Extensions of Self-Improving Sorters

Siu-Wing Cheng[1]
HKUST, Hong Kong, China
scheng@cse.ust.hk
https://orcid.org/0000-0002-3557-9935

Lie Yan[2]
Hangzhou, China
lieyan@yahoo.com

Abstract

Ailon et al. (SICOMP 2011) proposed a self-improving sorter that tunes its performance to the unknown input distribution in a training phase. The distribution of the input numbers $x_1, x_2, \ldots, x_n$ must be of the product type, that is, each x_i is drawn independently from an arbitrary distribution $\mathcal{D}_i$, and the $\mathcal{D}_i$'s are independent of each other. We study two extensions that relax this requirement. The first extension models hidden classes in the input. We consider the case that numbers in the same class are governed by linear functions of the same hidden random parameter. The second extension considers a hidden mixture of product distributions.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases sorting, self-improving algorithms, entropy

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2018.63

1 Introduction

Self-improving algorithms proposed by Ailon et al. [1] can tune their computational performance to the input distribution. There is a *training phase* in which the algorithm learns certain input features and computes some auxiliary structures. After the training phase, the algorithm uses these auxiliary structures in the *operation phase* to obtain an expected time complexity that is no worse and possibly better than the best worst-case complexity known. The expected time complexity in the operation phase is called the *limiting complexity.*

This computational model addresses two issues. First, the worst-case scenario may not happen, and so the worst-case optimal performance may not be the best possible. Second, previous efforts for mitigating the worst-case scenarios often consider average-case complexities, and the input distributions are assumed to be simple distributions like Gaussian, uniform, Poisson, etc. whose parameters are given beforehand. In contrast, Ailon et al. only assume that individual input items are independently distributed, while the distribution of an input item can be arbitrary. No other information is needed.

The problems of sorting and two-dimensional Delaunay triangulation are studied by Ailon et al. [1]. The sorting problem input I has n numbers. The i-th number is drawn from a hidden distribution $\mathcal{D}_i$, and the $\mathcal{D}_i$'s are independent from each other. The joint distribution $\prod_{i=1}^{n} \mathcal{D}_i$ is called a *product distribution.* Let $\pi(I)$ denote the sequence of the ranks of the x_i's, which is a permutation of $[n]$. It is shown that for any $\varepsilon \in (0, 1)$, there is a self-improving algorithm with limiting complexity $O(\varepsilon^{-1}(n + H_\pi))$, where H_π is the

[1] Supported by Research Grants Council, Hong Kong, China (project no. 16200317).
[2] Part of the work was conducted while the author was at HKUST and supported by the Hong Kong PhD Fellowship.

entropy of the distribution of $\pi(I)$. By Shannon's theory [3], any comparison-based sorting algorithm takes $\Omega(n + H_\pi)$ expected time. The self-improving sorter requires $O(n^{1+\varepsilon})$ space. The training phase processes $O(n^\varepsilon)$ input instances in $O(n^{1+\varepsilon})$ time, and it succeeds with probability at least $1 - 1/n$, i.e., the probabiliy of achieving the desired limiting complexity is at least $1 - 1/n$. For two-dimensional Delaunay triangulation, Ailon et al. also obtained an optimal limiting complexity for product distributions.

Subsequently, Clarkson et al. [2] developed self-improving algorithms for two-dimensional coordinatewise maxima and convex hull, assuming that the input comes from a product distribution. The limiting complexities for the maxima and the 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.

On one hand, the product distribution requirement is very strong; on the other hand, Ailon et al. showed that $\Omega(2^{n \log n})$ bits of storage are necessary for optimal sorting if the n numbers are drawn from an arbitrary distribution. We study two extensions of the input model that are natural and yet possess enough structure for efficient self-improving algorithms to be designed.

The first extension models the situations in which some input elements depend on each other. We consider a hidden partition of the input $I = (x_1, \cdots, x_n)$ into classes S_k's such that all x_i's in a class S_k are distinct linear functions of the same hidden random parameter z_k, and the distributions of the z_k's are arbitrary and independent of each other.[3] We call this model a *product distribution with hidden linear classes*. Choose any $\varepsilon \in (0, 1)$. Our self-improving sorter has an $O(n/\varepsilon + H_\pi/\varepsilon)$ limiting complexity, uses $O(n^2)$ space, and requires a training phase that processes $O(n^\varepsilon)$ input instances in $O(n^2 \log^3 n)$ time with a success probability at least $1 - 1/n$.

In the second extension, the distribution of I is a mixture $\sum_{q=1}^{\kappa} \lambda_q \mathcal{D}_q$, where κ and the λ_q's are hidden, and every $\mathcal{D}_q$ is a hidden product distribution of n real numbers. In other words, over a large collection of input instances, for all $q \in [1, \kappa]$, a fraction λ_q of them are expected to be drawn from $\mathcal{D}_q$. We assume that an upper bound $m \geq \kappa$ is given. We call this model *a hidden mixture of product distributions*. For any $\varepsilon \in (0, 1)$, our sorter has an $O(n \log\log(mn) + (n/\varepsilon)\log\kappa + H_\pi/\varepsilon)$ limiting complexity[4], uses $O(mn + m^\varepsilon n^{1+\varepsilon})$ space, and requires a training phase that processes $O(m(\log m + \log n) + n^\varepsilon)$ instances in $O(mn(\log m + \log n)^2 + m^\varepsilon n^{1+\varepsilon})$ time with a success probability at least $1 - 1/n$.

2 Hidden linear classes

There is a hidden partition of $[n]$ into classes. For every $i \in [1, n]$, the distribution of x_i is degenerate if x_i is equal to a fixed value. Each such x_i will be recognized in the training phase and i will be put in a class by itself. For the remaining i's, the distributions of x_i's are non-degenerate, and we use $S_1, \cdots, S_g$ to denote the hidden classes formed by them. Numbers in the same class S_k are generated by linear functions of the same hidden random parameter z_k. Different classes are governed by different random parameters. We know that the functions are linear, but no other information is given to us.

Let $\mathcal{D}_k$ denote the distribution of z_k. There is a technical condition that is required of the $\mathcal{D}_k$'s: there exists a constant $\rho \in (0, 1)$ such that for every $k \in [1, g]$ and every $c \in \mathbb{R}$, $\Pr[z_k = c] \leq 1 - \rho$. This condition says that $\mathcal{D}_k$ does not over-concentrate on any single value, which is quite a natural phenonmeon. Our algorithm does not need to know ρ.

[3] There is a technical condition required of the input distribution to be explained in Section 2.
[4] A less sophisticated method replaces $n \log\log(mn)$ by mn which is beneficial for $m = o(\log\log n)$.

2.1 Training phase

2.1.1 Learn the linear classes

We learn the classes and the linear functions using the first $3\ln^2 n$ input instances. Denote these instances by $I_1, I_2, \cdots, I_{3\ln^2 n}$. Let $x_i^{(a)}$ denote the i-th input number in I_a. We first recognize the degenerate distributions by checking which $x_i^{(a)}$ is fixed for $a \in [1, 3\ln^2 n]$.

