Privately Learning Thresholds: Closing the Exponential Gap

Haim Kaplan[*]   HAIMK@POST.TAU.AC.IL

Katrina Ligett[†]   KATRINA@CS.HUJI.AC.IL

Yishay Mansour[‡]   MANSOUR.YISHAY@GMAIL.COM

Moni Naor[§]   MONI.NAOR@WEIZMANN.AC.IL

Uri Stemmer[¶]   U@URI.CO.IL

Editors: Jacob Abernethy and Shivani Agarwal

Abstract

We study the sample complexity of learning threshold functions under the constraint of *differential privacy*. It is assumed that each labeled example in the training data is the information of one individual and we would like to come up with a generalizing hypothesis h while guaranteeing differential privacy for the individuals. Intuitively, this means that any single labeled example in the training data should not have a significant effect on the choice of the hypothesis. This problem has received much attention recently; unlike the non-private case, where the sample complexity is *independent of the domain size* and just depends on the desired accuracy and confidence, for private learning the sample complexity must depend on the domain size X (even for approximate differential privacy). Alon et al. (STOC 2019) showed a lower bound of $\Omega(\log^* |X|)$ on the sample complexity and Bun et al. (FOCS 2015) presented an approximate-private learner with sample complexity $\tilde{O}\left(2^{\log^* |X|}\right)$. In this work we reduce this gap significantly, almost settling the sample complexity. We first present a new upper bound (algorithm) of $\tilde{O}\left((\log^* |X|)^2\right)$ on the sample complexity and then present an improved version with sample complexity $\tilde{O}\left((\log^* |X|)^{1.5}\right)$.

Our algorithm is constructed for the related *interior point problem*, where the goal is to find a point between the largest and smallest input elements. It is based on selecting an input-dependent hash function and using it to embed the database into a domain whose size is reduced logarithmically; this results in a new database, an interior point of which can be used to generate an interior point of the original database in a differentially private manner.

1. Introduction

One of the most fundamental tasks in machine learning is that of learning 1-dimensional *threshold functions*. In this task, we are given a collection of examples, called *training data*, where every example is taken from a finite domain $X \subseteq \mathbb{R}$ and is labeled by a fixed (but unknown) *threshold function*. (A threshold function is a binary function that evaluates to 1 on some prefix of the domain.[1]) The goal is to generalize the training data into a hypothesis h that predicts the labels of unseen examples. In this paper we study this problem under the constraint of *differential privacy*. It is assumed that each labeled example in the training data is the information of one individual (e.g., every example might represent blood sugar level, and the label might indicate whether the individual has diabetes). We would like to come up with a generalizing hypothesis h while guaranteeing differential privacy for the individuals. Intuitively, this means that any single labeled example in the training data should not have a significant effect on the choice of h and in particular given the hypothesis it should be hard to distinguish whether an individual's data was used or not.

An important and natural measure for the efficiency of learning algorithms is the amount of data needed to produce a good hypothesis, a.k.a. the *sample complexity*. Without privacy constraints, learning thresholds is easy, and has a 'constant' sample complexity that depends only on the desired level of accuracy and confidence, but not on the domain size (i.e., one can simply set $X = \mathbb{R}$). With differential privacy, however, understanding the sample complexity of this supposedly simple problem has turned out to be quite challenging. Already in the first work on differentially private learning, Kasiviswanathan et al. (2011) presented a generic construction (obtained as a private variant of Occam's Razor (Blumer et al., 1987)) stating that the sample complexity of privately learning threshold functions over a domain X is at most $O(\log |X|)$. That is, unlike the non-private sample complexity, the upper bound of Kasiviswanathan et al. (2011) grows logarithmically with the size of the domain.

This gap between the sample complexity of learning thresholds with or without privacy has received much attention since then. For the case of *pure*-differential privacy (a strong variant of differential privacy), Feldman and Xiao (2015) showed that this gap is unavoidable, and that every pure-private learner for thresholds over a domain X must have sample complexity $\Omega(\log |X|)$. Beimel et al. (2013) showed that the lower bound of Feldman and Xiao (2015) can be circumvented by relaxing the privacy requirement from pure to approximate-differential privacy. Specifically, they presented an approximate-private learner for threshold functions over a domain X with sample complexity $\tilde{O}\left(8^{\log^* |X|}\right)$, a dramatic improvement in asymptotic terms over $\Theta(\log |X|)$. Bun et al. (2015) then presented a different approximate-private learner with improved sample complexity of $\tilde{O}\left(2^{\log^* |X|}\right)$, and another different algorithm with similar sample complexity was presented by Bun et al. (2018). Furthermore, Bun et al. (2015) showed a lower bound of $\Omega(\log^* |X|)$ on the sample complexity of every approximate-private learner for thresholds that outputs a hypothesis that is itself a threshold function (such a learner is called *proper*). Recently, Alon et al. (2019) showed that a lower bound of $\Omega(\log^* |X|)$ holds even for *improper* learners, i.e., for learners whose output hypothesis is not restricted to being a threshold function. To summarize, our current understanding of the task of privately learning thresholds places its sample complexity somewhere between $\Omega(\log^* |X|)$ and $\tilde{O}\left(2^{\log^* |X|}\right)$, a gap which is exponential in $\log^* |X|$, where at least three different algorithms are known with sample complexity $2^{O(\log^* |X|)}$.

---

1. Let $X \subseteq \mathbb{R}$. A threshold function f over X is specified by an element $u \in X$ so that $f(x) = 1$ if $x \le u$ and $f(x) = 0$ for $x > u$.

In this work we reduce this gap significantly, almost settling the sample complexity, and present a new upper bound of $\tilde{O}\left((\log^* |X|)^2\right)$. We then present an improvement of the algorithm with sample complexity $\tilde{O}\left((\log^* |X|)^{1.5}\right)$. While the main goal of the work is to resolve the asymptotic sample complexity of the problem, our algorithm is computationally efficient and may be useful more generally as a technique for shrinking the data domain.

The interior point problem: In order to obtain their results, Bun et al. (2015) presented reductions in both directions between (properly) learning thresholds and the *interior point problem*. Given a database S containing (unlabeled) elements from X, the interior point problem asks for an element of X between the smallest and largest elements in S.

Definition 1 *An algorithm $\mathcal{A}$ solves the* interior point problem *over a domain X with sample complexity n and failure probability β if for every database $S \in X^n$,*

$$\Pr[\min S \leq \mathcal{A}(S) \leq \max S] \geq 1 - \beta,$$

where the probability is taken over the coins of $\mathcal{A}$. We call a solution x with $\min S \leq x \leq \max S$ an interior point *of S. Note that x need not be a member of the database S.*

To see the equivalence between the interior point problem and (properly) learning thresholds, let $S \subseteq X$ be an input for the interior point problem, and construct a labeled database D containing all elements of S where the smallest $|S|/2$ elements are labeled by 1 and the largest $|S|/2$ elements are labeled by 0. Now consider applying a learner for thresholds on the database D, that returns a hypothesis h that is itself a threshold function. The point at which h switches from 1 to 0 must be an interior point of the database S. For the other direction, let D be a labeled database, and suppose that D contains both "a lot" of elements with the label 0 and "a lot" of elements with the label 1 (otherwise either $h \equiv 0$ or $h \equiv 1$ is a good output). Now construct an unlabeled database S containing (say) the largest $|D|/10$ elements in D which are labeled as 1 and the smallest $|D|/10$ elements in D which are labeled as 0. Now consider applying an algorithm for the interior point problem on S to obtain an outcome y, and define the hypothesis $f(x) = \mathbb{1}\{x \leq y\}$. It can be shown that this hypothesis has small error on D, completing the equivalence. In fact, Bun et al. (2015) showed that the following four problems are equivalent (up to constant factors) under differential privacy:

1. The interior point problem.

2. Learning of threshold functions (properly).

3. Distribution learning (with respect to Kolmogorov distance).[2]

4. Query release for threshold functions.[3]

---

2. Distribution learning is a fundamental problem in statistics. Given n i.i.d. samples from an unknown distribution $\mathcal{D}$, the goal is to produce a distribution $\mathcal{D}'$ with small distance to $\mathcal{D}$.

3. Given a set Q of queries $q : X^n \to \mathbb{R}$ the query release problem for Q is to output accurate answers to all queries in Q. In the case of threshold functions, each query is specified by a domain element $x \in X$, and asks for the number of input element that are smaller than x.

