at least $1 - O(1/n)$ after a polynomial-time training phase. The training takes $\tilde{O}(c_0 n^3 + c_0^2 n^2)$ time for sorting and $\tilde{O}(n^{10})$ time for DT.

We use several new techniques to obtain our results. To learn the hidden partition for sorting, we need to test if two indices i and j are in the same group. By collecting x_i 's and x_j 's from some instances, we can reduce the test to finding the longest monotonic subsequence (LMS) among the points (x_i, x_j) 's. We establish a threshold such that i and j are in the same group with very high probability if and only if the LMS has length greater than or equal to the threshold. In the operation phase, there are $O(n)$ ordered intervals from left to right, and we need to sort the subset of an input instance I within an interval. Under the group product distribution, there can be a large subset $I' \subset I$ from the same group that reside in an interval, which does not happen in the case of product distribution. We do not have enough time to sort I' from scratch. Instead, we need to recognize that I' gives a result similar to what we have seen in the training phase. To this end, we must be able to “read off” the answer from some precomputed information in the training phase in order to beat the worst-case bound. We use a trie to compute and store *Lehmer codes* [18] in the training phase. This trie structure is essential for achieving the optimal limiting complexity.

The hidden partition for DT seems harder to learn given the 2D nature of the problem. We employ tools from algebraic geometry to do so, which is the reason for requiring the $h_{i,k}^x$'s and $h_{i,k}^y$'s to be bivariate polynomials of fixed degree. In the operation phase, we need to compute the Voronoi diagram of the subsets of I inside the triangles of a canonical Delaunay triangulation. Under the group product distribution, a large subset $I' \subset I$ may fall in to the same triangle t , which does not happen in the case of product distribution. The big hurdle is to decide what information to compute and store in the training phase so that we can “read off” the Voronoi diagram needed. We need a structure that is equivalent to a Delaunay triangulation or Voronoi diagram. Yet it should be “more combinatorial” in nature so that it is not as sensitive to geometric perturbations. The split tree of a point set fits this role nicely because the Delaunay triangulation can be computed from it in linear expected time [2]. We can now expand the trie structure used for sorting to record different split trees that are generated in the training phase to facilitate the DT computation in the operation phase.

2 Self-improving sorter

Let $u_k \in \mathbb{R}$ denote the parameter that governs the group G_k . Let $h_{i,k}$ denote the function that determines $x_i = h_{i,k}(u_k)$ for $i \in G_k$. We do not impose any particular formulation of the $h_{i,k}$'s as long as they satisfy the following properties:

- $\forall k \forall i$, the graph of $h_{i,k}$ has at most c_0 extrema for a known value c_0 .
- $\forall k \forall i \neq j$, the graphs of $h_{i,k}$ and $h_{j,k}$ intersect at $O(1)$ points.
- $\forall k \forall i \forall c \in \mathbb{R}$, $\Pr[h_{i,k}(u_k) = c] = 0$.

2.1 Hidden partition and V-list

We first learn the hidden partition of $[1, n]$. Given a sequence σ of real numbers, let $\text{LMS}(\sigma)$ be the length of the *longest monotone subsequence* of σ (either increasing or decreasing), and let $\text{LIS}(\sigma)$ be the length of the *longest increasing subsequence* of σ .

We describe how to test if the indices 1 and 2 belong to the same group. The other index pairs can be handled in the same way. Let $N = \max\{100^3, (90 \ln(4n^3))^2, (6c_0 + 3)^2\}$, where c_0 is the upper bound on the number of extrema of $h_{i,k}$. Take N instances. Let I_1, I_2, \dots, I_N

be these instances in increasing order of their first items. That is, $x_1^{(1)} < \dots < x_1^{(N)}$, where $x_1^{(i)}$ denotes the first item in I_i . Similarly, $x_2^{(i)}$ denotes the second item in I_i . Then, compute $\text{LIS}(x_2^{(1)}, \dots, x_2^{(N)})$ and $\text{LIS}(x_2^{(N)}, \dots, x_2^{(1)})$ in $O(N \log N)$ time. The larger of the two is $\text{LMS}(x_2^{(1)}, \dots, x_2^{(N)})$. If $\text{LMS}(x_2^{(1)}, \dots, x_2^{(N)}) \geq N/(2c_0 + 1)$, report that 1 and 2 are in the same group. Otherwise, report that 1 and 2 are in different groups.

We show that the above test works correctly with very high probability. The following result is obtained by applying the first theorem in [17] and the setting that $N \geq 100^3$.

► **Lemma 1** ([17]). *If σ is a permutation of $[1, N]$ drawn uniformly at random, then $\Pr[\text{LIS}(\sigma) \geq 3\sqrt{N}] \leq \exp(-\sqrt{N}/90) = O(1/n^3)$.*

We are ready to show the correctness of our test procedure.

► **Lemma 2.** *If the indices 1 and 2 belong to the same group, then $\text{LMS}(x_2^{(1)}, \dots, x_2^{(N)}) \geq N/(2c_0 + 1)$; otherwise, $\Pr[\text{LMS}(x_2^{(1)}, \dots, x_2^{(N)}) \geq N/(2c_0 + 1)] = O(1/n^3)$.*

Proof. Suppose that 1 and 2 belong to group G_k . Then, $x_1 = h_{1,k}(u_k)$ and $x_2 = h_{2,k}(u_k)$. Let t_1, \dots, t_m , where $m \leq 2c_0$, be the values of u_k at the extrema of $h_{1,k}$ and $h_{2,k}$. Let $t_0 = -\infty$ and let $t_{m+1} = +\infty$. By the pigeonhole principle, there exists $j \in [0, m]$ such that $|\{x_1^{(1)}, \dots, x_1^{(N)}\} \cap [t_j, t_{j+1}]| \geq N/(m+1) \geq N/(2c_0 + 1)$, and both $h_{1,k}$ and $h_{2,k}$ are monotonic in $[t_j, t_{j+1}]$. It follows that $\text{LMS}(x_2^{(1)}, \dots, x_2^{(N)}) \geq N/(2c_0 + 1)$.