▶ **Lemma 1.** *Assume that $n \geq e^{2/(3\rho)}$. It holds with probability at least $1 - 1/n$ that for all $i \in [1, n]$, if $x_i^{(a)}$ is the same for all $a \in [1, 3\ln^2 n]$, the distribution of $x_i^{(a)}$ is degenerate.*

Proof. Let c_i be the observed value of $x_i^{(a)}$ for $a \in [1, 3\ln^2 n]$. If the distribution of $x_i^{(a)}$ is not degenerate, the probability of $x_i^{(a)} = c_i$ for all $a \in [1, 3\ln^2 n]$ is at most $(1-\rho)^{3\ln^2 n} \leq e^{-3\rho\ln^2 n} \leq e^{-2\ln n} = n^{-2}$. Applying the union bound establishes the lemma. ◀

Assume that the degenerate distributions are taken out of the consideration. If i and j belong to the same class S_k, then $x_i^{(a)}$ and $x_j^{(a)}$ are linearly related as a varies. Conversely, if i and j belong to different classes, it is highly unlikely that $x_i^{(a)}$ and $x_j^{(a)}$ remain linearly related as a varies because they are governed by independent random parameters. We check if the triples of points $(x_i^{(a-2)}, x_j^{(a-2)})$, $(x_i^{(a-1)}, x_j^{(a-1)})$, and $(x_i^{(a)}, x_j^{(a)})$ are collinear for each I_a, $a \in [3, 3\ln^2 n]$, and every distinct pair of i and j from $[1, n]$. We quantify this intuition in the following result.

▶ **Lemma 2.** *Let i and j be two distinct indices in $[1, n]$ that belong to different classes. For every $a \in [3, 3\ln^2 n]$, let $E_{ij}^{(a)}$ denote the event that the points $(x_i^{(a-2)}, x_j^{(a-2)})$, $(x_i^{(a-1)}, x_j^{(a-1)})$, and $(x_i^{(a)}, x_j^{(a)})$ are not collinear. For any $n \geq e^{3/\rho^2}$, $\Pr\left[\bigcup_{a=3}^{3\ln^2 n} E_{ij}^{(a)}\right] \geq 1 - n^{-3}$.*

Proof. We first prove a lower bound for $E_{ij}^{(3a)}$ for $a \in [1, \ln^2 n]$. It is well known [4] that the points $(x_i^{(3a-2)}, x_j^{(3a-2)})$, $(x_i^{(3a-1)}, x_j^{(3a-1)})$, and $(x_i^{(3a)}, x_j^{(3a)})$ are collinear if and only if

$$\begin{vmatrix} x_i^{(3a-2)} & x_j^{(3a-2)} & 1 \\ x_i^{(3a-1)} & x_j^{(3a-1)} & 1 \\ x_i^{(3a)} & x_j^{(3a)} & 1 \end{vmatrix} = 0. \tag{1}$$

Assume that $x_i^{(3a-2)} = c_1$ and $x_i^{(3a-1)} = c_2$ for two fixed values c_1 and c_2. Since i and j are in different classes, $x_i^{(b)}$ and $x_j^{(b')}$ are independent for all b and b'. Second, x_j in one instance I_b does not influence x_j in a different instance $I_{b'}$. So there is no dependence among $x_i^{(3a)}$, $x_j^{(3a-2)}$, $x_j^{(3a-1)}$, and $x_j^{(3a)}$.

Suppose that $c_1 \neq c_2$. If $E_{ij}^{(3a)}$ does not occur, then by (1), we can express $x_j^{(3a)}$ as a function $f(c_1, c_2, x_i^{(3a)}, x_j^{(3a-2)}, x_j^{(3a-1)})$. Hence,

$$\begin{aligned} & \Pr\left[E_{ij}^{(3a)} | x_i^{(3a-2)} = c_1 \wedge x_i^{(3a-1)} = c_2 \wedge c_1 \neq c_2\right] \\ = & \sum_{c_3, c_1', c_2'} \Pr\left[x_i^{(3a)} = c_3 \wedge x_j^{(3a-2)} = c_1' \wedge x_j^{(3a-1)} = c_2'\right] \cdot \Pr\left[x_j^{(3a)} \neq f(c_1, c_2, c_3, c_1', c_2')\right] \\ \geq & \sum_{c_3, c_1', c_2'} \Pr\left[x_i^{(3a)} = c_3 \wedge x_j^{(3a-2)} = c_1' \wedge x_j^{(3a-1)} = c_2'\right] \cdot \rho \\ = & \rho. \end{aligned}$$

If $c_1 = c_2$, then (1) becomes $(x_i^{(3a)} - x_i^{(3a-1)})(x_j^{(3a-1)} - x_j^{(3a-2)}) = 0$. Thus,

$$\begin{aligned}
& \Pr\left[E_{ij}^{(3a)} | x_i^{(3a-2)} = c_1 \wedge x_i^{(3a-1)} = c_2 \wedge c_1 = c_2\right] \\
= & \Pr\left[x_j^{(3a-2)} \neq x_j^{(3a-1)}\right] \cdot \Pr\left[x_i^{(3a)} \neq c_1\right] \\
\geq & \left(1 - \Pr\left[x_j^{(3a-2)} = x_j^{(3a-1)}\right]\right) \cdot \rho \\
= & \rho \cdot \left(1 - \sum_c \Pr\left[x_j^{(3a-2)} = c\right] \cdot \Pr\left[x_j^{(3a-1)} = c\right]\right) \\
\geq & \rho \cdot \left(1 - (1-\rho) \sum_c \Pr\left[x_j^{(3a-2)} = c\right]\right) \\
= & \rho^2.
\end{aligned}$$

The above shows that the probability of $E_{ij}^{(3a)}$ conditioned on some fixed values of $x_i^{(3a-2)}$ and $x_i^{(3a-1)}$ is at least ρ^2. Hence, $\Pr\left[E_{ij}^{(3a)}\right] \geq \rho^2 \cdot \sum_{c_1,c_2} \Pr\left[x_i^{(3a-2)} = c_1 \wedge x_i^{(3a-1)} = c_2\right] = \rho^2$.

The events in $\bigcup_{a=1}^{\ln n} E_{ij}^{(3a)}$ are independent of each other. Therefore,

$$\Pr\left[\bigcup_{a=3}^{3\ln^2 n} E_{ij}^{(a)}\right] \geq \Pr\left[\bigcup_{a=1}^{\ln^2 n} E_{ij}^{(3a)}\right] = 1 - \prod_{a=1}^{\ln^2 n} \Pr\left[\overline{E}_{ij}^{(3a)}\right] \geq 1 - (1-\rho^2)^{\ln^2 n}.$$

Since $n \geq e^{3/\rho^2}$, we get $(1-\rho^2)^{\ln^2 n} \leq e^{-\rho^2 \ln^2 n} \leq e^{-3\ln n} = n^{-3}$, establishing the lemma. ◀

By Lemma 2, we keep a dictionary that stores (i, j, b_{ij}) for all $i \neq j$ and $i, j \in [1, n]$ such that the distributions of x_i and x_j are non-degenerate. Initially, $b_{ij} = 1$ for all (i, j). For each I_a where $a \in [3, 3\ln^2 n]$, we perform the following. For every (i, j), we check the event $E_{ij}^{(a)}$ in $O(1)$ time, set a bit variable $\beta = 0$ if $E_{ij}^{(a)}$ occurs and $\beta = 1$ otherwise, and then update $b_{ij} := b_{ij} \wedge \beta$. After going through all $3\ln^2 n$ input instances, we put x_i and x_j in the same class if and only if $b_{ij} = 1$. By Lemmas 1 and 2 and the union bound, the classification is correct with probability at least $1 - 1/n$. The processing time needed is $O(n^2 \log^3 n)$.