Hence, showing new upper or lower bounds on the sample complexity of privately solving the interior point problem immediately results in new upper or lower bounds on the sample complexity of privately learning thresholds (properly), private distribution learning (w.r.t. Kolmogorov distance), and private query release for threshold functions.

1.1. Overview of Our (First) Construction

We design a new algorithm for privately solving the interior point problem with sample complexity $\tilde{O}\left((\log^* |X|)^2\right)$. At a high level, the algorithm works by embedding the input elements from the domain X in a smaller domain of size $\log |X|$, in such a way that guarantees that every interior point of the embedded elements can be (privately) translated into an interior point of the input elements. The algorithm is then applied recursively to identify an interior point of the embedded elements.

We now give an informal overview of the construction. Let X be a totally ordered domain, and consider a binary tree T with $|X|$ leaves, where every leaf is identified with an element of X. We abuse notation and use u_ℓ (for $1 \leq \ell \leq |X|$) to denote both the ℓth smallest element in X and to denote the ℓth left-most leaf of the tree T. Given an input database $S \in X^n$ containing n elements from X, we assign *weights* to the nodes of T in a bottom up manner. First, the weight $w(u_\ell)$ of every leaf u_ℓ is its multiplicity in S. That is, $w(u_\ell) = |\{x \in S : x = u_\ell\}|$. Now define the weight of every node v in T as the sum of the weights of its two children. For simplicity, we will assume throughout the introduction that every leaf has weight either 0 or 1 (i.e., we assume that no element appears twice in S; this is not true as we recurse).

We now explain how to generate the input database to the next recursive call. That is, how we embed the input elements in a domain of size $\log |X|$. To that end, suppose that we have generated a *path* π from the root to a leaf u_π with positive weight (by our simplifying assumption this leaf has weight 1). Had we been able to compute such a path *in a private manner* then our task would be completed, since if $w(u_\pi) > 0$ then $u_\pi \in S$ and hence u_π is an interior point of S, and there was no need to apply the recursion. Therefore we do not argue that we can release the path while maintaining differential privacy. Nevertheless, assume (for now) that such a path were given to us "for free", and instead of ending the computation by returning the leaf at the end of this path, we use it in order to construct the input database to the next recursive call.

Let v be a node in the path π, let v_{next} be the next node in π, and let v_{other} denote the other child of v in the tree T. We say that m input points *fall off* the path π at the node v if $w(v_{\text{other}}) = m$. Input points that *fall off* the path π at the node v are input points that belong to the subtree rooted at v, but not to the subtree rooted at the next node in π. As π starts at a node with weight n (the root of the tree) and ends at a leaf with weight 1 (the leaf u_π), we get that overall $n - 1$ input points fall off the path π.

One can think of the path as acting as a hash function of the domain into a range that is logarithmic in its size, where a point is mapped to the level where it falls off the path. The key property we need is that adjacent databases will be embedded to adjacent databases. Initialize an empty database D (which will be the input database to the next recursive call). We add elements to D by following the path π from the root down. If m points fall off the path at level ℓ of the tree, then we add m copies of ℓ to the database D. Observe that D contains elements from $[\log |X|]$, i.e., the domain size is reduced logarithmically.

Now suppose that (by recursion) we obtained an interior point ℓ^* of the database D, and let $v_\pi^{\ell^*}$ denote the node at level ℓ^* in the path π. Furthermore, let us assume for simplicity that there are in

fact "many" points in D that are bigger than ℓ^* and "many" points which are smaller than ℓ^*, and so ℓ^* is a "deep" interior point of D.[4] This means that "many" elements from S fall off the path π before level ℓ^* and that "many" elements fall off after level ℓ^* of the tree. Note that since "many" elements from S fall off π after level ℓ^*, we get that the weight of $v_\pi^{\ell^*}$ (the node at level ℓ^* of the path π) is "large".

Now let v^* be a node at level ℓ^* of the tree that is chosen according to its weight in a differentially private manner (such a "heavy" node exists since $v_\pi^{\ell^*}$ is "heavy", and can be privately identified using standard *stability based* techniques). Now, since "many" elements from S fall of the path π before level ℓ^*, it cannot be the case that all of the points in S belong to the subtree rooted at v^*. Indeed, if that were the case then the path π must go through v^*, i.e., $v^* = v_\pi^{\ell^*}$, but then all the elements of S that fall off before level ℓ^* cannot belong to the subtree rooted at v^*. (For simplicity we assumed here that these elements fall *strictly* before level ℓ^*; in the actual algorithm we handle such a case differently.)

The final output is then chosen from one of two descendants of v^*: the left-most and right-most leaves of the sub-tree rooted at v^*, denoted as v_{left} and v_{right}, respectively.[5] To see that at least one of these leafs is a good output, observe that since v^* has positive weight, then some of the elements of S must belong to the subtree rooted at v^*. That is, the database S contains elements between v_{left} and v_{right}. In addition, since the subtree rooted at v^* does not contain *all* elements of S, then either S contains elements which are bigger than v_{right}, in which case v_{right} is an interior point of S, or S contains elements which are smaller than v_{left}, in which case v_{left} is an interior point of S.

Selecting the path. In the above description we assumed that the path π is given to us "for free". While the path π contains sensitive information and cannot be released privately, we show that it is possible to select a path π randomly in such a way that guarantees the following condition. Let S and S' be two neighboring databases, and let $\mathcal{D}$ denote the distribution on databases that results from sampling a path π in the execution on S and embedding the points from S using π. Similarly let $\mathcal{D}'$ denote this distribution w.r.t. S'. Then we show that these two distributions are "close" in the sense that for any database D in the support of $\mathcal{D}$ (with non-negligible probability mass) there exists a *neighboring* database D' with roughly the same probability mass under $\mathcal{D}'$. So, intuitively, while the paths π that appear in the executions on S and on S' might be very different, we show that the resulting distributions on the databases for the next recursive call are "similar". This can be formalized to show that the algorithm satisfies differential privacy.

We obtain the following theorem, which we present in Section 3.

Theorem 2 *Let X be a totally ordered domain. There exists an (ε, δ)-differentially private algorithm that solves the interior point problem on X with success probability 9/10 and sample complexity $n = \tilde{O}\left(\frac{1}{\varepsilon} \cdot \log(\frac{1}{\delta}) \cdot (\log^* |X|)^2\right)$.*

In Section A we present an improved analysis of (a variant of) this algorithm, which results in a sample complexity of $n = \tilde{O}\left(\frac{1}{\varepsilon} \cdot \log^{1.5}(\frac{1}{\delta}) \cdot (\log^* |X|)^{1.5}\right)$. Using the equivalence of Bun et al. (2015), our new algorithm for the interior point problem results in new algorithms for privately learning threshold functions, for private distribution learning (w.r.t. Kolmogorov distance), and for private query release of threshold functions, all with sample complexity proportional to

---

4. Here "many" should be thought of as $\approx \frac{1}{\varepsilon} \log \frac{1}{\delta}$; in the actual algorithm we ensure that these conditions indeed hold by 'trimming' these elements from the database before the recursive call.

5. In the actual algorithm we also consider two additional descendants of v^*.

$\tilde{O}\left((\log^* |X|)^{1.5}\right)$. Previous algorithms for these tasks had sample complexity proportional to $\tilde{O}\left(2^{\log^* |X|}\right)$.

We stress that the fundamental difference between our work and the previous works of Beimel et al. (2013); Bun et al. (2015, 2018) is that we construct the input to the recursive call using a path which is *not* computed in a private manner. The main conceptual new idea is that this can be done while still guaranteeing that the algorithm itself is differentially private.

2. Preliminaries

An algorithm operating on databases is said to preserve differential privacy if changing a single record of its input database does not significantly change the output distribution of the algorithm. Intuitively, this means that whatever is learned about an individual could also be learned with her data arbitrarily modified (or without her data). Formally:

Definition 3 (Dwork et al. (2006b,a)) *Two databases are called* neighboring *if they differ in a single entry. A randomized algorithm $\mathcal{A}$ is (ε, δ)-differentially private *if for every two neighboring databases S, S' and for any event T,*

$$\Pr[\mathcal{A}(S) \in T] \leq e^\varepsilon \cdot \Pr[\mathcal{A}(S') \in T] + \delta.$$

If $\delta == 0$ the type of privacy guarantee is pure *and if $0 < \delta < 1$ it is* approximate.