Suppose that 1 and 2 belong to different groups. The distribution of $x_2^{(1)}, \dots, x_2^{(N)}$ is the same as the uniform distribution of the permutations of $[1, N]$. Lemma 1 implies that

$$\Pr[\text{LIS}(x_2^{(1)}, \dots, x_2^{(N)}) \geq 3\sqrt{N}] \leq O(1/n^3).$$

Symmetrically,

$$\Pr[\text{LIS}(x_2^{(N)}, \dots, x_2^{(1)}) \geq 3\sqrt{N}] \leq O(1/n^3).$$

As $\text{LMS}(x_2^{(1)}, \dots, x_2^{(N)}) = \max\{\text{LIS}(x_2^{(1)}, \dots, x_2^{(N)}), \text{LIS}(x_2^{(N)}, \dots, x_2^{(1)})\}$, by the union bound, we get $\Pr[\text{LMS}(x_2^{(1)}, \dots, x_2^{(N)}) \geq 3\sqrt{N}] \leq O(1/n^3)$. Since $N \geq (6c_0 + 3)^2$, we have $3\sqrt{N} \leq N/(2c_0 + 1)$. Hence, $\Pr[\text{LMS}(x_2^{(1)}, \dots, x_2^{(N)}) \geq N/(2c_0 + 1)] = O(1/n^3)$. ◀

There are $O(n^2)$ index pairs to check, each taking $O(N \log N)$ time. We conclude that:

► **Corollary 3.** *The partition of $[1, n]$ into groups can be learned in $\tilde{O}(c_0^2 n^2)$ time using $\tilde{O}(c_0^2 n^2)$ instances. The probability of success is at least $1 - O(1/n)$.*

Following [1], we define a V -list, $(v_0, v_1, \dots, v_{n+1})$, in the training phase as follows. Take $\lambda = n^2 \ln n$ instances. Let $y_1 < \dots < y_{\lambda n}$ be these numbers sorted in increasing order. Define $v_0 = -\infty$, $v_r = y_{\lambda r}$ for all $r \in [1, n]$, and $v_{n+1} = +\infty$. We take $[v_0, v_1]$ to be $(-\infty, v_1)$.

► **Lemma 4.** *It holds with probability at least $1 - 1/n^{192}$ that for every $r \in [0, n]$,*

$$\mathbb{E}_{I \sim \mathcal{D}}[|I \cap [v_r, v_{r+1}]|] = O(1).$$

Proof. Recall that $y_1 < \dots < y_{\lambda n}$ is the sorted list of the λ instances used to define the V -list. Let $y_0 = -\infty$ and let $y_{\lambda n+1} = \infty$. Fix a pair (i, j) for some distinct $i, j \in [0, \lambda n + 1]$ such that $y_i < y_j$. Let m_{ij} be the number of instances among the λ instances that contain neither y_i nor y_j . Denote them by $I_1, \dots, I_{m_{ij}}$. Note that $m_{ij} = \lambda - 2$ or $\lambda - 1$.

For $a \in [1, m_{ij}]$, define the random variable $Y_a^{(i,j)} = |I_a \cap [y_i, y_j]|$. Define $Y^{(i,j)} = Y_1^{(i,j)} + \dots + Y_{m_{ij}}^{(i,j)}$. We call (i, j) a *good pair* if $\mathbb{E}[Y^{(i,j)}] \leq 11\lambda$ or $Y^{(i,j)} > \lambda$. We prove in the following that (i, j) is a good pair with high probability.

For $a \in [1, m_{ij}]$, let $X_a = \frac{1}{n}Y_a^{(i,j)}$. Let $X = X_1 + \dots + X_{m_{ij}} = \frac{1}{n}Y^{(i,j)}$. Note that $X_1, \dots, X_{m_{ij}}$ are independent and each lies in the range $[0, 1]$. By Hoeffding's inequality [15], $\Pr[|X - \mathbb{E}[X]| \geq \beta m_{ij}] < 2e^{-2m_{ij}\beta^2}$ for any $\beta > 0$. Setting $\beta = 10\mathbb{E}[X]/(11m_{ij})$ gives $\Pr[|X - \mathbb{E}[X]| \geq 10\mathbb{E}[X]/11] < 2e^{-2m_{ij}(10\mathbb{E}[X])^2/(11m_{ij})^2}$. Thus, $\Pr[X < \mathbb{E}[X]/11] < 2e^{-200\mathbb{E}[X]^2/(121m_{ij})}$. When $\mathbb{E}[X] > 11\lambda/n$, we have $\Pr[X < \lambda/n] \leq \Pr[X < \mathbb{E}[X]/11] < 2e^{-200 \times 121\lambda^2/(121n^2m_{ij})} = 2e^{-200\lambda^2/(n^2m_{ij})} < 2e^{-200\lambda/n^2} = 2n^{-200}$. In other words, it holds that $\Pr[Y^{(i,j)} < \lambda] < 2n^{-200}$ when $\mathbb{E}[Y^{(i,j)}] > 11\lambda$. Hence, for a fixed pair (i, j) , it holds with probability at least $1 - n^{-199}$ that (i, j) is a good pair.

There are at most $(\lambda n + 2)^2 < n^7$ pairs of distinct indices. By the union bound, it holds with probability at least $1 - n^{-192}$ that all pairs of distinct indices from $[0, \lambda n + 1]$ are good.

Assume that all distinct pairs of indices from $[0, \lambda n + 1]$ are indeed good. Consider two consecutive elements v_r and v_{r+1} in the V -list. They correspond to y_i and y_j , respectively, for some distinct $i, j \in [0, \lambda n + 1]$. By the construction of the V -list, the range (y_i, y_j) contains fewer than λ points among $y_1 < \dots < y_{\lambda n}$. There are m_{ij} instances used in defining V that contain neither y_i and y_j . Therefore, fewer than λ points from these m_{ij} instances fall in $[y_i, y_j]$, that is, $Y^{(i,j)} < \lambda$. Then, $\mathbb{E}[Y^{(i,j)}] \leq 11\lambda$ because (i, j) is a good pair by assumption. Hence, $\mathbb{E}[|I \cap [v_r, v_{r+1}]|] \leq \mathbb{E}[Y^{(i,j)}]/m_{ij} \leq \mathbb{E}[Y^{(i,j)}]/(\lambda - 2) = O(1)$. ◀

Lemma 4 is more general than its counterparts in [1, 8] in that it does not make any assumption on the distribution that generates the instances. The price to pay is that $O(n^2 \log n)$ instances are needed instead of $O(\log n)$ in [1, 8], but this cost will be dominated by the trie construction cost in the training phase to be discussed in the next section.

2.2 Trie

In the operation phase, we distribute the numbers in an instance I to the intervals $[v_r, v_{r+1})$'s, sort numbers in each interval, and concatenate the results. We need an efficient method to distribute the numbers. In [8], the functions $h_{i,k}$'s are linear, so we can reformulate each $h_{i,k}$ as a linear function in another input number, say x_1 . This gives an arrangement of lines for each G_k (including the horizontal lines at each v_r). Cutting vertically through each arrangement vertex gives a sorted order of the x_i 's and the v_r 's within a range on x_1 .