2.1.2 Structures for the operation phase

After we obtain the classes, for each class S_k, we fix an arbitrary index $s_k \in S_k$. Then, we compute the equation of the line ℓ_i that expresses x_i as a linear function in x_{s_k} for each $i \in S_k \setminus \{s_k\}$. This can be done by picking any two input instances I_a and I_b, and then computing the equation of the support line through $(x_{s_k}^{(a)}, x_i^{(a)})$ and $(x_{s_k}^{(b)}, x_i^{(b)})$ in $O(1)$ time. The processing time needed over all classes is $O(n)$.

Take another $\ln n$ input instances. Sort all numbers in these instances into one sorted list L. Form the *V-list* $(v_0, v_1, \cdots, v_n, v_{n+1})$, where $v_0 = -\infty$, $v_{n+1} = \infty$, and v_i has rank $i \ln n$ in the list L. The V-list requires $O(n)$ space and can be computed in $O(n \log^2 n)$ time. Note that if the distribution of x_i is degenerate, the same x_i appears $\ln n$ times in the sorted list L, which implies that x_i must be selected to be an element of the V-list.

The V-list induces n horizontal lines at y-coordinates $v_1, v_2, \cdots, v_n$. Let $\mathcal{A}_k$ denote the arrangement of the lines ℓ_i computed for $i \in S_k \setminus \{s_k\}$. We overlay these horizontal lines on top of $\mathcal{A}_k$. We draw vertical lines through the intersections between these horizontal lines and the lines in $\mathcal{A}_k$. We also draw vertical lines through the vertices of $\mathcal{A}_k$. The plane is divided into a set W of vertical slabs, where $|W| = O(n|S_k|)$. Within each slab in W, each line ℓ_i in $\mathcal{A}_k$ lies strictly between two consecutive values v_r and v_{r+1}, i.e., v_r is the predecessor of ℓ_i in the V-list.

By a plane sweep over $\mathcal{A}_k$ and the n horizontal lines, we can figure out the predecessor of ℓ_i within each slab in W. For each slab in W, we store a list of the ℓ_i's in bottom-to-top order, and each line in the list stores its predecessor in the V-list. The lists for two consecutive slabs differ by either swapping two lines in $\mathcal{A}_k$ or changing the predecessor of a line in $\mathcal{A}_k$. Therefore, the $|W|$ lists can be stored in $O(|W| + |S_k|) = O(n|S_k|)$ space using a persistent lists data structure [5]. These persistent lists can be generated by a plane sweep over $\mathcal{A}_k$ and the n horizontal lines in $O(n|S_k| \log n)$ time.

We need to provide fast access to a particular slab in W after specifying x_{s_k}. Take another n^ε input instances for any choice of $\varepsilon \in (0, 1)$. Record the frequencies of x_{s_k} falling into the slabs in W among these n^ε instances. We build a binary search tree on these slabs whose expected search time is asymptotically optimal with respect to the recorded frequencies. Let T_k denote this asymptotically optimal binary search tree. There are $O(n|S_k|)$ nodes in T_k. At each node of T_k, we store the persistent list of lines in $\mathcal{A}_k$ in bottom-to-top order within the slab corresponding to that node. The size of T_k is $O(n|S_k|)$, and it can be constructed in $O(n|S_k|)$ time [6, 8]. Very low frequencies cannot give good estimate of the probability distribution of x_{s_k}, so navigating down T_k to a node of very low frequency may be too time-consuming. Thus, if a search of T_k reaches a node at depth below $\frac{\varepsilon}{3} \log_2 n$, we abort and perform a binary search among the slabs in W, which takes $O(\log |W|) = O(\log n)$ time.

The last ingredient is to allow the predecessor of x_{s_k} in the V-list to be quickly located for all $k \in [1, g]$. We record the frequencies of x_{s_k} falling to the intervals $[v_r, v_{r+1})$ among the n^ε instances. Then, we build an asymptotically optimal binary search tree $\hat{T}_k$ with respect to these frequencies. The tree $\hat{T}_k$ uses $O(n)$ space, and it can be constructed in $O(n)$ time [6, 8]. As in the case of T_k, if a search of $\hat{T}_k$ reaches a node at depth below $\frac{\varepsilon}{3} \log_2 n$, we abort and perform a binary search in the V-list, which takes $O(\log n)$ time.

2.2 Operation phase

Given an input instance $I = (x_1, \cdots, x_n)$, for each class S_k, we query T_k with x_{s_k} to retrieve the sorted list σ_k of numbers belonging to the class S_k. Precisely, T_k gives fast access to the sorted sequence $\sigma_k \setminus \{x_{s_k}\}$, and then we spend $O(|\sigma_k|)$ time to insert x_{s_k} into $\sigma_k \setminus \{x_{s_k}\}$. The numbers in $\sigma_k \setminus \{x_{s_k}\}$ are already stored with their predecessors in the V-list. We query $\hat{T}_k$ to obtain the predecessor of x_{s_k} in the V-list.

Initialize an empty set Z_r of lists for each interval $[v_r, v_{r+1})$. For each x_i that is degenerately distributed, add x_i to Z_r where $v_r = x_i$. For each $k \in [1, g]$, if $\sigma_k \cap [v_r, v_{r+1})$ is non-empty, add $\sigma_k \cap [v_r, v_{r+1})$ to Z_r. Distributing σ_k to the Z_r's takes $O(|\sigma_k|) = O(|S_k|)$ time. Merge all lists in Z_r into one sorted list. The merging is facilitated by a min-heap that stores the next element from each list in Z_r to be considered for the next output number for the merged list. Thus, each step of the merging takes $O(\log |Z_r|)$ time. Finally, we concatenate in $O(n)$ time the merged lists for all Z_r's to form the output sorted list.

Correctness is obvious. The limiting complexity has two main components. First, the sum of expected query times of T_k and $\hat{T}_k$ for $k \in [1, g]$. Second, the total time spent on merging the lists in Z_r for $r \in [0, n]$. The remaining processing time is $O(n + \sum_{k=1}^{g} |S_k|) = O(n)$. We give the analysis in the next section to show that the first two components sum to $O(n/\varepsilon + H_\pi/\varepsilon)$. Recall that $\pi(I)$ is the sequence of the ranks of numbers in I, which is a permutation of $[n]$, and H_π is the entropy of the distribution of $\pi(I)$.