One of the key reasons differential privacy is such a powerful notion is the composition properties it enjoys. In particular, in "advanced composition" the adaptive application of k mechanisms, each of which is (ε, δ)-differentially private, satisfies $\approx (\varepsilon\sqrt{k}, \delta k)$-differential privacy (Dwork et al., 2010). For background on differential privacy see Dwork and Roth (2014) or Vadhan (2016).

2.1. The Exponential and Choosing Mechanisms

Let X^* denote the set of all finite databases over a domain X. A quality function $q : X^* \times \mathcal{F} \to \mathbb{N}$ defines an *optimization problem* over the domain X and a finite solution set $\mathcal{F}$: Given a database $S \in X^*$, choose $f \in \mathcal{F}$ that (approximately) maximizes $q(S, f)$. We say that the function q has *sensitivity* Δ if for all neighboring databases S and S' and for all $f \in \mathcal{F}$ we have $|q(S, f) - q(S', f)| \leq \Delta$.

The Exponential Mechanism of McSherry and Talwar (2007) solves such optimization problems by choosing a random solution where the probability of outputting any solution f increases exponentially with its quality $q(S, f)$. Specifically, it outputs each $f \in \mathcal{F}$ with probability proportional to $\exp(\varepsilon \cdot q(S, f)/(2\Delta))$. The privacy and utility of the mechanism are expressed as:

Lemma 4 (McSherry and Talwar (2007)) *The Exponential Mechanism is $(\varepsilon, 0)$-differentially private. Let q be a quality function with sensitivity at most 1. Fix a database $S \in X^n$ and let $\text{OPT} = \max_{f \in \mathcal{F}}\{q(S, f)\}$. With probability at least $(1 - \beta)$, the Exponential Mechanism outputs a solution f with quality $q(S, f) \geq \text{OPT} - \frac{2}{\varepsilon}\ln\left(\frac{|\mathcal{F}|}{\beta}\right)$.*

Note that the quality of the solution guaranteed by the Exponential Mechanism decreases with $\log |\mathcal{F}|$. For a sub family of low-sensitivity functions, called *bounded-growth* functions, this can

be avoided by using an (ε, δ)-differentially private variant of the Exponential Mechanism called the *Choosing Mechanism*, introduced by Beimel et al. (2013). A quality function with sensitivity at most 1 is of *k-bounded-growth* if adding an element to a database can increase (by 1) the score of at most k solutions, without changing the scores of other solutions. Specifically, it holds that

1. $q(\emptyset, f) = 0$ for all $f \in \mathcal{F}$,

2. If $S_2 = S_1 \cup \{x\}$, then $q(S_1, f) + 1 \geq q(S_2, f) \geq q(S_1, f)$ for all $f \in \mathcal{F}$, and

3. There are at most k values of f for which $q(S_2, f) = q(S_1, f) + 1$.

Example 1 *Consider a case where $\mathcal{F} = X$ and $q(S, f) = |\{x \in S : x = f\}|$. That is, the quality of a solution (=domain element) f is its multiplicity in the database S. Note that with this quality function we are aiming to identify an element $f \in X$ with large multiplicity in S. Observe that this function is 1-bounded growth.*

The Choosing Mechanism is a differentially private algorithm for approximately solving bounded-growth choice problems. The following lemma specifies its privacy and utility guarantees.

Lemma 5 (Beimel et al. (2013); Bun et al. (2015)) *Let $\delta > 0$, and $0 < \varepsilon \leq 2$. The Choosing Mechanism is (ε, δ)-differentially private. Let the Choosing Mechanism be executed on a k-bounded-growth quality function, and on a database S containing n elements. Denote $\mathrm{OPT} = \max_{f \in \mathcal{F}}\{q(S, f)\}$. With probability at least $(1 - \beta)$, the Choosing Mechanism outputs a solution f with quality $q(S, f) \geq \mathrm{OPT} - \frac{16}{\varepsilon} \ln(\frac{4kn}{\beta \varepsilon \delta})$.*

3. Algorithm `TreeLog` and its Analysis

In this section we present algorithm `TreeLog`, which privately solves the interior point problem with sample complexity $\tilde{O}\left((\log^* |X|)^2\right)$. Consider algorithm `TreeLog`, described in Algorithm 1. We analyze its privacy properties in Section 3.1 and analyze its utility properties in Section 3.2. For an illustration of the algorithm see Figure 1.

3.1. Privacy Analysis

First, observe that it suffices to analyze the algorithm without Step 2. The reason is that for any two neighboring databases S, S' we have that the multisets $\hat{S}, \hat{S}'$ containing the middle elements of S, S', respectively, are neighboring multisets (this technique was also used by Bun et al. (2015)). So, for the privacy analysis, we will assume that $\hat{S}$ contains all elements of S.

Fix two neighboring databases S and $S' = S \cup \{x'\}$, and consider the (top-level call of the) execution of `TreeLog` on S and on S'. We use $w_S(v)$ and $w_{S'}(v)$ to denote the weight of a node v during the two executions, respectively. Also, let $P(S)$ and $P(S')$ denote the set of possible paths that can be obtained in Step 4 of the execution on S and on S', respectively. That is, $P(S)$ denotes the support of the distribution on paths defined in Step 4 of the (top-level call of the) execution on S. For a path π we write $\Pr_S[\pi]$ and $\Pr_{S'}[\pi]$ to denote the probability of obtaining this path in the executions on S and on S', respectively. We make the following observations.

Observation 6 $|P(S)| \leq |S|$ *and* $|P(S')| \leq |S'|$.

Algorithm 1 `TreeLog`

Input: Parameters ε, δ and a database $S \in X^n$ where X is a totally ordered domain. We assume that $|X|$ is a power of 2 (otherwise extend X to the closest power of 2). Set trimming parameter $t = \Theta\left(\frac{1}{\varepsilon}\log\frac{1}{\delta}\right)$.

1. If $|X| = O(1)$ then use the Exponential Mechanism to return $y \in X$ with large quality

$$q(S, y) = \min\left\{ \, |\{x \in S : x \leq y\}|, \; |\{x \in S : x \geq y\}| \, \right\}.$$

2. Sort S and let $\hat{S} \in X^{n-2t}$ be a database containing all elements of S except for the t largest and t smallest elements.

3. Let T be a complete binary tree with $|X|$ leaves that correspond to elements of X. A leaf u has weight $w(u) = |\{x \in \hat{S} : x = u\}|$. A node v has weight $w(v)$ that equals the sum of the weights of its children.

4. Sample a path π in T, starting from the root and constructed by the following process:

   (a) Let v be the current node in the path.

   (b) If v is a leaf, or if $w(v) \leq t$, then v is the last node in π.

   (c) Else, let v_0, v_1 be the two children of v in T. If one of them has weight 0 then proceed to the other child. Otherwise, proceed to v_b with probability proportional to $\exp(\varepsilon \cdot w(v_b))$.

5. Initialize $D = \emptyset$. Add elements to D by following the path π starting from the root:

   (a) If $|D| \geq n - 3t$ then goto Step 6.

   (b) Let v be the current node in the path, and let ℓ denote its level in T (the root is in level 0 and the leaves are in level $\log|X|$).

   (c) If v is the last node in π then add $(n - 3t - |D|)$ copies of ℓ to D and goto Step 6.

   (d) Else, let v_{next} be the next node in π, and let v_{other} be the other child of v in T. Add $\min\{w(v_{\text{other}}), n - 3t - |D|\}$ copies of ℓ to D, and goto Step 5a with v_{next} as the current node.

6. Execute `TreeLog` recursively on D and let ℓ^* denote the returned outcome.

7. Use the Choosing Mechanism with privacy parameters ε, δ to choose a node v^* at level ℓ^* of T with large weight $w(v^*)$.

8. Let v_{left} and v_{right} be the left-most and right-most leaves, respectively, of the sub-tree rooted at v^*. Also let $v_{\text{inner-left}}$ be the right-most leaf of the sub-tree rooted at the left child of v^*, and let $v_{\text{inner-right}}$ be the left-most leaf of the sub-tree rooted at the right child of v^*.