We cannot compute such arrangements here because there is no assumption on the formulations of the $h_{i,k}$'s. We use a different method. Define:

$$b_k : \mathbb{R}^{|G_k|} \rightarrow [0, n]^{|G_k|} \text{ such that for all } |G_k|\text{-tuple of numbers } (z_1, \dots, z_{|G_k|}), \text{ we have } b_k(z_1, \dots, z_{|G_k|}) = (r_1, \dots, r_{|G_k|}) \text{ such that for all } m \in [1, |G_k|], z_m \in [v_{r_m}, v_{r_{m+1}}).$$

The output of b_k tells us to which interval a number in I with index from G_k belongs. In order to distribute these numbers quickly, we need another function defined as follows:

$$\pi_k : \mathbb{R}^{|G_k|} \rightarrow [0, n]^{|G_k|} \text{ such that for all } |G_k|\text{-tuple of numbers } (z_1, \dots, z_{|G_k|}), \text{ we have } \pi_k(z_1, \dots, z_{|G_k|}) = (j_1, \dots, j_{|G_k|}) \text{ such that for all } m \in [1, |G_k|], j_m = 0 \text{ if } z_m = \min_{a \in [1, m]} z_a; \text{ otherwise, } j_m \text{ is the index of the largest element in } \{z_1, \dots, z_{m-1}\} \text{ that is less than } z_m.$$

The output of π_k is the Lehmer code of $z_1, \dots, z_{|G_k|}$ [18]. For our purposes, $(z_1, \dots, z_{|G_k|})$ is given as a doubly linked list L , and the output of π_k is a list of pointers to entries in L . Given $\pi_k(z_1, \dots, z_{|G_k|})$, it is easy to sort $z_1, \dots, z_{|G_k|}$ in increasing order in $O(|G_k|)$ time.

Let $I|_{G_k}$ denote the subsequence of I with indices from G_k . For any set $S \subseteq \mathbb{R}^{|G_k|}$, let $b_k(S) = \{b_k(x) : x \in S\}$ and let $\pi_k(S) = \{\pi_k(x) : x \in S\}$.

► **Lemma 5.** *Let $S = \{I|_{G_k} : I \sim \mathcal{D}\}$. Then, $|b_k(S)| = O(c_0 n |G_k|)$ and $|\pi_k(S)| = O(|G_k|^2)$.*

Proof. Assume that $G_k = [1, m]$. The graph of each $h_{i,k}(u_k)$ is a curve in \mathbb{R}^2 . By assumption, there are $O(m^2)$ intersections among the $h_{i,k}$'s for $i \in [1, m]$. The vertical lines through these intersections divide \mathbb{R}^2 into $O(m^2)$ slabs. Clearly, $\pi_k(x_1, \dots, x_m)$ is invariant when u_k is restricted to any one of these slabs. So there are $O(m^2)$ possible outcomes for $\pi_k(x_1, \dots, x_m)$.

Add horizontal lines $y = v_r$ for each $r \in [1, n]$. There are at most $(c_0 + 1)nm$ intersections between these horizontal lines and the graphs of $h_{i,k}$'s mentioned above. The vertical lines through the intersections in the overlay of the graphs of $h_{i,k}$'s and these horizontal lines define $O(c_0 nm)$ slabs. Clearly, $b_k(x_1, \dots, x_m)$ is invariant when u_k is restricted to any one of these slabs. So there are $O(c_0 nm)$ possible outcomes for $b_k(x_1, \dots, x_m)$. ◀

We prove the main tool for achieving our results on sorting.

► **Theorem 6.** *Let $S = \{I|_{G_k} : I \sim \mathcal{D}\}$. Let $f \in \{b_k, \pi_k\}$. Let $n_0 = \max\{n, |f(S)|\}$. Using $n_0 \ln n_0 (\ln n + \ln c_0)$ instances in the training phase, with probability at least $1 - n_0^{-2}$, we can compute a data structure such that given $I \sim \mathcal{D}$, it returns $f(I|_{G_k})$ in $O(|G_k| + H_f)$ expected time, where H_f is the entropy of $f(S)$. The data structure uses $O(n_0 |G_k|)$ space, and it can be constructed in $\tilde{O}(n_0 n)$ time.*

Proof. Let $N = n_0 \ln n_0 (\ln n + \ln c_0)$. Take instances I_1, \dots, I_N in the training phase. Let $\{\beta_1, \dots, \beta_M\}$ be the set of distinct outcomes among $f(I_1|_{G_k}), \dots, f(I_N|_{G_k})$. Let $\tilde{\rho}_i$ be the frequency of β_i divided by N .

Store the β_j 's in a trie T . There is one leaf in T for each β_j . Each edge in T has a label from $[0, n]$ so that for $j \in [1, M]$, β_j is equal to the string of symbols on the path from the root to the leaf for β_j .

Given an instance I in the operation phase, we show how to return $f(I|_{G_k})$. For simplicity, let $I|_{G_k} = (x_1, \dots, x_{|G_k|})$. Let x_0 be a dummy symbol to the left of x_1 . Then, repeat the following starting at the root of T : when we are at x_{i-1} and a node u , find the child w of u such that the label on uw is consistent with x_i , and then move to x_i and w . The existence of a particular child w of u depends on whether there is an input instance in the training phase that prompts the creation of w . If we reach a leaf, we return the corresponding β_j . If we cannot find an appropriate child to proceed at any point, we abort and compute $f(I|_{G_k})$ in $O(|G_k| \log n)$ time using plain algorithms. We discuss how to find the correct child quickly.

If $f = b_k$, we seek an edge label r such that $x_i \in [v_r, v_{r+1})$. We use a nearly optimal binary search tree A_u to store the labels of edges to the children of u [19], which can be constructed in linear time [13]. The weight of w in A_u is the node weight that we assign to all nodes of T recursively as follows. The weight of a leaf of T for β_i is $\tilde{\rho}_i$. The weight of an internal node of T is the sum of the weights of its children. The search time of A_u is $O(\log(\text{weight}(u)/\text{weight}(w)))$. To build A_u , in the training phase, we grow a balanced binary search tree L_u from being initially empty to the final set of elements in A_u . That is, as new elements of A_u are discovered, they are inserted into L_u in $O(\log n)$ each. Also, L_u provides access to children of u in the trie in $O(\log n)$ time in the training phase. At the end of the training phase, we build A_u as a nearly optimal binary search tree of the elements in L_u . So the construction of T for b_k takes $O(n_0 n \log n)$ time.

Consider the case of $f = \pi_k$. At the node u , inductively, we know $\gamma : [1, i-1] \rightarrow [1, i-1]$ such that $x_{\gamma(a)}$ is the a -th smallest number among x_1, \dots, x_{i-1} . We saw the same permutation γ at u in the training phase. Thus, we organize a search tree A_u with i leaves corresponding to

the i intervals $\omega_0, \omega_1, \dots, \omega_{i-1}$, where ω_0 denotes the interval between $-\infty$ and the smallest number, ω_a denotes the interval between the a -th and $(a+1)$ -th smallest numbers for $a \in [1, i-2]$, and ω_{i-1} denotes the interval between the largest number and ∞ . We store $\gamma(a)$ and a pointer to $x_{\gamma(a)}$ at the internal node in A_u that separates the a -th smallest number from the $(a+1)$ -th smallest number, so that we can decide in $O(1)$ time whether $x_i < x_{\gamma(a)}$ or $x_i > x_{\gamma(a)}$. The search of A_u terminates at a leaf representing the interval between $x_{\gamma(c)}$ and $x_{\gamma(c+1)}$ for some c . Thus, $\gamma(c)$ is the index of the largest element in $\{x_1, \dots, x_{i-1}\}$ that is less than x_i . That leaf of A_u also stores a pointer to the corresponding child of u .