2.3 Analysis

Assign labels 0 to $n+1$ to $v_0, v_1, \cdots, v_n, v_{n+1}$ in this order. Similarly, assign labels $n+2$ to $2n+1$ to the input numbers $x_1, \cdots, x_n$ in this order.

Define the random variable B^V to be the permutation of the labels that appear from left to right after sorting $\{v_0, \cdots, v_{n+1}\} \cup \{x_1, \cdots, x_n\}$ in increasing order.

For each $k \in [1, g]$, define a random variable B_k^V to be the permutation of the labels that appear from left to right after performing the following operations: (1) sort $\{v_0, \cdots, v_{n+1}\} \cup \{x_i : i \in S_k \setminus \{s_k\}\}$ in increasing order, and (2) remove all v_r's that do not immediately precede some x_i's in the sorted list. Let H_k^V denote the entropy of the distribution of B_k^V. Determining the sorted order $\sigma_k \setminus \{x_{s_k}\}$ and these numbers' predecessors in the V-list takes at least H_k^V expectd time.

For each $k \in [1, g]$, define a random variable $\hat{B}_k^V$ to be the label of the predecessor of x_{s_k} in the V-list. Let $\hat{H}_k^V$ denote the entropy of the distribution of $\hat{B}_k^V$. Determining the predecessor of x_{s_k} in the V-list takes at least $\hat{H}_k^V$ expected time.

Our algorithm queries T_k and $\hat{T}_k$ for $k \in [1, g]$, constructs σ_k for $k \in [1, g]$ in $O(\sum_{k=1}^{g} |\sigma_k|)$ time, and then perform mergings in $O(n + \sum_{r=0}^{n} \sum_{k=1}^{g} |\sigma_k \cap [v_r, v_{r+1})| \log |Z_r|)$ time. The additive $O(n)$ term takes care of every interval that contains only one x_i that is degenerately distributed. Recall that $|Z_r|$ is the number of classes that have numbers falling into $[v_r, v_{r+1})$. If T_k and $\hat{T}_k$ were the ideal binary search trees, querying them would take H_k^V and $\hat{H}_k^V$ expected time, respectively. The total expected running time would then be

$$O\left(n + \sum_{k=1}^{g} H_k^V + \sum_{k=1}^{g} \hat{H}_k^V\right) + O\left(\mathrm{E}\left[\sum_{r=0}^{n} \sum_{k=1}^{g} |\sigma_k \cap [v_r, v_{r+1})| \log |Z_r|\right]\right). \tag{2}$$

We prove in the rest of this section that both $\sum_{k=1}^{g} H_k^V$ and $\sum_{k=1}^{g} \hat{H}_k^V$ are $O(n + H_\pi)$, and that $\mathrm{E}\left[\sum_{r=0}^{n} \sum_{k=1}^{g} |\sigma_k \cap [v_r, v_{r+1})| \log |Z_r|\right] = O(n)$. Moreover, although T_k and $\hat{T}_k$ are not ideal binary search trees, their expected query complexities are $O(H_k^V/\varepsilon)$ and $O(\hat{H}_k^V/\varepsilon)$, respectively, as shown in [1, Lemma 3.4]. Therefore, the total expected running time is $O(n/\varepsilon + H_\pi/\varepsilon)$.

We need two technical results.

▶ **Lemma 3** ([3, Theorem 2.5.1]). *Let $H(X_1, \cdots, X_n)$ be the joint entropy of independent random variables $X_1, \cdots, X_n$. Then $H(X_1, \cdots, X_n) = \sum_{i=1}^{n} H(X_i)$.*

▶ **Lemma 4** ([1, Lemma 2.3]). *Let $X : \mathcal{U} \to \mathcal{X}$ and $Y : \mathcal{U} \to \mathcal{Y}$ be two random variables obtained with respect to the same arbitrary distribution over the universe $\mathcal{U}$. Suppose that the function $f : (I, X(I)) \mapsto Y(I)$, $I \in \mathcal{U}$, can be computed by a comparison-based algorithm with C expected comparisons, where the expectation is over the distribution on $\mathcal{U}$. Then, $H(Y) \le C + O(H(X))$.*

We show that both $\sum_{k=1}^{g} H_k^V$ and $\sum_{k=1}^{g} \hat{H}_k^V$ are $O(n + H_\pi)$.

▶ **Lemma 5.**
(a) $\sum_{k=1}^{g} H_k^V = O\left(n + H(B^V)\right) = O\left(n + H_\pi\right)$,
(b) $\sum_{k=1}^{g} \hat{H}_k^V = O(n + H_\pi)$.

Proof. Consider (a). Suppose that we are given a setting of B^V, i.e., the permutation of labels from left to right in the sorted order of $\{v_0, \cdots, v_{n+1}\} \cup \{x_1, \cdots, x_n\}$. We scan the sorted list from left to right. We maintain the most recently scanned v_r. Suppose that we see a number x_i. Let S_k be the class to which x_i belongs. If this is the first time that we encounter an index in S_k, we initialize an output list for B_k^V that contains the label of v_r followed by the label of x_i. If this is not the first time that we encounter an index in S_k, we append the label of x_i to the output list for B_k^V. There is an exception that when $i = s_k$; we do not output the label of x_{s_k}. Clearly, we obtain the settings of all B_k^V's correctly from B^V. The number of comparisons needed is $O(n)$. Therefore, Lemmas 3 and 4 imply that $\sum_{k=1}^{g} H_k^V = H(B_1^V, \cdots, B_g^V) = O(n + H(B^V))$.

Given $(I, \pi(I))$, we use $\pi(I)$ to sort I and then merge the sorted order with $(v_0, \cdots, v_{n+1})$. Afterwards, we scan the sorted list to output the labels of the numbers. This gives the setting of B^V. Clearly, $O(n)$ comparisons suffice, and so Lemma 4 implies that $H(B^V) = O(n + H_\pi)$. This completes the proof of (a).

The settings of $\hat{B}_1^V, \cdots, \hat{B}_g^V$ can be derived similarly by using $\pi(I)$ to sort I, merging the sorted sequence with $(v_0, \cdots, v_{n+1})$, and then scanning the merged sequence. Then, Lemmas 3 and 4 imply that $\sum_{k=1}^{g} \hat{H}_k^V = H(\hat{B}_1^V, \cdots, \hat{B}_g^V) = O(n + H_\pi)$, establishing (b). ◀

We will show that it holds with high probability that $\mathrm{E}[|Z_r|] = O(1)$ for all $r \in [0, n]$ simultaneously. It implies that $\mathrm{E}\left[\max_{r \in [0,n]} |Z_r|\right] = O(1)$ with high probability. Then,

$$\begin{aligned} \mathrm{E}\left[\sum_{r=0}^{n} |\sigma_k \cap [v_r, v_{r+1})| \log |Z_r|\right] &\le \mathrm{E}\left[\max_{r \in [0,n]} |Z_r| \cdot \sum_{r=0}^{n} |\sigma_k \cap [v_r, v_{r+1})|\right] \\ &= |\sigma_k| \cdot \mathrm{E}\left[\max_{r \in [0,n]} |Z_r|\right] \\ &= O(|\sigma_k|). \end{aligned}$$

Hence,

$$\begin{aligned} \mathrm{E}\left[\sum_{r=0}^{n}\sum_{k=1}^{g} |\sigma_k \cap [v_r, v_{r+1})| \log |Z_r|\right] &= \sum_{k=1}^{g} \mathrm{E}\left[\sum_{r=0}^{n} |\sigma_k \cap [v_r, v_{r+1})| \log |Z_r|\right] \\ &\le O\left(\sum_{k=1}^{g} |\sigma_k|\right) \\ &= O(n). \end{aligned}$$

The second term in (2) can thus be replaced by $O(n)$.