9. Use the Exponential Mechanism with privacy parameter ε to return $y \in \{v_{\text{left}}, v_{\text{right}}, v_{\text{inner-left}}, v_{\text{inner-right}}\}$ with large quality

$$q(S, y) = \min\left\{ \, |\{x \in S : x \leq y\}|, \; |\{x \in S : x \geq y\}| \, \right\}.$$

This is because paths only end in nodes with positive weight, and because if $\pi_1 \neq \pi_2$ are both in $P(S)$ then π_1 is not a prefix of π_2 and vice verse.

Observation 7 *Let $\pi \in P(S)$, and let v_π denote the last node in π. There exists either one or two paths in $P(S')$ such that π is their prefix.*

This is because if v_π is reachable from the root during the execution on S then it is also reachable during the execution on S'. When reaching v_π in the execution on S', the path ends in v_π if $w_{S'}(v_\pi) \leq t$, and otherwise the path continues and can split at most once into two different paths.

Definition 8 *A path $\pi \in P(S)$ is* weak *for S if $\Pr_S[\pi] \leq \delta$. Otherwise, the path is* strong *for S. We use similar definitions w.r.t. S'.*

Observation 9 *Let $\pi' \in P(S')$. Either there is a path $\pi \in P(S)$ such that π is a prefix of π', or else π' is weak for S'.*

Proof Assume that no path in $P(S)$ is a prefix of π'. Let v_{last} denote the last node in the path π' that appears in a path of $P(S)$, and let v_{break} denote the next node in π'. Note that v_{break} must exist because otherwise $\pi' \in P(S)$. Since no path in $P(S)$ is a prefix of π', we get that any path in $P(S)$ that reaches v_{last} does not stop in v_{last}, and hence, $w_S(v_{\text{last}}) > t$ and therefore $w_{S'}(v_{\text{last}}) > t$. Moreover, since $w_S(v_{\text{last}}) > t$ and since v_{break} is not on a path of $P(S)$, we have that $w_S(v_{\text{break}}) = 0$ and hence $w_{S'}(v_{\text{break}}) = 1$. So $w_{S'}(v_{\text{break}}) = 1$ and $w_{S'}(v_{\text{last}}) > t$ and therefore the probability of proceeding from v_{last} to v_{break} is at most δ. ∎

Claim 10 *Let $\pi \in P(S)$ be a* strong *path, and let $\Pi' \subseteq P(S')$ be a subset containing all paths in $P(S')$ such that π is their prefix. Then $\Pr_S[\pi] \leq e^{3\varepsilon \cdot \log n} \cdot \Pr_{S'}[\Pi']$.*

Proof Let v_0 denote the first node in π (the root of the tree T) and let v_π denote the last node in π. Also let $v_1, v_2, \ldots, v_k$ denote all the nodes in the path π (different from v_π) such that both of
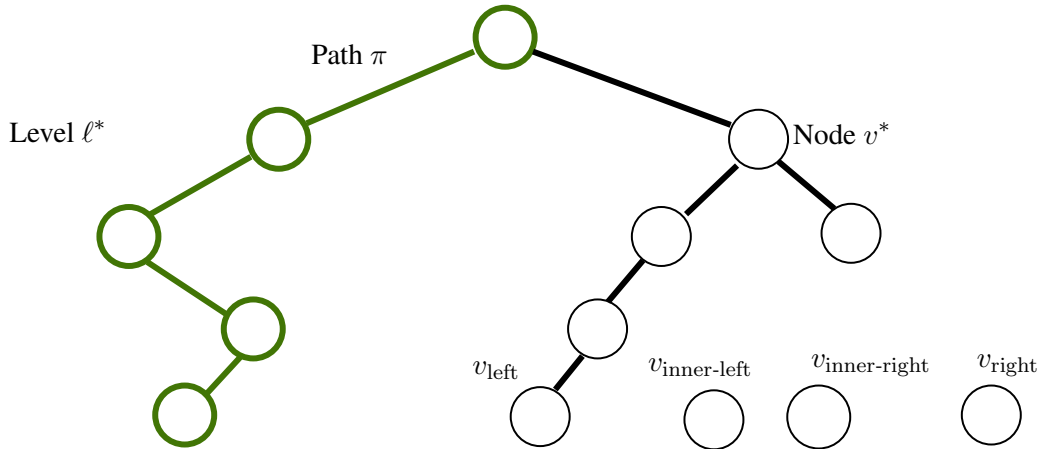

Figure 1: An illustration of the objects used by Algorithm `TreeLog`.

their children in T have positive weight in the execution on S (these are the nodes from which the construction of the path in Step 4 is done randomly). For every such node v_i let $v_{i,\text{next}}$ denote its following node in π, and let $v_{i,\text{other}}$ denote v_i's other child in T. Note that $v_{k,\text{next}} = v_\pi$. So π can be written as

$$\pi := v_0 \rightsquigarrow v_1 \to v_{1,\text{next}} \rightsquigarrow v_2 \to v_{2,\text{next}} \rightsquigarrow \cdots \rightsquigarrow v_k \to v_\pi.$$

With these notations we have that

$$\mathrm{Pr}_S[\pi] = \frac{e^{\varepsilon \cdot w_S(v_{1,\text{next}})}}{e^{\varepsilon \cdot w_S(v_{1,\text{next}})} + e^{\varepsilon \cdot w_S(v_{1,\text{other}})}} \cdot \frac{e^{\varepsilon \cdot w_S(v_{2,\text{next}})}}{e^{\varepsilon \cdot w_S(v_{2,\text{next}})} + e^{\varepsilon \cdot w_S(v_{2,\text{other}})}} \cdots \frac{e^{\varepsilon \cdot w_S(v_{k,\text{next}})}}{e^{\varepsilon \cdot w_S(v_{k,\text{next}})} + e^{\varepsilon \cdot w_S(v_{k,\text{other}})}}.$$

During the execution on S', some of these weights could be larger by (at most) 1. In addition, there might be another (at most one) node along the path π such that the weight of both its children is positive, but in that case the weight of one of its children is exactly 1 (since during the execution on S the weight of this node is 0). Hence, the probability of proceeding from this node along π (during the execution on S') is at least $(1 - \delta)$. Therefore,

$$\mathrm{Pr}_{S'}[\Pi] \geq (1 - \delta) \cdot \frac{e^{\varepsilon \cdot w_S(v_{1,\text{next}})}}{e^{\varepsilon \cdot [w_S(v_{1,\text{next}})+1]} + e^{\varepsilon \cdot [w_S(v_{1,\text{other}})+1]}} \cdots \frac{e^{\varepsilon \cdot w_S(v_{k,\text{next}})}}{e^{\varepsilon \cdot [w_S(v_{k,\text{next}})+1]} + e^{\varepsilon \cdot [w_S(v_{k,\text{other}})+1]}}.$$
(1)

Because the path π is *strong*, for every i it holds that $w_S(v_{i,\text{next}}) > w_S(v_{i,\text{other}}) - \frac{1}{\varepsilon} \log(\frac{1}{\delta})$, as otherwise the probability of proceeding from v_i to $v_{i,\text{next}}$ would be at most δ and the path would not be strong. In addition, for every i such that $w_S(v_{i,\text{next}}) > w_S(v_{i,\text{other}}) + \frac{1}{\varepsilon} \log(\frac{1}{\delta})$ we have that the term corresponding to i in Inequality (1) is at least $(1 - \delta)$. Finally, note that there could be at most $2 \log n$ indices i such that $|w_S(v_{i,\text{next}}) - w_S(v_{i,\text{other}})| \leq \frac{1}{\varepsilon} \log(\frac{1}{\delta})$. This is because the path π ends when reaching a node of weight at most $t = \Theta(\frac{1}{\varepsilon} \log \frac{1}{\delta})$, and every step in which $|w_S(v_{i,\text{next}}) - w_S(v_{i,\text{other}})| \leq \frac{1}{\varepsilon} \log(\frac{1}{\delta})$ decreases the weight of the next node by at least $1/3$ if $t \geq \frac{2}{\varepsilon} \log(\frac{1}{\delta})$. Putting these observations together, and using the fact that that $k \leq n$ (which follows from Observation 6), we get that

$$\mathrm{Pr}_{S'}[\Pi] \geq (1 - \delta) \cdot (1 - \delta)^n \cdot e^{-2\varepsilon \log n} \cdot \frac{e^{\varepsilon \cdot w_S(v_{1,\text{next}})}}{e^{\varepsilon \cdot w_S(v_{1,\text{next}})} + e^{\varepsilon \cdot w_S(v_{1,\text{other}})}} \cdots \frac{e^{\varepsilon \cdot w_S(v_{k,\text{next}})}}{e^{\varepsilon \cdot w_S(v_{k,\text{next}})} + e^{\varepsilon \cdot w_S(v_{k,\text{other}})}}$$

$$= (1 - \delta) \cdot (1 - \delta)^n \cdot e^{-2\varepsilon \log n} \cdot \mathrm{Pr}_S[\pi]$$

$$\geq e^{-4\delta n} \cdot e^{-2\varepsilon \log n} \cdot \mathrm{Pr}_S[\pi]$$

$$\geq e^{-3\varepsilon \log n} \cdot \mathrm{Pr}_S[\pi],$$

where the last inequality holds when $\delta \leq \frac{\varepsilon}{4n}$. ∎

Lemma 11 *Algorithm* `TreeLog` *with N recursive calls is $(5\varepsilon N \log n, 3\delta n N e^{3\varepsilon N \log n})$ differentially private.*