We store each map γ as a persistent search tree [12] to facilitate an efficient construction in the training phase. The a -th node in the symmetric order stores $\gamma(a)$. When we create a child w of u in the trie and label the trie edge uw by $\gamma(c)$, we need to extend γ to a map $\gamma' : [1, i] \rightarrow [1, i]$ at w such that $\gamma'(a) = \gamma(a)$ for $a \in [1, c]$, $\gamma'(c+1) = i$, and $\gamma'(a+1) = \gamma(a)$ for $a \in [c+1, i-1]$. This can be done by a persistent insertion of a new node between the c -th and $(c+1)$ -th nodes in γ . The new version of γ produced is γ' . This takes $O(\log n)$ amortized time and $O(1)$ amortized space.

We store A_u as a nearly optimal binary search tree as in the case of $f = b_k$. In the operation phase, accessing a child w in A_u takes $O(\log(\text{weight}(u)/\text{weight}(w)))$ time.

As a result, querying T takes $O(|G_k| + \log(1/\tilde{\rho}_i))$ time if the search terminates at the leaf of T for β_i . Although the true probability of this event is not $\tilde{\rho}_i$, the entropy of H_f is approximated well by $\sum_i \tilde{\rho}_i \log(1/\tilde{\rho}_i)$, giving an expected time of $O(|G_k| + H_f)$. If we are stuck at some internal node of T , the query time is $O(|G_k| \log n)$ due to the total access time of the A_u 's before getting stuck and computing $f(I|_{G_k})$ (by plain algorithms) afterwards. Nevertheless, the probability of this event is very low thanks to the training phase. Hence, the overall expected query time is $O(|G_k| + H_f)$. The details are given in the full version [7]. ◀

Fredman [14] obtained a special case of Theorem 6 when $f = \pi_k$, $\pi_k(S)$ is given, and every outcome in $\pi_k(S)$ is equally likely.

2.3 Operation phase

1. For $r \in [0, n]$, initialize $Z_r := \emptyset$.
2. Repeat the following steps for each group G_k .
 - a. Let $I|_{G_k} = (x_{i_1}, \dots, x_{i_{|G_k|}})$. Compute $\pi_k(x_{i_1}, \dots, x_{i_{|G_k|}})$ and $b_k(x_{i_1}, \dots, x_{i_{|G_k|}})$ using Theorem 6.
 - b. Use $\pi_k(x_{i_1}, \dots, x_{i_{|G_k|}})$ to sort $(x_{i_1}, \dots, x_{i_{|G_k|}})$. Let $x_{s_1} < \dots < x_{s_{|G_k|}}$ denote the sorting output. Using $b_k(x_{i_1}, \dots, x_{i_{|G_k|}})$, for all $j \in [1, |G_k|]$, we can read off the interval $[v_{r_j}, v_{r_j+1})$ to which x_{s_j} belongs.
 - c. By a left-to-right scan, break $(x_{s_1}, \dots, x_{s_{|G_k|}})$ at the boundaries of the intervals $[v_r, v_{r+1})$'s into contiguous subsequences. Insert each contiguous subsequence σ as a new element into Z_r , where $[v_r, v_{r+1})$ is the interval that contains the numbers in σ .
3. For each $r \in [0, n]$, merge the subsequences in Z_r into one sorted list.
4. Concatenate the sorted lists of step 3 in the left-to-right order of the intervals. Return the output.

► **Theorem 7.** *Under the group product distribution setting, there is a self-improving sorter with a limiting complexity of $O(n + H_S)$. The storage needed by the operation phase is $O(c_0 n^3)$. The training phase processes $\tilde{O}(c_0^2 n^2)$ instances in $\tilde{O}(c_0 n^3 + c_0^2 n^2)$ time using $\tilde{O}(c_0 n^3 + c_0^2 n^2)$ space. The success probability is at least $1 - O(1/n)$.*

Proof. Correctness is obvious. The training complexities and space used in the operation phase follow from Corollary 3, Lemma 5, and Theorem 6. By Theorem 6, step 2(a) takes $O(n + \sum_k H_{b_k} + \sum_k H_{\pi_k})$ time. By the result in [1, Lemma 2.3], $\sum_k H_{b_k} = O(n + H_S)$ because one can merge the sorted order of I with the V -list to obtain the outputs of all b_k 's in $O(n)$ time. As there are $O(|G_k|^2)$ different outputs of π_k by Lemma 5, $H_{\pi_k} = O(\ln |G_k|) = O(|G_k|)$ and so $\sum_k H_{\pi_k} = O(n)$. Step 3 runs in $O(\sum_r \sum_k |\sigma_{k,r}| \log |Z_r|) = O(\sum_r \sum_k |Z_r| |\sigma_{k,r}|)$ time, where $\sigma_{k,r} = I|_{G_k \cap [v_r, v_{r+1})}$. Let $y_{k,r}$ be number of groups other than G_k that have elements in Z_r . Then, $|Z_r| \leq y_{k,r} + 1$. So $\mathbb{E}[|Z_r| |\sigma_{k,r}|] \leq \mathbb{E}[(y_{k,r} + 1) |\sigma_{k,r}|] = \mathbb{E}[|\sigma_{k,r}|] + \mathbb{E}[y_{k,r} |\sigma_{k,r}|] = \mathbb{E}[|\sigma_{k,r}|] + \mathbb{E}[y_{k,r}] \mathbb{E}[|\sigma_{k,r}|] = (1 + \mathbb{E}[y_{k,r}]) \mathbb{E}[|\sigma_{k,r}|]$, which is $O(\mathbb{E}[|\sigma_{k,r}|])$ as Lemma 4 implies that $\mathbb{E}[y_{k,r}] = O(1)$. Finally, $\sum_k \sum_r \mathbb{E}[|\sigma_{k,r}|] = O(n)$. ◀

3 Self-improving Delaunay triangulator

An input instance I consists of n points, (p_1, \dots, p_n) , where $p_i = (p_{i,x}, p_{i,y})$. We assume that there is a hidden partition of $[1, n]$ into disjoint groups G_1, G_2, \dots . For all $k \geq 1$, G_k is governed by a random variable $u_k \in \mathbb{R}^2$. That is, there exist $h_{i,k}^x, h_{i,k}^y : \mathbb{R}^2 \rightarrow \mathbb{R}$ such that $p_{i,x} = h_{i,k}^x(u_k)$ and $p_{i,y} = h_{i,k}^y(u_k)$. The functions $h_{i,k}^x$ and $h_{i,k}^y$ are bivariate polynomials with degree at most some known constant d_0 . We assume that the following properties hold.

- For all non-zero bivariate polynomial f of degree at most $d_0 d_1$, $\Pr[f(u_k) = 0] = 0$, where $d_1 = 2(d_0^2/2 + d_0)^{16}$.
- $\forall i \forall k \forall c \in \mathbb{R}$, both $\Pr[h_{i,k}^x(u_k) = c]$ and $\Pr[h_{i,k}^y(u_k) = c]$ are zero.

We show in the full version [7] that G_1, G_2, \dots can be learned in $O(n^2)$ time almost surely using $O(n^2)$ instances.