Our proof of $\mathrm{E}[|Z_r|] = O(1)$ for all $r \in [0, n]$ with high probability is modeled after the proof of a similar result in [1]. There is a small twist due to the handling of the classification.

▶ **Lemma 6.** *It holds with probability at least $1 - O(1/n)$ that for all $r \in [0, n]$, $\mathrm{E}[|Z_r|] = O(1)$.*

Proof. Let $I_1, \cdots, I_{\ln n}$ denote the input instances used in the training phase for building the V-list. Let $y_1, y_2, \cdots, y_{n \ln n}$ denote the sequence formed by concatenating $I_1, \cdots, I_{\ln n}$ in this order. We adopt the notation that for each $\alpha \in [1, n \ln n]$, y_α belongs to the class S_{k_α} and the input instance I_{a_α}.

Fix any distinct index pairs $\alpha, \beta \in [1, n \ln n]$ such that $y_\alpha \le y_\beta$. Let $\mathcal{J}_\alpha^\beta$ be the set of index pairs $\{(a, k) : a \in [1, \ln n], k \in [1, g]\} \setminus \{(a_\alpha, k_\alpha), (a_\beta, k_\beta)\}$. For any $(a, k) \in \mathcal{J}_\alpha^\beta$, let $\mathrm{Y}_\alpha^\beta(a, k)$ be an indicator random variable such that if some element of the input instance I_a, $a \in [1, \ln n]$, belongs to S_k and falls into $[y_\alpha, y_\beta)$, then $\mathrm{Y}_\alpha^\beta(a, k) = 1$; otherwise, $\mathrm{Y}_\alpha^\beta(a, k) = 0$. Define $\mathrm{Y}_\alpha^\beta = \sum_{(a,k) \in \mathcal{J}_\alpha^\beta} \mathrm{Y}_\alpha^\beta(a, k)$.

Among the (a, k)'s in $\mathcal{J}_\alpha^\beta$, the random variables $\mathrm{Y}_\alpha^\beta(a, k)$ are independent from each other. By Chernoff's bound, for any $\mu \in [0, 1]$,

$$\Pr\left[\mathrm{Y}_\alpha^\beta \le (1 - \mu)\mathrm{E}[\mathrm{Y}_\alpha^\beta]\right] \le e^{-\mu^2 \mathrm{E}[\mathrm{Y}_\alpha^\beta]/2}.$$

Setting $\mu = \sqrt{35} - 5 \approx 0.9161$ shows that if $\mathrm{E}[Y_\alpha^\beta] > \frac{1}{6-\sqrt{35}} \ln n$, then $Y_\alpha^\beta > \ln n$ with probability at least $1 - n^{-5}$. Since the above statement holds for any fixed choices of α and β such that $y_\alpha \leq y_\beta$, we can apply the union bound to the $O(n^2 \log^2 n)$ possible choices of α and β and conclude that:

It holds with probability at least $1 - O(n^{-2})$ that for any distinct index pairs $\alpha, \beta \in [1, n \ln n]$ such that $y_\alpha \leq y_\beta$, if $\mathrm{E}[Y_\alpha^\beta] > \frac{1}{6-\sqrt{35}} \ln n$, then $Y_\alpha^\beta > \ln n$.

For every $r \in [0, n+1]$, let y_{α_r} denote v_r, where $y_{\alpha_0} = -\infty$ and $y_{\alpha_{n+1}} = \infty$. Fix a particular $r \in [0, n+1]$. By construction, there are $\ln n$ numbers among $I_1, \cdots, I_{\ln n}$ that fall in $[v_r, v_{r+1})$, which guarantees the event of $Y_{\alpha_r}^{\alpha_{r+1}} \leq \ln n$. Our previous conclusion implies that $\mathrm{E}[Y_{\alpha_r}^{\alpha_{r+1}}] \leq \frac{1}{6-\sqrt{35}} \ln n$ with probability at least $1 - O(n^{-2})$.

We relate $\mathrm{E}[Y_{\alpha_r}^{\alpha_{r+1}}]$ to $\mathrm{E}[|Z_r|]$ as follows. Let X_{kr} be the indicator random variable such that if some element of the input instance belongs to S_k and falls into $[v_r, v_{r+1})$, then $X_{kr} = 1$; otherwise, $X_{kr} = 0$. Then $\sum_{k=1}^g X_{kr} = |Z_r|$, implying that $\sum_{k=1}^g \mathrm{E}[X_{kr}] = \mathrm{E}[|Z_r|]$. The random process that generates the input instances is oblivious of the training phase. It follows that $\mathrm{E}[Y_{\alpha_r}^{\alpha_{r+1}}]$ should be the same as $\sum_{a=1}^{\ln n} \sum_{k=1}^g \mathrm{E}[X_{kr}]$, except that the index pairs $(a_{\alpha_r}, k_{\alpha_r})$ and $(a_{\alpha_{r+1}}, k_{\alpha_{r+1}})$ are excluded from $\mathcal{J}_{\alpha_r}^{\alpha_{r+1}}$ but these two cases are considered in $\sum_{a=1}^{\ln n} \sum_{k=1}^g \mathrm{E}[X_{kr}]$. Therefore,

$$\mathrm{E}[Y_{\alpha_r}^{\alpha_{r+1}}] \geq \left(\sum_{a=1}^{\ln n} \sum_{k=1}^{g} \mathrm{E}[X_{kr}] \right) - 2 = \ln n \cdot \mathrm{E}[|Z_r|] - 2. \tag{3}$$

We have shown previously that $\mathrm{E}[Y_{\alpha_r}^{\alpha_{r+1}}] \leq \frac{1}{6-\sqrt{35}} \ln n$ with probability at least $1 - O(n^{-2})$. It follows that $\mathrm{E}[|Z_r|] = O(1)$ with probability at least $1 - O(n^{-2})$. Since the above statement holds for every fixed $r \in [0, n]$, by the union bound, it holds with probability at least $1 - O(1/n)$ that $\mathrm{E}[|Z_r|] = O(1)$ for all $r \in [0, n]$. ◀

It remains to show that the expected query complexities of T_k and $\hat{T}_k$ are $O(H_k^V/\varepsilon)$ and $O(\hat{H}_k^V/\varepsilon)$, respectively. The argument is based on Chernoff's bound and the fact that if a search in T_k or $\hat{T}_k$ reaches a pruned node, it means that the search requires $\Omega(\varepsilon \log n)$ time. The exact same arguments have been made by Ailon et al. [1, Lemma 3.4].