Proof Assume towards induction that the lemma holds whenever the algorithm performs at most $N - 1$ recursive calls, and let X be a domain that causes the algorithm to perform N recursive calls. Let $S \in X^n$ and let $S' = S \cup \{x'\}$. Fix a set $F \subseteq X$ of possible outcomes. We will show that $\mathrm{Pr}[\text{\texttt{TreeLog}}(S) \in F] \leq e^{5\varepsilon N \log n} \cdot \mathrm{Pr}[\text{\texttt{TreeLog}}(S') \in F] + 3\delta n N e^{3\varepsilon N \log n}$. The other direction follows from similar arguments.

Let $\pi \in P(S)$, and let $\Pi' \subseteq P(S')$ be a subset containing all paths in $P(S')$ such that π is their prefix. We first show that the following inequality holds:

$$\Pr[\texttt{TreeLog}(S) \in F | \pi] \leq e^{5\varepsilon(N-1)\log(n)+2\varepsilon} \cdot \Pr[\texttt{TreeLog}(S') \in F | \Pi'] + 3\delta n(N-1)e^{3\varepsilon(N-1)\log n} + 2\delta. \tag{2}$$

To see this, let D and D' denote the multisets constructed in Step 5 of the execution on S and on S' respectively. Now note that when fixing π in the execution on S, and any $\pi' \in \Pi'$ in the execution on S', the resulting multisets D, D' are neighboring. Specifically, if $\pi = \pi'$ then this is clear (since the number of elements that fall off π in S and S' differ by at most one in a single position). Now suppose that $\pi \neq \pi'$ (but π is still a prefix of π'). Let v_π denote the last node in π. Since π ends in v_π but π' does not, we have that $w_S(v_\pi) = t$ and $w_{S'}(v_\pi) = t + 1$. This means that the part of D generated before we reach v_π when running on S is identical to the part of D' generated before we reach v_π when running on S', because the weight of all the nodes that fall off π and π' before v_π is the same in S and in S'. Since $w_{S'}(v_\pi) = t + 1$ and since we stop adding elements to D once the remaining weight is t, we get that exactly one more element will be added to D'.

Therefore, once we fix π and π', Step 6 satisfies $(5\varepsilon(N - 1)\log n, 3\delta n(N - 1)e^{3\varepsilon(N-1)\log n})$-differential privacy by the induction assumption. In Steps 7 and 9 we apply the Choosing Mechanism and the Exponential Mechanism, each of which satisfies (ε, δ)-differential privacy. Inequality 2 follows from (simple) composition. The proof now follows by the following inequality, where we write $\texttt{TL}$ as a shorthand for $\texttt{TreeLog}$.

$$\Pr[\texttt{TreeLog}(S) \in F] = \sum_{\pi \in P(S)} \Pr_S[\pi] \cdot \Pr[\texttt{TreeLog}(S) \in F | \pi]$$

$$\leq \sum_{\text{weak } \pi} \Pr_S[\pi] \cdot \Pr[\texttt{TreeLog}(S) \in F | \pi] + \sum_{\text{strong } \pi} \Pr_S[\pi] \cdot \Pr[\texttt{TreeLog}(S) \in F | \pi]$$

$$\leq n\delta + \sum_{\text{strong } \pi} \Pr_S[\pi] \cdot \Pr[\texttt{TreeLog}(S) \in F | \pi]$$

$$\leq n\delta + \sum_{\text{strong } \pi} e^{3\varepsilon \log n} \Pr_{S'}[\Pi'] \left(e^{5\varepsilon(N-1)\log(n)+2\varepsilon} \Pr[\texttt{TL}(S') \in F | \Pi'] + 3\delta n(N-1)e^{3\varepsilon(N-1)\log n} + 2\delta \right)$$

$$\leq n\delta + 3\delta n(N-1)e^{3\varepsilon N \log n} + 2\delta e^{3\varepsilon N \log n} + \sum_{\text{strong } \pi} e^{5\varepsilon N \log(n)} \cdot \Pr_{S'}[\Pi'] \cdot \Pr[\texttt{TreeLog}(S') \in F | \Pi']$$

$$\leq 3\delta n N e^{3\varepsilon N \log n} + e^{5\varepsilon N \log(n)} \cdot \Pr[\texttt{TreeLog}(S') \in F],$$

where the third inequality follows from Claim 10 and from Inequality 2. ∎

Recall that each recursive call shrinks the domain size logarithmically, and hence, on a database $S \in X^*$ algorithm $\texttt{TreeLog}$ preforms at most $\log^* |X|$ recursive calls. The following lemma, therefore, follows directly from Lemma 11.

Lemma 12 *Let algorithm* $\texttt{TreeLog}$ *be executed on databases containing n elements from a domain X. The algorithm is $(5\varepsilon \log^* |X| \log n, 3\delta n \log^* |X| e^{3\varepsilon \log^* |X| \log n})$-differentially private.*

3.2. Utility Analysis

Given a database $S \subseteq X$ over a totally ordered domain X and a point $y \in X$, we say that y is an interior point of S with score Δ if there are at least Δ elements in S that are greater or equal to y

and there are at least Δ elements in S that are smaller or equal to y. We say that y is an interior point, without mentioning the score, to mean that it is an interior point with score at least 1.

Lemma 13 *Let* `TreeLog` *be executed on a database* $S \in X^n$ *of size* $n \geq O\left(\frac{\log^* |X|}{\varepsilon} \log \frac{1}{\delta}\right)$. *With probability at least* $1 - O\left(\delta \log^* |X|\right)$, *the algorithm returns an interior point of S with score at least* $\Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$.

Proof The utility analysis is by induction on the number of recursive calls. Specifically, let N denote the number of recursive calls, and let S_i denote the input to the ith recursive call (we identify the top-level call with the index 0, so $S_0 = S$, and the deepest call with the index N). We denote $n_i = |S_i|$. We will show that if the $(i + 1)$th call returns an interior point of S_{i+1} then, w.h.p., so does the ith call.

For the base case, observe that every iteration of `TreeLog` shrinks the size of the domain logarithmically, and hence, $N = O(\log^* |X|)$. In contrast, the database size only decreases additively in each recursive call, by a factor of $3t$. Therefore, when $n \geq O(t \cdot \log^* |X|) = O(\frac{\log^* |X|}{\varepsilon} \log \frac{1}{\delta})$ the last recursive call (which halts in Step 1) is performed with a database of size $\Omega(\frac{1}{\varepsilon} \log \frac{1}{\delta})$. In this case, by Lemma 4 (Exponential mechanism with $\beta = \delta$), the last recursive call returns an interior point of S_N with score at least $\Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$ with probability at least $(1 - \delta)$.

Now consider the ith call. Let $\hat{S}_i$ be the database of size $n_i - 2t$ constructed in Step 2, let π be the path selected in Step 4, and let D be the database constructed in Step 5. Assume that by induction, with probability at least $(1 - 2\delta(N - (i+1)))$, the recursive call in Step 6 returned a point ℓ^* that is an interior point of D with score at least $\Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$. This means that at least $\Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$ points from $\hat{S}_i$ fall off the path π on or before level ℓ^* of the tree T, and at least $\Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$ points from $\hat{S}_i$ fall off on or after level ℓ^*.