3.1 Auxiliary structures

Take a set S of $\lambda = n^2 \ln n$ instances. Take a $(1/n)$ -net V' of S with respect to disks, that is, for any disk C , $|C \cap S| \geq |S|/n \Rightarrow C \cap V' \neq \emptyset$. It is known that $|V'| = O(n)$ [10]. Add to V' three special points that form a huge triangle τ such that any input point lies inside τ . Let V be the union of V' and these three special points. The canonical Delaunay triangulation $\text{Del}(V)$ satisfies the following property with a proof analogous to that of Lemma 4.

► **Lemma 8.** *It holds with probability at least $1 - 1/n^{189}$ that for every triangle $t \in \text{Del}(V)$, $\mathbb{E}_{I \sim \mathcal{D}}[|I \cap C_t|] = O(1)$, where C_t is the circumscribing disk of t .*

We will need the following function which is the counterpart of b_k for self-improving sorters. Let $|\text{Del}(V)|$ denote the number of triangles in $\text{Del}(V)$. Arbitrarily assign indices from 1 to $|\text{Del}(V)|$ to the triangles in $\text{Del}(V)$; the triangle with index r is denoted by t_r .

$$B_k : \mathbb{R}^{2|G_k|} \rightarrow [1, |\text{Del}(V)|]^{|G_k|} \text{ such that for all } |G_k|\text{-tuple of points } (q_1, \dots, q_{|G_k|}), \\ B_k(q_1, \dots, q_{|G_k|}) = (r_1, \dots, r_{|G_k|}) \text{ such that } q_i \text{ lies in the triangle } t_{r_i} \in \text{Del}(V).$$

We use a variant of the *fair split tree* in [3]. Let \hat{R} and R be the smallest axis-aligned bounding square and the smallest axis-aligned bounding rectangle, respectively, of the given input instance I . We initialize the split tree to be a single node u with $R(u) = R$ and $\hat{R}(u) = \hat{R}$. In general, for any internal node w , $R(w)$ is the smallest axis-aligned rectangle of the subset of points represented by w , and $\hat{R}(w)$ is an outer rectangle that encloses $R(w)$. To split w , take the bisecting line of $R(w)$ that is perpendicular to a longest side of $R(w)$. This line splits the point set at w into two non-empty subsets, and it also splits $\hat{R}(w)$ into the outer rectangles of the children of w . The expansion bottoms out at nodes that represent only one point in I . This gives a split tree I which is a full binary tree with n leaves. Since we bisect $R(w)$ at each internal node w , we call the output split tree a *halving split tree*.

For every rectangle r , let $\ell_{\min}(r)$ and $\ell_{\max}(r)$ be the minimum and maximum side lengths of r , respectively. The following property is satisfied [3, equation (1) in Lemma 4.1]:

$$\text{For each node } u \text{ of the split tree, } \ell_{\min}(\hat{R}(u)) \geq \frac{1}{3}\ell_{\max}(R(\text{parent}(u))). \tag{1}$$

There are non-halving split trees that also satisfy (1), which are called fair split trees in [3]. We use $\text{Split}T(I)$ to denote a fair split tree of I . Using (1), one can show that a fair split tree can be used to produce a well-separated pair decomposition of $O(n)$ size, which can then be used to produce a Delaunay triangulation in $O(n)$ expected time (see Lemma 12 below).

We will require the following function which is the counterpart of π_k for self-improving sorters. Let HST denote the set of halving split trees of all possible $|G_k|$ points in \mathbb{R}^2 .

$$\begin{aligned} \Pi_k : \mathbb{R}^{2|G_k|} &\rightarrow [0, n]^{|G_k|} \times [0, n]^{|G_k|} \times HST \text{ such that for all } |G_k|\text{-tuple of points } \\ \mathbf{q}, \Pi_k(\mathbf{q}) &= (\pi_k(\mathbf{q}_x), \pi_k(\mathbf{q}_y), \text{the halving split tree of } \mathbf{q}), \text{ where } \mathbf{q} = (q_1, \dots, q_{|G_k|}), \\ \mathbf{q}_x &= (q_{1,x}, \dots, q_{|G_k|,x}), \text{ and } \mathbf{q}_y = (q_{1,y}, \dots, q_{|G_k|,y}). \end{aligned}$$

We call the first output of Π_k the x -order and the second output the y -order.

► **Lemma 9.** *Given $S = \{I_{G_k} : I \sim \mathcal{D}\}$, $|B_k(S)| = O(n^2|G_k|^2)$ and $|\Pi_k(S)| = O(|G_k|^8)$.*

Proof. Assume that $G_k = [1, m]$. Recall that the functions $h_{i,k}^x$'s and $h_{i,k}^y$'s have degrees at most d_0 . Consider the following equations for some possibly non-distinct indices $i_1, i_2, i_3, i_4 \in [1, m]$: $h_{i_1,k}^x = h_{i_2,k}^x$, $h_{i_1,k}^y = h_{i_2,k}^y$, $h_{i_1,k}^x - h_{i_2,k}^x = h_{i_3,k}^y - h_{i_4,k}^y$, $h_{i_1,k}^x + h_{i_2,k}^x = 2h_{i_3,k}^x$, and $h_{i_1,k}^y + h_{i_2,k}^y = 2h_{i_3,k}^y$. Each of these equations is an algebraic curve in \mathbb{R}^2 of degree at most d_0 , so the arrangement \mathcal{A} of these $O(m^4)$ curves has complexity $O(m^8)$. We argue that there are no more combinatorially different halving split trees than the cells in \mathcal{A} .

Take the coarser arrangement \mathcal{A}_0 formed by the curves $h_{i_1,k}^x = h_{i_2,k}^x$ and $h_{i_1,k}^y = h_{i_2,k}^y$ for distinct indices $i_1, i_2 \in [1, m]$. Each cell of \mathcal{A}_0 gives a distinct combination of x -order and y -order, so each cell may lead to a different split tree. Take a cell C of \mathcal{A}_0 . Suppose that u is the split tree root, $\hat{R}(u)$ has p_{i_1} and p_{i_2} on its vertical sides, and we split $\hat{R}(u)$ vertically. The curves $\{h_{i_1,k}^x + h_{i_2,k}^x = 2h_{i_3,k}^x : i_3 \in [1, m] \setminus \{i_1, i_2\}\}$ divide C into interior-disjoint regions. Each region corresponds to a distinct partition of points into the children w_1 and w_2 of u . Take one such region C' . Suppose that the smallest bounding rectangle of points in w_1 have $p_{i'_1}$ and $p_{i'_2}$ on its vertical sides and $p_{i'_3}$, and $p_{i'_4}$ on its horizontal sides. The sign of $(h_{i'_1,k}^x - h_{i'_2,k}^x) - (h_{i'_3,k}^y - h_{i'_4,k}^y)$ determines whether $\hat{R}(w_1)$ is split vertically or horizontally. Similarly, the sign of $(h_{i'_5,k}^x - h_{i'_6,k}^x) - (h_{i'_7,k}^y - h_{i'_8,k}^y)$ for some i'_5, i'_6, i'_7, i'_8 tells us if $\hat{R}(w_2)$ is split vertically or horizontally. So the two curves $(h_{i'_1,k}^x - h_{i'_2,k}^x) = (h_{i'_3,k}^y - h_{i'_4,k}^y)$ and $(h_{i'_5,k}^x - h_{i'_6,k}^x) = (h_{i'_7,k}^y - h_{i'_8,k}^y)$ divide C' to subregions such that each subregion corresponds to a combinatorial distinct splitting of w_1 and w_2 . Continuing with the above argument shows that there are no more halving split trees than the cells in \mathcal{A} . So $|\Pi_k(S)| = O(m^8)$.