▶ **Theorem 7.** *For any $\varepsilon \in (0, 1)$, there exists a self-improving sorter of $O(n/\varepsilon + H_\pi/\varepsilon)$ limiting complexity for any product distribution with hidden linear classes. The storage needed is $O(n^2)$. The training phase processes $O(n^\varepsilon)$ input instances in $O(n^2 \log^3 n)$ time, and it succeeds with probability at least $1 - 1/n$.*

3 Mixture of product distributions

Let κ be the number of product distributions in the mixture. Although κ is hidden, we are given an upper bound $m \geq \kappa$. Let $\mathcal{D}_q$, $q \in [1, \kappa]$, be the hidden product distributions in the mixture. In each $\mathcal{D}_q$, the i-th input number is drawn from $\mathcal{D}_{q,i}$, i.e., $\mathcal{D}_q = \prod_{i=1}^n \mathcal{D}_{q,i}$. The input distribution is $\sum_{q=1}^\kappa \lambda_q \mathcal{D}_q$ for some hidden positive λ_q's such that $\sum_{q=1}^\kappa \lambda_q = 1$.

3.1 Training phase

Take $m\,(\ln m + \ln n)$ input instances and sort all of these numbers in increasing order. Select the numbers in the sorted list of ranks $\ln m + \ln$, $2\,(\ln m + \ln n)$, $\cdots$, $mn\,(\ln m + \ln n)$. The selected numbers induce a doubly linked list V of intervals: $(-\infty, v_1)$, $[v_1, v_2)$, $\cdots$, $[v_{mn}, \infty)$.

We denote these intervals as $[v_r, v_{r+1})$ for $r \in [0, mn]$, where $v_0 = -\infty$, $v_{mn+1} = \infty$, and we abuse the notation slightly to take $[v_0, v_1)$ to mean $(-\infty, v_1)$.

We organize a balanced binary search tree T^V whose nodes correspond to intervals in V.

Use another $O(m^\varepsilon n^\varepsilon)$ input instances to record the frequency f_{ir} that x_i falls into $[v_r, v_{r+1})$. Then, for every $i \in [1, n]$, build an asymptotically optimal binary search tree T_i with respect to the f_{ir}'s on the intervals with positive frequencies. This can be done in $O(m^\varepsilon n^\varepsilon)$ time [6, 8]. The size of T_i is $O(m^\varepsilon n^\varepsilon)$. If a search of T_i reaches a node at depth below $\frac{\varepsilon}{3}\log_2(mn)$ or is unsuccessful, we answer the query by searching T^V which takes $O(\log(mn))$ time

We also need a fast dictionary data structure that can be built in $O(mn)$ time and space. But we defer its description until we explain the need for it in the operation phase.

The total space required is $O(mn + m^\varepsilon n^{1+\varepsilon})$. The total time spent in the training phase is $O(mn(\log m + \log n)^2 + m^\varepsilon n^{1+\varepsilon})$.

3.2 Operation phase

We first give a naive method that is slow to illustrate the overall strategy. Given an input instance $I = (x_1, \cdots, x_n)$, for each $i \in [1, n]$, we search T_i to place x_i in the interval $[v_r, v_{r+1})$ that contains it. For each $r \in [0, mn]$, the entry $[v_r, v_{r+1})$ in V keeps a list N_r of x_i's that fall into it. We sort each N_r in $O(|N_r| \log |N_r|)$ time. Then, we concatenate the sorted N_r's in increasing order of r to form the output sorted list.

Let t_i denote the expected time to query T_i with x_i. Assume that we can prove as in [1] that $\mathrm{E}[|N_r|^2] = O(1)$. Then, sorting each N_r takes only $O(1)$ expected time. Therefore, the total time for processing I is $O(mn + \sum_{i=1}^n t_i)$. This is too slow unless $m = o(\log n)$. The $O(mn)$ term arises from scanning the list V in order to concatenate the sorted N_r's in the right order.. However, at most n of these $mn + 1$ intervals are "useful" because there are only n input numbers. We describe an improvement below.

We maintain a dictionary U that is initially empty. For each $i \in [1, n]$, T_i leads us to the interval $[v_r, v_{r+1})$ that contains x_i. We find v_r in U. If v_r is present in U, we simply add x_i to N_r. Otherwise, we insert v_r to U and initialize N_r to contain x_i alone. After seeing all n input numbers, we find the minimum element in U and then find successors iteratively. This allows us to visit the non-empty N_r's in increasing order of r. So we can concatenate the sorted N_r's in $O(n)$ time. At the end, we delete all elements from U in preparation for sorting the next input instance.

The van Emde Boas tree [9] supports dictionary operations in $O(\log\log N)$ worst-case time each, where N is the size of the universe. It means $O(\log\log(mn))$ time in our case. In the training phase, we construct a van Emde Boas tree with universe V. It uses $O(mn)$ space and can be built in $O(mn)$ time.[5] The asymptotical storage and processing time required by the training phase is unaffected.

In all, the running time is reduced to $O(n \log\log(mn) + \sum_{i=1}^n t_i)$.

3.3 Analysis

We first show that sorting all N_r's takes $O(n)$ expected time.

▶ **Lemma 8.** *It holds with probability at least $1 - 1/n$ that $\mathrm{E}\left[\sum_{r=0}^{mn} |N_r| \log |N_r|\right] = O(n)$.*

[5] The space usage according to the description in [9] is $O(mn \log\log(mn))$, but it can be improved to $O(mn)$ as mentioned in [7].

Proof. We first prove that $\mathrm{E}[|N_r|] = O(1)$ for all $r \in [0, mn]$ are satisfied simultaneously with probability at least $1 - 1/n$.

As a shorthand, let $\gamma = \ln m + \ln n$. Let $I_1, \cdots, I_{m\gamma}$ denote the input instances used in the training phase for building the list V. Let $y_1, y_2, \cdots, y_{mn\gamma}$ denote the sequence formed by concatenating $I_1, \cdots, I_{m\gamma}$ in this order. We adopt the notation that for each $\alpha \in [1, mn\gamma]$, y_α belongs to I_{a_α}, and y_α is drawn from $\mathcal{D}_{q_\alpha, i_\alpha}$.