Let $v_\pi^{\ell^*}$ denote the node in π at level ℓ^* of the tree. Since at least $\Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$ points from $\hat{S}_i$ fall off from π on or after level ℓ^*, we get that the weight of $v_\pi^{\ell^*}$ is at least $\Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$. Hence, by Lemma 5 (Choosing mechanism with $\beta = \delta$), the node v^* identified in Step 7 also has weight at least $\Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$ with probability at least $(1 - \delta)$. Suppose that the weight of v^* is strictly less than $|\hat{S}_i|$. Then either v_{left} or v_{right} is an interior point of $\hat{S}_i$, and hence, it is an interior point of S_i with score at least $\Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$. In this case, with probability at least $(1 - \delta)$, the Exponential Mechanism in Step 9 identifies an interior point of S_i with score at least $\Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$.

Now suppose that v^* has weight exactly $|\hat{S}_i|$, which means that $v^* = v_\pi^{\ell^*}$ (since in this case the path π must go through v^*). Recall that at least $\Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$ points from $\hat{S}_i$ fall off from π on or before level ℓ^*. Since $v^* = v_\pi^{\ell^*}$ has weight $|\hat{S}_i|$, this means that all these points fall of π exactly on level ℓ^*. So both children of v^* have positive weight. This means that either $v_{\text{inner-left}}$ of $v_{\text{inner-right}}$ is an interior point of $\hat{S}_i$, and hence, it is an interior point of S_i with score at least $\Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$. As we argued in the base case of the induction, with probability at least $(1 - \delta)$, the Exponential Mechanism in Step 9 identifies an interior point of S_i with score at least $\Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$.

Overall, with probability at least $(1 - 2\delta(N - i))$, the ith call returns an interior point of S_i with score at least $\Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$. ∎

Theorem 2 now follows by combining Lemma 12 and Lemma 13. In Appendix A we reduce the sample complexity of algorithm `TreeLog` from $\tilde{O}\left((\log^* |X|)^2\right)$ to $\tilde{O}\left((\log^* |X|)^{1.5}\right)$.

Acknowledgments

References

Noga Alon, Roi Livni, Maryanthe Malliaris, and Shay Moran. Private PAC learning implies finite littlestone dimension. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019.*, pages 852–860, 2019. doi: 10.1145/3313276.3316312. URL https://doi.org/10.1145/3313276.3316312.

Amos Beimel, Kobbi Nissim, and Uri Stemmer. Private learning and sanitization: Pure vs. approximate differential privacy. In *APPROX-RANDOM*, volume 8096 of *Lecture Notes in Computer Science*, pages 363–378. Springer, 2013.

Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Occam's razor. *Inf. Process. Lett.*, 24(6):377–380, 1987. doi: 10.1016/0020-0190(87)90114-1. URL https://doi.org/10.1016/0020-0190(87)90114-1.

Mark Bun, Kobbi Nissim, Uri Stemmer, and Salil P. Vadhan. Differentially private release and learning of threshold functions. In *FOCS*, pages 634–649, 2015.

Mark Bun, Cynthia Dwork, Guy N. Rothblum, and Thomas Steinke. Composable and versatile privacy via truncated cdp. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 74–86, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5559-9. doi: 10.1145/3188745.3188946. URL http://doi.acm.org/10.1145/3188745.3188946.

Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014. doi: 10.1561/0400000042. URL http://dx.doi.org/10.1561/0400000042.

Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 486–503. Springer, 2006a.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography, Third Theory of Cryptography Conference*, pages 265–284, 2006b.

Cynthia Dwork, Moni Naor, Omer Reingold, Guy N. Rothblum, and Salil P. Vadhan. On the complexity of differentially private data release: efficient algorithms and hardness results. In Michael Mitzenmacher, editor, *STOC*, pages 381–390. ACM, 2009.

Cynthia Dwork, Guy N. Rothblum, and Salil P. Vadhan. Boosting and differential privacy. In *FOCS*, pages 51–60. IEEE Computer Society, 2010.

Vitaly Feldman and David Xiao. Sample complexity bounds on differentially private learning via communication complexity. *SIAM J. Comput.*, 44(6):1740–1764, 2015. doi: 10.1137/140991844. URL http://dx.doi.org/10.1137/140991844.

Shiva Prasad Kasiviswanathan, Homin K. Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? *SIAM J. Comput.*, 40(3):793–826, 2011.

Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '07, pages 94–103, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-3010-9. doi: 10.1109/FOCS.2007.41. URL http://dx.doi.org/10.1109/FOCS.2007.41.

Salil Vadhan. *The Complexity of Differential Privacy*. 2016.

Appendix A. Decomposing Algorithm `TreeLog`

In this section we reduce the sample complexity of algorithm `TreeLog` from $\tilde{O}\left(\left(\log^* |X|\right)^2\right)$ to $\tilde{O}\left(\left(\log^* |X|\right)^{1.5}\right)$. Specifically, we show the following theorem.

Theorem 14 *Let X be a totally ordered domain. There is an (ε, δ)-differentially private algorithm for solving the interior point problem on databases of size $n = \tilde{O}\left(\frac{1}{\varepsilon} \cdot (\log^* |X|)^{1.5} \cdot \log^{1.5}(\frac{1}{\delta})\right)$.*

Recall that in the construction in Section 3, the database size is reduced (additively) by $\approx \frac{1}{\varepsilon} \log \frac{1}{\delta}$ in each iteration, and hence we needed to start with at least $\approx \frac{\log^* |X|}{\varepsilon} \log \frac{1}{\delta}$ input points in order to make it to the end of the recursion without losing all the data points. In addition, with these parameters, algorithm `TreeLog` only guaranteed $\approx (\varepsilon \cdot \log^* |X|, \delta \cdot \log^* |X|)$-differential privacy. To get (ε, δ)-privacy overall, we divided our privacy parameters by $\approx \frac{1}{\log^* |X|}$, which gives sample complexity of $\tilde{O}\left(\left(\log^* |X|\right)^2\right)$. Informally, in this section we apply composition theorems for differential privacy to argue that it suffices to work with a privacy parameter of $\approx \frac{\varepsilon}{\sqrt{\log^* |X|}}$, which would result in a sample complexity of $\tilde{O}\left(\left(\log^* |X|\right)^{1.5}\right)$. However, as we explain next, this requires some additional technical work, and does not follow from a direct application of existing composition theorems to the construction of Section 3.

Recall that composition theorems for differential privacy state that the application of k, (ε, δ)-differentially private mechanisms, satisfies $\approx (\varepsilon \sqrt{k}, \delta k)$-differential privacy. For example, consider an algorithm $\mathcal{B}$ that interacts with its input database S only through differentially private mechanisms, as follows. In every step $i \in [k]$, algorithm $\mathcal{B}$ selects an (ε, δ)-differentially private mechanism $\mathcal{A}_i$, and runs $\mathcal{A}_i$ on S to obtain an outcome a_i, where the choice of $\mathcal{A}_i$ might depend on the previous outcomes $a_1, \ldots, a_{i-1}$. Assuming that this is the only interaction $\mathcal{B}$ has with its input database, then these theorems state that $\mathcal{B}$ is $\approx (\varepsilon \sqrt{k}, \delta k)$-differentially private.

Unraveling the recursion in Algorithm `TreeLog` from Section 3, we observe that it consists of $2 \log^* |X|$ steps. Each of the first $\log^* |X|$ steps computes the input to the following recursive call, and each of the last $\log^* |X|$ steps uses the output of the recursive call to compute an interior point of large score. The difficulty in applying the standard composition arguments to `TreeLog` arises since the first $\log^* |X|$ steps are obviously not differentially private: They just spit out the private input compressed into a smaller domain. (Furthermore, the path which we use to do the compression is also highly sensitive.)

Alternatively, one can think of Algorithm `TreeLog` as consisting of $\log^* |X|$ steps, where the first step computes all of the paths and databases throughout the execution down to the last recursive call, and then privately identifies an interior point of the database in that last recursive call. Afterwards, each of the next steps takes a privately computed interior point for the database at depth d of the recursion, and privately translates it into an interior point for the database at depth $d - 1$. Now, when viewing algorithm `TreeLog` this way, the difficulty in applying composition theorems is that all of these $\log^* |X|$ steps *share a state* which was *not* computed privately. Specifically, all of these steps know (and use) the paths (or the databases) that were computed throughout the execution. This picture is not supported by existing composition theorems, which allow only a *privately* computed state to be shared among the composed mechanisms.