Let \mathcal{L} be the set of support lines of all edges in $\text{Del}(V)$. Let the equations of these lines be $\alpha_j x + \beta_j y + \gamma_j = 0$ for $j \in [1, |\mathcal{L}|]$. Consider the following algebraic curves in \mathbb{R}^2 .

$$\alpha_j h_{i,k}^x(u_k) + \beta_j h_{i,k}^y(u_k) + \gamma_j = 0 \quad \text{for } i \in [1, m] \text{ and } j \in [1, |\mathcal{L}|].$$

Let \mathcal{A}' be the arrangement of these $O(nm)$ curves. Since these curves have degrees no more than d_0 , the complexity of \mathcal{A}' is $O(n^2 m^2)$. Inside any cell of \mathcal{A}' , the signs of

$$\alpha_j h_{i,k}^x(u_k) + \beta_j h_{i,k}^y(u_k) + \gamma_j \quad \text{for } i \in [1, m] \text{ and } j \in [1, |\mathcal{L}|]$$

are invariant. These signs determine $B_k(p_1, \dots, p_m)$. Hence, $|B_k(S)| = O(n^2 m^2)$. ◀

We can generalize Theorem 6 to work for B_k and Π_k .

► **Theorem 10.** *Theorem 6 holds for $f \in \{B_k, \Pi_k\}$.*

Proof. The input to B_k and Π_k is the tuple $\mathbf{q} = I|_{G_k}$. W.l.o.g., let $\mathbf{q} = (q_1, \dots, q_{|G_k|})$.

For B_k , we build a trie T like the case of $f = b_k$ in the proof of Theorem 6. If we are at q_{i-1} and a node u of T , we seek an edge uw for some child w of u such that q_i lies in $t_r \in \text{Del}(V)$, where r is the label of uw . Therefore, we build a distribution-sensitive planar point location structure for the subset of triangles of $\text{Del}(V)$ represented by the labels of edges from u to its children [16]. For an edge uw with label r , the weight of t_r in the point location structure is equal to the weight of w in T , which is defined recursively as in the proof of Theorem 6. So finding w takes $O(\log(\text{weight}(u)/\text{weight}(w)))$ time [16] as before.

Consider Π_k . The top $|G_k|$ levels of T is a trie T_1 for determining $\pi_k(q_{1,x}, \dots, q_{|G_k|,x})$. So T_1 is a copy of the trie for π_k in the proof of Theorem 6. We expand each leaf u of T_1 into a trie $T_{2,u}$ for determining $\pi_k(q_{1,y}, \dots, q_{|G_k|,y})$. Let T_2 be the composition of T_1 and all $T_{2,u}$'s. Given an instance I , we know the x -order and y -order of I at a leaf in T_2 .

We expand each leaf v of T_2 to a trie $T_{3,v}$ as follows. Let $P_v = I$. For $i \in [1, n-1]$, v may have a child corresponding to a vertical cut between the i -th and $(i+1)$ -th points in the x -order of P_v . Similarly, for $i \in [1, n-1]$, v may have a child corresponding to a horizontal cut between the i -th and $(i+1)$ -th points in the y -order of P_v . So v has at most $2n-2$ children. The existence of a particular child w of v depends on whether some input instance in the training phase prompts the creation of w . Each child w of v represents a subset $P_w \subset P_v$. We recursively expand the children of v , and the recursion bottoms out when we reach a node that represents only a single point. The trie $T_{3,v}$ models the sequence of possible cuts deployed in the training phase in constructing a split tree for a point set that has the same x -order and y -order represented by v . Hence, each leaf of $T_{3,v}$ represents a split tree. Let T_3 be the composition of T_2 and all $T_{3,v}$'s.

We need a fast way to locate a child in T_3 . We build a nearly optimal binary search trees at each internal node of T_1 and all $T_{2,u}$'s as in the case of π_k in the proof of Theorem 6. At each internal node u of $T_{3,v}$, we keep two nearly optimal binary search trees $A_{u,0}$ and $A_{u,1}$ organized as follows. At the node u , inductively, we know two functions $\gamma_x : [1, |P_u|] \rightarrow [1, n]$ and $\gamma_y : [1, |P_u|] \rightarrow [1, n]$ such that $q_{\gamma_x(a)}$ and $q_{\gamma_y(a)}$ has the a -th smallest x - and y -coordinates in P_u , respectively. We discovered the same functions γ_x and γ_y at u in the training phase. We organize a nearly optimal binary search tree $A_{u,0}$ with $|P_u|+1$ leaves corresponding to the $|P_u|+1$ gaps among $-\infty$, the x -coordinates of points in P_u in increasing order, and ∞ . We store $\gamma_x(a)$ at the internal node in $A_{u,0}$ that separates the a -th and the $(a+1)$ -th smallest x -coordinates. By comparing $q_{\gamma_x(|P_u|),x} - q_{\gamma_x(1),x}$ and $q_{\gamma_y(|P_u|),y} - q_{\gamma_y(1),y}$ in $O(1)$ time, we can determine whether u should be split vertically or horizontally.

If the cutting line at u is vertical and has x -coordinate X , we can decide in $O(1)$ time whether $X < q_{\gamma_x(a),x}$ or $X > q_{\gamma_x(a),x}$. The search of $A_{u,0}$ terminates at a leaf representing the gap between $q_{\gamma_x(c),x}$ and $q_{\gamma_x(c+1),x}$ for some c . It means that the cut at X should lead us to the child w of u such that the label uw represents a vertical cut between the c -th and $(c+1)$ -th smallest x -coordinates. The weight of the leaf $A_{u,0}$ corresponding to child w is $\text{weight}(w)$. The search time of $A_{u,0}$ is thus $O(\log(\text{weight}(u)/\text{weight}(w)))$.

The search tree $A_{u,1}$ is symmetrically organized for the horizontal cuts using γ_y .

Since we preserve the time to descend from a node of the trie to its appropriate child as in the proof of Theorem 6, the overall expected search time of the trie is still $O(|G_k| + H_f)$.