Fix any distinct index pairs $\alpha, \beta \in [1, mn\gamma]$ such that $y_\alpha \le y_\beta$. For every $q \in [1, \kappa]$, let $\mathcal{J}_\alpha^\beta(q)$ be the set of index triples $\{(a, q, i) : a \in [1, m\gamma], i \in [1, n]\} \setminus \{(a_\alpha, q_\alpha, i_\alpha), (a_\beta, q_\beta, i_\beta)\}$. For any $(a, q, i) \in \mathcal{J}_\alpha^\beta(q)$, let $Y_\alpha^\beta(a, q, i)$ be the indicator random variable such that if $I_a \sim \mathcal{D}_q$ and x_i in I_a falls into $[y_\alpha, y_\beta)$, then $Y_\alpha^\beta(a, q, i) = 1$; otherwise, $Y_\alpha^\beta(a, q, i) = 0$. Define $Y_\alpha^\beta(q) = \sum_{(a,q,i) \in \mathcal{J}_\alpha^\beta(q)} Y_\alpha^\beta(a, q, i)$.

Among the (a, q, i)'s in $\mathcal{J}_\alpha^\beta(q)$, the random variables $Y_\alpha^\beta(a, q, i)$'s are independent from each other. By Chernoff's bound, for any $\mu \in [0, 1]$, $\Pr\left[Y_\alpha^\beta(q) \le (1 - \mu)\mathrm{E}[Y_\alpha^\beta(q)]\right] \le e^{-\mu^2 \mathrm{E}[Y_\alpha^\beta(q)]/2}$. Setting $\mu = \sqrt{35} - 5 \approx 0.9161$ shows that if $\mathrm{E}[Y_\alpha^\beta(q)] > \frac{1}{6-\sqrt{35}}\gamma$, then $Y_\alpha^\beta(q) > \gamma$ with probability at least $1 - m^{-5}n^{-5}$. Since the above statement holds for any fixed choices of q, α and β such that $y_\alpha \le y_\beta$, we can apply the union bound to the $O(\kappa m^2 n^2 (\log m + \log n)^2)$ possible choices of q, α and β and conclude that:

It holds with probability at least $1 - O(m^{-1}n^{-2})$ that for any $q \in [1, \kappa]$ and any $\alpha, \beta \in [1, mn\gamma]$ such that $y_\alpha \le y_\beta$, if $\mathrm{E}[Y_\alpha^\beta(q)] > \frac{1}{6-\sqrt{35}}\gamma$, then $Y_\alpha^\beta(q) > \gamma$.

For every $r \in [0, mn + 1]$, let y_{α_r} denote v_r, where $y_{\alpha_0} = -\infty$ and $y_{\alpha_{mn+1}} = \infty$. Fix a particular $r \in [0, mn]$. By construction, there are γ numbers among $I_1, \cdots, I_{m\gamma}$ that fall in $[v_r, v_{r+1})$, which guarantees the event of $Y_{\alpha_r}^{\alpha_{r+1}}(q) \le \gamma$ for all $q \in [1, \kappa]$. By our previous conclusion, it holds with probability at least $1 - O(m^{-1}n^{-2})$ that $\mathrm{E}[Y_{\alpha_r}^{\alpha_{r+1}}(q)] \le \frac{1}{6-\sqrt{35}}\gamma$ for all $q \in [1, \kappa]$. Let $Y_{\alpha_r}^{\alpha_{r+1}} = \sum_{q=1}^{\kappa} Y_{\alpha_r}^{\alpha_{r+1}}(q)$. It follows that:

$$\mathrm{E}[Y_{\alpha_r}^{\alpha_{r+1}}] = O(\kappa\gamma) \text{ holds with probability at least } 1 - O(m^{-1}n^{-2}). \tag{4}$$

Let X_{ir} be the indicator random variable such that if x_i falls into the interval $[v_r, v_{r+1})$, then $X_{ir} = 1$; otherwise, $X_{ir} = 0$. Then $\sum_{i=1}^n X_{ir} = |N_r|$. Note that $Y_{\alpha_r}^{\alpha_{r+1}}$ counts every x_i's in I_a that falls into $[v_r, v_{r+1})$, except for the two cases of $(a, i) = (a_{\alpha_r}, i_{\alpha_r}) \wedge I_a \sim \mathcal{D}_{q_{\alpha_r}}$ and $(a, i) = (a_{\alpha_{r+1}}, i_{\alpha_{r+1}}) \wedge I_a \sim \mathcal{D}_{q_{\alpha_{r+1}}}$. The random process that generates the input is oblivious of the training phase. Therefore, $\mathrm{E}[Y_{\alpha_r}^{\alpha_{r+1}}]$ is expected to be the same as $\sum_{a=1}^{m\gamma} \sum_{i=1}^n \mathrm{E}[X_{ir}]$, except that the cases of $(a, i) = (a_{\alpha_r}, i_{\alpha_r}) \wedge I_a \sim \mathcal{D}_{q_{\alpha_r}}$ and $(a, i) = (a_{\alpha_{r+1}}, i_{\alpha_{r+1}}) \wedge I_a \sim \mathcal{D}_{q_{\alpha_{r+1}}}$ are excluded from $\mathcal{J}_{\alpha_r}^{\alpha_{r+1}}$, but these two cases are considered in $\sum_{a=1}^{m\gamma} \sum_{i=1}^n \mathrm{E}[X_{ir}]$. Hence,

$$\mathrm{E}[Y_{\alpha_r}^{\alpha_{r+1}}] \ge \left(m\gamma \cdot \sum_{i=1}^n \mathrm{E}[X_{ir}] \right) - 2 = (m\gamma \cdot \mathrm{E}[|N_r|]) - 2.$$

Substituting (4) into the above inequality shows that $\mathrm{E}[|N_r|] = O(1)$. The $O(1)$ bounds on $\mathrm{E}[|N_r|]$ hold for a fixed r with probability at least $1 - O(m^{-1}n^{-2})$. Applying the union bound over $r \in [0, mn]$ establishes the claim that $\mathrm{E}\left[|N_r|\right] = O(1)$ for all $r \in [0, mn]$ are satisfied simultaneously with probability at least $1 - 1/n$. It follows that $\mathrm{E}\left[\max_{r \in [0,mn]} |N_r|\right] = O(1)$ with probability at least $1 - 1/n$.

The expected total time to sort the N_r's is

$$\mathrm{E}\left[\sum_{r=0}^{mn} |N_r| \log |N_r|\right] = \mathrm{E}\left[\sum_{r=0}^{mn} \sum_{i=1}^{n} X_{ir} \log |N_r|\right] \le \sum_{i=1}^{n} \mathrm{E}\left[\max_{r \in [0,mn]} |N_r| \cdot \sum_{r=0}^{mn} X_{ir}\right].$$

Observe that $\sum_{r=0}^{mn} X_{ir} = 1$ because x_i falls into exactly one of the $mn + 1$ intervals. As a result, it holds with probability at least $1 - 1/n$ that $\mathrm{E}\left[\sum_{r=0}^{mn} |N_r| \log |N_r|\right] = O(n)$. ◀

Next, we bound $\sum_{i=1}^{n} t_i$. Let μ_{iqr} denote the probability of $x_i \in [v_r, v_{r+1})$ conditioned on $x_i \sim \mathcal{D}_{q,i}$. Define μ_{ir} to be the the probability of $x_i \in [v_r, v_{r+1})$, and therefore, $\mu_{ir} = \sum_{q=1}^{\kappa} \lambda_q \mu_{iqr}$.