Loosely speaking, we overcome this difficulty by modifying `TreeLog` such that it becomes a composition of $O(\log^* |X|)$ differentially private mechanisms that do not share a non-private state. The crucial observation that enables this modification is that once we reach a level $d^* \in [\log^* |X|]$ of the recursion in which many points fall off the heavy path at the same node, then we can in fact stop the recursion and *privately* report the level d^*. Recall that in algorithm `TreeLog` we sample a random path π by proceeding randomly from a node v to one of its children with probability that grows exponentially with the weight of the child. Now, since d^* is the first level in the recursion in which many points fall off the heavy path at the same node, in all levels $d < d^*$, when constructing the path π we always have that the weight of one child is significantly bigger then the weight of its sibling. This means that we will in fact proceed to the heavier child with overwhelming probability, to the extent that we can ignore the randomness in that step and just select it deterministically.

This allows us to modify the way in which the paths are selected throughout the execution to be *deterministic*. As a result, we are able to decompose `TreeLog` into $O(\log^* |X|)$ different algorithms (one for every level of the recursion), where each of these algorithms recomputes from scratch all of the input databases up to its current level. More specifically, we decompose algorithm `TreeLog` into the following 5 algorithms:

1. **Algorithm `ConstructPaths`:** This algorithm (deterministically) generates all the paths and the databases for the entire execution. The other algorithms use `ConstructPaths` as a subroutine in order to recompute the paths and the databases.

2. **Algorithm `StoppingPoint`:** This algorithm instantiates the sparse vector technique in order to privately identify the first level in which, when following the heavy path deterministically, we reach a node such that both its children have large weight. In other words, this algorithm privately computes a stopping point for the construction.

3. **Algorithm `OneRandomPath`:** This algorithm is similar to (one call of) algorithm `TreeLog` that samples a path randomly. It is applied only once at the last level in the execution, where we might not be able to select the path deterministically.

4. **Algorithm `LevelUp`:** This algorithm gets a parameter d and a point y which is an interior point of the $(d+1)$th database that algorithm `ConstructPaths` generates, and translates y into an interior point of the dth database that `ConstructPaths` generates. This algorithm is executed up to $\log^* |X|$ times (with different parameters).

5. **Algorithm `HeavyPaths`:** This is a wrapper algorithm that runs the previous 4 algorithms.

Before presenting these five algorithms, we recall the sparse vector technique of Dwork et al. (2009), which we use in order to compute a stopping point for the construction. Consider a sequence of low sensitivity functions $f_1, f_2, \ldots, f_m$, which are given (one by one) to a data curator (holding a database S). Algorithm `AboveThreshold` by Dwork et al. (2009) privately identifies the first query f_i whose value $f_i(S)$ is greater than some threshold. Formally,

Theorem 15 (Algorithm `AboveThreshold` Dwork et al. (2009)) *There exists an $(\varepsilon, 0)$ differentially private algorithm $\mathcal{A}$ such that for m rounds, after receiving a sensitivity-1 query $f_i : U^* \to \mathbb{R}$, algorithm $\mathcal{A}$ either outputs $\top$ and halts, or outputs $\bot$ and waits for the next round. If $\mathcal{A}$ was executed with a database $S \in U^*$ and a threshold parameter c, then the following holds with probability $(1 - \beta)$: (i) If a query f_i was answered by $\top$ then $f_i(S) \geq c - \frac{8}{\varepsilon} \log(2m/\beta)$; (ii) If a query f_i was answered by $\bot$ then $f_i(S) \leq c + \frac{8}{\varepsilon} \log(2m/\beta)$.*

We are now ready to present our improved construction for privately identifying an interior point. We begin by presenting algorithms `ConstructPaths` and `StoppingPoint`. For their analysis, we need the following notation. Let λ be a global constant, and let $k = \frac{\lambda}{\varepsilon} \cdot \log\left(\frac{1}{\delta} \cdot \log^* |X|\right)$. Consider the functions f_d defined in algorithm `StoppingPoint`. For a database $S \in X^n$ we define $d^*(S)$ to be the smallest index such that $f_{d^*(S)}(S) \geq k$.

In the next lemma we analyze how `ConstructPaths` behaves on neighboring databases.

Lemma 16 *Let $S \in X^n$ and $S' \in X^n$ be two neighboring databases, and let $t \geq 2k = \frac{2\lambda}{\varepsilon} \cdot \log\left(\frac{1}{\delta} \cdot \log^* |X|\right)$. Let $\{(X_d, S_d, T_d, \pi_d)\}_{d=1}^{\log^* |X|}$ and $\{(X_d, S'_d, T_d, \pi'_d)\}_{d=1}^{\log^* |X|}$ denote the outcomes of the executions of `ConstructPaths` with parameter t on S and on S', respectively. Then for every $d < d^*(S)$ we have that S_d and S'_d are neighboring databases.*

Proof The proof is by induction on d. For the base case, observe that $S_1 = S$ and $S'_1 = S'$ are neighboring databases. Now fix $d < d^*(S)$, and suppose that S_{d-1} and S'_{d-1} are neighboring databases. Since d is strictly smaller than $d^*(S)$, we have that all the elements of S_d have multiplicities less than $k \triangleq \frac{\lambda}{\varepsilon} \cdot \log\left(\frac{1}{\delta} \cdot \log^* |X|\right)$. By the way S_d is constructed, this means that less than k elements from S_{d-1} fall off the path π_{d-1} in every single node of T_{d-1}. Recall that when constructing the path π_{d-1}, every node in the path has weight at least t (except maybe for the last node in the path whose weight might be $t/2$). This means that throughout the construction of π_{d-1} in Step 4 of `ConstructPaths`, the weight of the child we proceed to (in Step 4c) is always larger than the weight of its sibling by at least $t - k \gg 1$. Since S_{d-1} and S'_{d-1} are neighboring databases by the induction assumption, this gap is also bigger than 1 when constructing π'_{d-1} during the execution of S'. As a result, the construction of π'_{d-1} proceeds identically to the construction of π_{d-1}, except possibly that one path might be longer than the other. So π_{d-1} and π'_{d-1} are either exactly the same path, or one of them is a prefix of the other. As in the analysis of algorithm `TreeLog` in Section 3, in such a case we have that the resulting S_d and S'_d are neighboring databases. ■

The next lemma specifies the utility of algorithm `StoppingPoint`.

Lemma 17 *Let $S \in X^n$ be a database of size $n = \Omega\left(\frac{\log^* |X|}{\varepsilon} \cdot \log\left(\frac{1}{\delta} \cdot \log^* |X|\right)\right)$, and let $\hat{d}$ denote the outcome of `StoppingPoint` on S. Then, with probability at least $1 - \delta$ we have that $\hat{d} \leq d^*(S)$, and that $f_{\hat{d}}(S) \geq \Omega(k) = \Omega\left(\frac{1}{\varepsilon} \cdot \log\left(\frac{1}{\delta} \cdot \log^* |X|\right)\right)$.*

16

Algorithm 2 `ConstructPaths`

Input: A database $S \in X^n$ where X is a totally ordered domain. We assume that $|X|$ is a power of 2 (otherwise extend X to the closest power of 2).

Global parameter: t.

1. If $|S| \leq 10t$ or if $|X| = O(1)$ then halt. Otherwise continue to the next step.

2. Sort S and let $\hat{S} \in X^{n-2t}$ be a database containing all elements of S except for the t largest and t smallest elements.

3. Let T be a complete binary tree with $|X|$ leaves that correspond to elements of X. A leaf u has weight $w(u) = |\{x \in \hat{S} : x = u\}|$. A node v has weight $w(v)$ that equals the sum of the weights of its children.

4. Let π be the path in T constructed as follows (starting from the root):

   (a) Let v be the current node in the path.

   (b) If v is a leaf, or if $w(v) \leq t$, then v is the last node in π.

   (c) Else, let v_0, v_1 be the two children of v in T. Proceed to the child v_b with larger weight $w(v_b)$, where $b \in \{0, 1\}$ and where ties are broken arbitrarily.

5. Initialize $D = \emptyset$. Add elements to D by following the path π starting from the root:

   (a) If $|D| \geq n - 3t$ then goto Step 6.