The construction of T_2 has been described in Theorem 6. The construction of $T_{3,v}$ for each leaf v of T_2 follows the procedure to construct a split tree [3]. The construction of the auxiliary structures γ_x 's, γ_y 's, $A_{u,0}$'s, and $A_{u,1}$'s in the training phase is also similar to the construction of analogous structures in Theorem 6. ◀

Given a point set P and $Q \subseteq P$, we can construct $SplitT(Q)$ from $SplitT(P)$ as follows. Make a copy T of $SplitT(P)$. Remove all leaves of T that represent points in $P \setminus Q$. Repeatedly remove nodes in T with only one child until there is none. For each node u of $SplitT(P)$ that is inherited by $SplitT(Q)$, the same cut that splits u in $SplitT(P)$ is also used in splitting u in $SplitT(Q)$. So $SplitT(Q)$ may not be a halving split tree. For every surviving node u in $SplitT(Q)$, $R(u)$ may shrink due to point deletions, but $\hat{R}(u)$ may remain the same or expand. For example, if parent of u is deleted but the grandparent of u survives, then $\hat{R}(u)$ in $SplitT(Q)$ is equal to $\hat{R}(\text{parent}(u))$ in $SplitT(P)$. Hence, (1) is still satisfied by $SplitT(Q)$.

► **Lemma 11.** *Suppose that we have constructed $SplitT(P)$ for a point set P .*

- (i) *For any $Q \subseteq P$, $SplitT(Q)$ can be computed from $SplitT(P)$ in $O(|P|)$ time.*
- (ii) *For any subsets Q_1, \dots, Q_m of P , if each Q_i is ordered as in the preorder traversal of $SplitT(P)$, then $SplitT(Q_1), \dots, SplitT(Q_m)$ can be computed from $SplitT(P)$ in $O(\alpha(|P|) \cdot (|P| + \sum_{i=1}^m |Q_i|))$ time, where $\alpha(\cdot)$ is the inverse Ackermann function.*

Proof. The correctness of (i) follows from our previous discussion. For (ii), the construction of $SplitT(Q_i)$ boils down to $O(|Q_i|)$ nearest common ancestor queries in $SplitT(P)$, which can be solved in the time stated [20]. There are solutions without the factor $\alpha(\cdot)$, but they require table lookup which is incompatible with the comparison-based model here. ◀

We also need the following result that follows from the works in [3, 2]. The proof is in the full version [7].

► **Lemma 12.** *There is a randomized algorithm that constructs $Del(P)$ from $SplitT(P)$ in $O(|P|)$ expected time.*

3.2 Operation phase

Let $I = (p_1, \dots, p_n)$ be an input instance. The construction of $Del(I)$ proceeds as follows.

1. For each k , compute $B_k(I|_{G_k})$ and $\Pi_k(I|_{G_k})$ using Theorem 10.
2. For $i \in G_k$, $B_k(I|_{G_k})$ gives the triangle $t' \in Del(V)$ that contains p_i , and a BFS in $Del(V)$ from t' gives $\Delta_i = \{t \in Del(V) : p_i \in C_t\}$, where C_t is the circumscribing disk of t .
3. For each k ,
 - a. $\Pi_k(I|_{G_k})$ gives $SplitT(I|_{G_k})$ (note that the halving split tree is also a fair split tree);
 - b. traverse $SplitT(I|_{G_k})$ in preorder to produce an ordered list Q_k of points in $I|_{G_k}$;
 - c. initialize $Q_{k,t} = \emptyset$ for all $t \in \bigcup_{i \in G_k} \Delta_i$;
 - d. for all $p_i \in Q_k$ (in order) and $t \in \Delta_i$, append p_i to $Q_{k,t}$;
4. Compute $SplitT(Q_{k,t})$ for all k and $t \in \bigcup_{i \in G_k} \Delta_i$ from $SplitT(I|_{G_k})$ using Lemma 11(ii).
5. For all k and $t \in \bigcup_{i \in G_k} \Delta_i$, compute $Del(Q_{k,t})$ from $SplitT(Q_{k,t})$ using Lemma 12.
6. Compute $Vor(V \cup I)$ and hence $Del(V \cup I)$ from the $Del(Q_{k,t})$'s over all k and t .
7. Split $Del(V \cup I)$ to produce $Del(I)$ and $Del(V)$. Return $Del(I)$.

By Lemma 9 and Theorem 10, the training phase takes $\tilde{O}(n^{10})$ time. Most of time is spent on constructing the tries for the groups G_1, G_2, \dots to support the retrieval of the $B_k(I|_{G_k})$'s and $\Pi_k(I|_{G_k})$'s.

By Theorem 10, step 1 takes $O(n + \sum_k H_{B_k} + \sum_k H_{\Pi_k})$ expected time. Observe that $|Q_{k,t}|$ is the total number of pairs (p_i, t) , where $p_i \in I$ and $t \in Del(V)$, such that $i \in G_k$ and $p_i \in C_t$. Therefore, $\sum_{k,t} |Q_{k,t}| = \sum_{i=1}^n |\Delta_i|$. By Lemma 8, $E[\sum_{i=1}^n |\Delta_i|] = O(n)$, and therefore, $E[\sum_k |Q_{k,t}|] = O(1)$. So steps 2 and 3 run in $O(n)$ expected time. By Lemma 11(ii), the expected running time of step 4 is $O(\sum_k |G_k| \alpha(n) + E[\sum_{k,t} |Q_{k,t}| \alpha(n)]) =$

$O(n\alpha(n) + \mathbb{E}[\sum_{i=1}^n |\Delta_i|] \alpha(n)) = O(n\alpha(n))$. By Lemma 12, the expected running time of step 5 is $O(\mathbb{E}[\sum_{k,t} |Q_{k,t}|]) = O(n)$. For step 7, a randomized algorithm is given in [6] that splits $\text{Del}(V \cup I')$ in $O(|V| + |I'|) = O(n)$ expected time. It remains to discuss the implementation and running time of step 6.

Step 6 is decomposed into two tasks. Let P_t be subset of I that lie in C_t , i.e., $P_t = \bigcup_k Q_{k,t}$. First, for all $t \in \text{Del}(V)$, compute $\text{Del}(P_t)$ by merging the non-empty $\text{Del}(Q_{k,t})$'s over all k . Second, construct $\text{Vor}(V \cup I)$ by merging the $\text{Vor}(P_t)$'s over all $t \in \text{Del}(V)$.

The first task is equivalent to computing $\text{Vor}(P_t)$ for all $t \in \text{Del}(V)$. It is known how to merge two Voronoi diagrams in linear time [4, 5]. So we can merge the non-empty $\text{Vor}(Q_{k,t})$'s in $O(\sum_k z_t |Q_{k,t}|)$ time, where z_t is the number of groups G_k 's with points in C_t . The expected running time of the first task is $O(\mathbb{E}[\sum_t \sum_k z_t |Q_{k,t}|])$. We have seen the analysis of the similar quantity $O(\mathbb{E}[\sum_r \sum_k |Z_r| |\sigma_{k,r}|])$ in the proof of Theorem 7, and the same analysis gives $\mathbb{E}[\sum_t z_t \sum_k |Q_{k,t}|] = O(\sum_t \sum_k \mathbb{E}[|Q_{k,t}|]) = O(n)$.