Let H_i^V be the entropy of the distribution of the predecessor of x_i in V. So $H_i^V = \sum_{r=0}^{mn} \mu_{ir} \log(1/\mu_{ir})$. As shown in [1, Lemma 3.4], T_i has an expected search time of

$$O\left(\frac{H_i^V}{\varepsilon}\right) = O\left(\frac{1}{\varepsilon}\sum_{r=0}^{mn} \mu_{ir} \log(1/\mu_{ir})\right)$$
$$= O\left(\frac{1}{\varepsilon}\sum_{r=0}^{mn}\left(\sum_{q=1}^{\kappa}\lambda_q\mu_{iqr}\right)\log\left(\frac{1}{\sum_{q=1}^{\kappa}\lambda_q\mu_{iqr}}\right)\right).$$

Observe that $\log\left(1/\sum_{q=1}^{\kappa}\lambda_q\mu_{iqr}\right) \le \log\frac{1}{\lambda_q\mu_{iqr}}$ for any q. Therefore,

$$H_i^V \le \sum_{q=1}^{\kappa}\left(\sum_{r=0}^{mn}\lambda_q\mu_{iqr}\log\frac{1}{\lambda_q\mu_{iqr}}\right)$$
$$= \sum_{q=1}^{\kappa}\sum_{r=0}^{mn}\lambda_q\mu_{iqr}\log\frac{1}{\lambda_q} + \sum_{q=1}^{\kappa}\sum_{r=0}^{mn}\lambda_q\mu_{iqr}\log\frac{1}{\mu_{iqr}}.$$

Note that $\sum_{r=0}^{mn}\lambda_q\mu_{iqr} = \lambda_q$ as $\sum_{r=0}^{mn}\mu_{iqr} = 1$. Then, $\sum_{q=1}^{\kappa}\sum_{r=0}^{mn}\lambda_q\mu_{iqr}\log(1/\lambda_q) = \sum_{q=1}^{\kappa}\lambda_q\log(1/\lambda_q)$, which is at most $\log\kappa$. Then,

$$\sum_{i=1}^{n} t_i = O\left(\frac{1}{\varepsilon}\sum_{i=1}^{n} H_i^V\right)$$
$$= O\left(\frac{n}{\varepsilon}\log\kappa\right) + O\left(\frac{1}{\varepsilon}\sum_{i=1}^{n}\sum_{q=1}^{\kappa}\sum_{r=0}^{mn}\lambda_q\mu_{iqr}\log\frac{1}{\mu_{iqr}}\right)$$
$$= O\left(\frac{n}{\varepsilon}\log\kappa\right) + O\left(\frac{1}{\varepsilon}\sum_{q=1}^{\kappa}\lambda_q\left(\sum_{i=1}^{n}\sum_{r=0}^{mn}\mu_{iqr}\log\frac{1}{\mu_{iqr}}\right)\right).$$

Let $H_{q,i}^V = \sum_{r=0}^{mn}\mu_{iqr}\log(1/\mu_{iqr})$, the entropy of the distribution of the predecessor of x_i in V conditioned on $x_i \sim \mathcal{D}_{q,i}$. Then, $\sum_{i=1}^{n}\sum_{r=0}^{mn}\mu_{iqr}\log(1/\mu_{iqr}) = \sum_{i=1}^{n} H_{q,i}^V$. By Lemma 5(b) and setting $g = n$, we obtain $\sum_{i=1}^{n} H_{q,i}^V = O(n + H_{\pi,q})$, where $H_{\pi,q}$ is the entropy of $\pi(I)$ conditioned on $I \sim \mathcal{D}_q$. It is well-known that an unconditional entropy is greater than or equal to its conditioned counterpart, so $H_\pi \ge H_{\pi,q}$. Therefore, $\sum_{i=1}^{n} H_{q,i}^V = O(n + H_\pi)$.

Thus, $\sum_{i=1}^{n} t_i = O\left(\frac{n}{\varepsilon}\log\kappa + \frac{1}{\varepsilon}\sum_{q=1}^{\kappa}\lambda_q(n + H_\pi)\right) = O\left((n/\varepsilon)\log\kappa + H_\pi/\varepsilon\right)$.

▶ **Theorem 9.** *For any constant $\varepsilon > 0$, there exists a self-improving sorter of limiting complexity $O\left(n\log\log(mn) + (n/\varepsilon)\log\kappa + H_\pi/\varepsilon\right)$ for any hidden mixture of κ product distribution. The parameter κ is hidden, but an upper bound $m \ge \kappa$ is given. The storage needed is $O(mn + m^\varepsilon n^{1+\varepsilon})$. The training phase processes $O(m(\log m + \log n) + m^\varepsilon n^\varepsilon)$ input instances in $O(mn(\log m + \log n)^2 + m^\varepsilon n^{1+\varepsilon})$ time, and it succeeds with probability at least $1 - 1/n$.*

4 Conclusion

There are several possible directions for future research. One is to extend the hidden classification to allow the x_i's in the same class S_k to be some fixed-degree polynomial in the random parameter z_k. Linear functions in z_k have the nice property that any x_i and

x_j in the same class are linearly related. This helps us to learn the hidden classes. We lose this property in the case of fixed-degree polynomials. Another direction is to improve the limiting complexity in the case of a hidden mixture of product distributions. Can the term $O(n \log\log(mn) + (n/\varepsilon)\log\kappa)$ be reduced? If the upper bound m of κ is not too far off, then $n\log\log(mn) \approx n\log\log\kappa + n\log\log n$, which means that our limiting complexity becomes $O(n\log\log n + (n/\varepsilon)\log\kappa + H_\pi/\varepsilon)$. Although $n\log\log n$ is $o(n\log n)$, it would be nice to eliminate it or reduce it further. It is also unclear whether the factor $\log\kappa$ is necessary.

It would also be interesting to design self-improving algorithms for other problems and possibly other input settings as well.

References

1 N. Ailon, B. Chazelle, K.L. Clarkson, D. Liu, W. Mulzer, and C. Seshadhir. Self-improving algorithms. *SIAM Journal on Computing*, 40(2):350–375, 2011.

2 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.

3 T.M. Cover and J.A. Thomas. *Elements of Information Theory.* Wiley-Interscience, New York, 2nd edition, 2006.

4 M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications.* Springer-Verlag Berlin Heidelberg, 3rd edition, 2008.

5 J.R. Driscoll, N. Sarnak, D.D. Sleator, and R.E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38:86–124, 1989.

6 M.L. Fredman. Two applications of a probabilistic search technique: sorting $X + Y$ and building balanced search trees. In *Proceedings of the 7th Symposium on Theory of Computing*, pages 240–244, 1975.

7 G.F. Italiano and R. Raman. Topics in Data Structures. In M.J. Atallah and M. Blanton, editors, *Algorithms and Theory of Computation Handbook*, pages 5:1–29. Chapman & Hall/CRC, 2nd edition, 2009.

8 K. Mehlhorn. Nearly optimal binary search trees. *Acta Informatica*, 5:287–295, 1975.

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