Consider the second task of merging the $\text{Vor}(P_t)$'s over all $t \in \text{Del}(V)$. We use the same strategy in [1]. For each $t \in \text{Del}(V)$, let ν_t be the Voronoi vertex dual to t in $\text{Vor}(V)$. For each Voronoi cell C of $\text{Vor}(V)$, pick the vertex ν_t of C such that $|P_t|$ is smallest, breaking ties by selecting t with the smallest id in $\text{Del}(V)$, and then triangulate C by connecting ν_t to other vertices of C . This gives the *geode triangulation* of $\text{Vor}(V)$ with respect to I .

► **Lemma 13** ([1]). *For any geode triangle $\tau = \nu_{t_1} \nu_{t_2} \nu_{t_3}$, $\text{Vor}(V \cup I) \cap \tau = \text{Vor}(\{v_\tau\} \cup \bigcup_{i=1}^3 P_{t_i}) \cap \tau$, where v_τ is the point in V whose Voronoi cell contains τ .*

By Lemma 13, we can compute $\text{Vor}(\{v_\tau\} \cup \bigcup_{i=1}^3 P_{t_i}) \cap \tau$ for all geode triangles τ , and stitch these fragments to form $\text{Vor}(V \cup I)$. The expected running time is dominated by the expected construction time of $\text{Vor}(\{v_\tau\} \cup \bigcup_{i=1}^3 P_{t_i})$ for all τ 's. We merge $\text{Vor}(P_{t_1}), \text{Vor}(P_{t_2}), \text{Vor}(P_{t_3}), v_\tau$ to form $\text{Vor}(\{v_\tau\} \cup \bigcup_{i=1}^3 P_{t_i})$ in linear time [4, 5]. The expected merging time is $O(\mathbb{E}[1 + \sum_{i=1}^3 |P_{t_i}|]) = O(1)$ because $\mathbb{E}[|P_{t_i}|] = O(1)$ by Lemma 8.

In summary, we conclude that step 6 runs in $O(n)$ expected time.

► **Theorem 14.** *Under the group product distribution setting, there is a self-improving Delaunay triangulator with a limiting complexity of $O(n\alpha(n) + H_{\text{DT}})$. The storage needed by the operation phase is $O(n^9)$. The training phase processes $\tilde{O}(n^9)$ instances in $\tilde{O}(n^{10})$ time using $O(n^9)$ space. The success probability is at least $1 - O(1/n)$.*

Proof. The training time complexity and the space complexities of the training and operation phases follow from previous discussion. Moreover, as discussed earlier, the expected running time in the operation phase is $O(n\alpha(n) + \sum_k H_{B_k} + \sum_k H_{\Pi_k})$. Given I and $\text{Del}(I)$, an algorithm is given in [1, Section 4.2] for finding the triangles in $\text{Del}(V)$ that contain the points in I . The same algorithm also works in the group product distribution setting. The expected running time of this algorithm is dominated by $\mathbb{E}[\sum_{i=1}^n |\Delta_i|]$, which is $O(n)$ as we argued previously. We can then apply the result in [1, Lemma 2.3] to conclude that $\sum_k H_{B_k} = O(n + H_{\text{DT}})$. There are $O(|G_k|^8)$ different outputs of Π_k , so $\sum_k H_{\Pi_k} = O(\sum_k \ln |G_k|) = O(n)$. ◀

References

- 1 N. Ailon, B. Chazelle, K. Clarkson, D. Liu, W. Mulzer, and C. Seshadhri. Self-improving algorithms. *SIAM Journal on Computing*, 40(2):350–375, 2011. doi:10.1137/090766437.
- 2 K. Buchin and W. Mulzer. Delaunay triangulations in $O(\text{sort}(N))$ time and more. *Journal of the ACM*, 58(2):6:1–6:27, 2011. doi:10.1145/1944345.1944347.

- 3 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(1):67–90, 1995. doi:10.1145/200836.200853.
- 4 T.M. Chan. A simpler linear-time algorithm for intersecting two convex polyhedra in three dimensions. *Discrete & Computational Geometry*, 56(4):860–865, 2016. doi:10.1007/s00454-016-9785-3.
- 5 B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM Journal on Computing*, 21(4):671–696, 1992. doi:10.1137/0221041.
- 6 B. Chazelle, O. Devillers, F. Hurtado, M. Mora, V. Sacristan, and M. Teillaud. Splitting a delaunay triangulation in linear time. *Algorithmica*, 34(1):39–46, 2002. doi:10.1007/s00453-002-0939-8.
- 7 S.-W. Cheng, M.-K. Chiu, K. Jin, and M.T. Wong. A generalization of self-improving algorithms. *CoRR*, abs/2003.08329, 2020. arXiv:2003.08329.
- 8 S.-W. Cheng, K. Jin, and L. Yan. Extensions of self-improving sorters. *Algorithmica*, 82:88–106, 2020. doi:10.1007/s00453-019-00604-6.
- 9 K.L. Clarkson, W. Mulzer, and C. Seshadhri. Self-improving algorithms for coordinatewise maxima and convex hulls. *SIAM Journal on Computing*, 43(2):617–653, 2014. doi:10.1137/12089702X.
- 10 K.L. Clarkson and K. Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete and Computational Geometry*, 37:43–58, 2007. doi:10.1007/s00454-006-1273-8.
- 11 T.M. Cover and J.A. Thomas. *Elements of Information Theory*. Wiley-Interscience, New York, 2 edition, 2006.
- 12 J.R. Driscoll, N. Sarnak, D.D. Sleator, and R.E. Tarjan. Making data structures persistent. *Journal of Computer System and Sciences*, 38:86–124, 1989. doi:10.1016/0022-0000(89)90034-2.
- 13 M.L. Fredman. Two applications of a probabilistic search technique: sorting $x + y$ and building balanced search trees. In *Proceedings of the 7th ACM Symposium on Theory of Computing*, pages 240–244, 1975. doi:10.1145/800116.803774.
- 14 M.L. Fredman. How good is the information theory bound in sorting? *Theoretical Computer Science*, 1(4):355–361, 1976. doi:10.1016/0304-3975(76)90078-5.
- 15 W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963. URL: <http://www.jstor.org/stable/2282952>.
- 16 J. Iacono. Expected asymptotically optimal planar point location. *Computational Geometry: Theory and Applications*, 29:19–22, 2004. doi:10.1016/j.comgeo.2004.03.010.
- 17 J.H. Kim. On increasing subsequences of random permutations. *Journal of Combinatorial Theory, Series A*, 76(1):148–155, 1996. doi:10.1006/jcta.1996.0095.
- 18 D.H. Lehmer. Teaching combinatorial tricks to a computer. In *Proceedings of Symposia in Applied Mathematics, Combinatorial Analysis*, volume 10, pages 179–193. American Mathematics Society, 1960.
- 19 K. Mehlhorn. Nearly optimal binary search trees. *Acta Informatica*, 5:287–295, 1975. doi:10.1007/BF00264563.
- 20 R.E. Tarjan. Applications of path compression on balanced trees. *Journal of the ACM*, 26(4):690–715, 1979. doi:10.1145/322154.322161.