   (b) Let v be the current node in the path, and let ℓ denote its level in T (the root is in level 0 and the leaves are in level $\log |X|$).

   (c) If v is the last node in π then add $(n - 3t - |D|)$ copies of ℓ to D and goto Step 6.

   (d) Else, let v_{next} be the next node in π, and let v_{other} be the other child of v in T. Add $\min \{w(v_{\text{other}}), n - 3t - |D|\}$ copies of ℓ to D, and goto Step 5a with v_{next} as the current node.

6. Output the domain X, the database S, the tree T, and the path π.

7. Execute `ConstructPaths` recursively on D.

Algorithm 3 `StoppingPoint`

Input: Parameters ε, δ and a database $S \in X^n$ where X is a totally ordered domain.

Additional input: Parameter $t = 2k = \frac{2\lambda}{\varepsilon} \cdot \log \left(\frac{1}{\delta} \cdot \log^* |X|\right)$, where λ is a global constant.

1. Instantiate algorithm `AboveThreshold` (see Theorem 15) with the database S, privacy parameter ε, and threshold $c = k/2$.

2. For $d = 1, 2, \ldots, \log^* |X|$ do

   (a) Define the following query $f_d : X^* \to \mathbb{N}$. To compute f_d on a database S, apply algorithm `ConstructPaths` on S with parameter $t = 4c$, and let $S_d \in (X_d)^*$ be the dth database that it outputs. Then $f_d(S) = \max_{y \in X_d} |\{x \in S_d : x = y\}|$.

   (b) Query algorithm `AboveThreshold` on f_d. If the answer is $\top$ then halt and output d. Otherwise continue.

Lemma 17 follows directly from the utility properties of algorithm `AboveThreshold` (see Theorem 15; use $\beta = \delta$, notice that the additive error of `AboveThreshold` is at most $k/2$ for an appropriate choice of λ), after observing that when $n = \Omega\left(\frac{\log^* |X|}{\varepsilon} \cdot \log\left(\frac{1}{\delta} \cdot \log^* |X|\right)\right)$ there must exist an index d such that $f_d(S) \geq k$ and therefore $d^*(S)$ is well defined. This follows since after $\log^* |X|$ iterations of algorithm `ConstructPaths` we get that the size of the domain is constant, while the size of the database is still at least $n - O(t \log^* |X|)$. We now proceed with the privacy analysis of Algorithm `StoppingPoint`.

Lemma 18 *Algorithm `StoppingPoint` is (ε, δ)-differentially private.*

Proof Observe that the outcome of algorithm `StoppingPoint` is a post-processing of the outcomes of algorithm `AboveThreshold`. Hence, it suffices to argue that the sequence of outcomes obtained from algorithm `AboveThreshold` satisfies (ε, δ)-differential privacy. Intuitively, but somewhat inaccurately, this will be done by showing that with probability $(1 - \delta)$ all the queries issued to algorithm `AboveThreshold` are of sensitivity 1, in which case `AboveThreshold` guarantees $(\varepsilon, 0)$-differential privacy.

Let $S \in X^n$ and $S' \in X^n$ be two neighboring databases, and let $\{(X_d, S_d, T_d, \pi_d)\}_{d=1}^{\log^* |X|}$ and $\{(X_d, S'_d, T_d, \pi'_d)\}_{d=1}^{\log^* |X|}$ denote the outcomes of the executions of `ConstructPaths` on S and on S', respectively. By Lemma 16, for every $d < d^*(S)$ it holds that S_d and S'_d are neighboring databases. Therefore, for every $d < d^*(S)$, we have that $|f_d(S) - f_d(S')| \leq 1$ by the definition of the query f_d.

We now argue that also for $d \triangleq d^*(S)$ we have that $|f_d(S) - f_d(S')| \leq 1$, even though S_d and S'_d might not be neighboring databases. Recall that $f_d(S)$ equals to the maximal multiplicity of an element of S_d. Equivalently, $f_d(S)$ is determined by the weight (w.r.t. S_{d-1}) of the heaviest node that falls off the previous path π_{d-1}. Suppose that π_{d-1} and π'_{d-1} are *not* the same path and that none of them is a prefix of the other (as otherwise S_d and S'_d would be neighboring databases and then $|f_d(S) - f_d(S')| \leq 1$). Let u be the last node common to π_{d-1} and π'_{d-1}, and let v_0, v_1 denote its children. Since S_{d-1} and S'_{d-1} are neighboring, we have that $|w(v_0) - w'(v_0)| \leq 1$ and that $|w(v_1) - w'(v_1)| \leq 1$, where here w and w' denote the weight of the elements of T_{d-1} during the executions on S and on S', respectively. In addition, since one path proceeds to v_0 and the other proceeds to v_1, we have that $|w(v_0) - w(v_1)| \leq 2$ and that $|w'(v_0) - w'(v_1)| \leq 2$. Note that $z \triangleq \min\{w(v_0), w(v_1)\}$ elements fall off the path π_{d-1} at the node v, and that less than z elements can fall off this path after node v, since we would be left with only $\max\{w(v_0), w(v_1)\} \leq z + 2$ potential elements that can fall off the path after v (which means that the maximal fall could be of size at most $(z + 2)/2$). Hence, $f_d(S)$ and $f_d(S')$ are determined by the weight (w.r.t. S_{d-1} and S'_{d-1}) of the heaviest nodes that fall off the paths π_{d-1} and π'_{d-1}, *before or at* the node v. Since the paths are the same until v, and since S_{d-1} and S'_{d-1} are neighboring databases, we get that $|f_d(S) - f_d(S')| \leq 1$.

So, for the first $d^*(S)$ queries that are issued to algorithm `AboveThreshold` we have that $|f_d(S) - f_d(S')| \leq 1$. Moreover, by Lemma 17, with probability $1 - \delta$, other queries are never issued to algorithm `AboveThreshold`. The lemma therefore follows from the privacy properties of `AboveThreshold`. Specifically, let τ and τ' denote the sequence of outcomes of `AboveThreshold` during the executions of algorithm `StoppingPoint` on S and on S', respectively. Let T be some set of possible such sequences, let $T_{\text{long}} \subset T$ contain all sequences

longer than $d^*(S)$, and let $T_{\text{short}} = T \setminus T_{\text{long}}$. We have that,

$$\begin{aligned}
\Pr[\tau \in T] &= \Pr[\tau \in T_{\text{short}}] + \Pr[\tau \in T_{\text{long}}] \\
&\leq \mathrm{e}^{\varepsilon} \cdot \Pr[\tau' \in T_{\text{short}}] + \delta \\
&\leq \mathrm{e}^{\varepsilon} \cdot \Pr[\tau' \in T] + \delta.
\end{aligned}$$

$\blacksquare$

We now present the analysis of algorithm `OneRandomPath`.

Lemma 19 *Fix two neighboring databases $S, S' \in X^n$, and consider running* `OneRandomPath` *on S and on S' with parameter $d < d^*(S)$. Then for every set of outcomes T we have*

$$\Pr[\texttt{OneRandomPath}(S) \in T] \leq e^{6\varepsilon \cdot \log n} \cdot \Pr[\texttt{OneRandomPath}(S') \in T] + 4\delta n \cdot e^{4\varepsilon \cdot \log n}.$$

The proof of Lemma 19 is essentially identical to the privacy analysis of (a single iteration of) algorithm `TreeLog` (after applying Lemma 16 to argue that in the executions on S and on S' we have that the resulting S_d and S'_d are neighboring databases). We omit the proof for brevity (see Lemmas 11 and 12). (The constants are different than in Lemma 12 since `OneRandomPath` applies the Choosing Mechanism twice where `TreeLog` applies it only once.)

Lemma 20 *Let $t = 2k = \frac{2\lambda}{\varepsilon} \cdot \log\left(\frac{1}{\delta} \cdot \log^* |X|\right)$, where λ is a global constant, and let $S \in X^n$ be a database. Let $\{(X_d, S_d, T_d, \pi_d)\}_{d=1}^{\log^* |X|}$ denote the outcomes of the execution of algorithm* `ConstructPaths` *with parameter t on S. Let algorithm* `OneRandomPath` *be executed on S with parameters t, d such that $f_{d+1}(S) \geq \Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$, where the function $f_{d+1}(\cdot)$ is defined in algorithm* `StoppingPoint`. *Then, with probability at least $1 - \delta$, the outcome y is an interior point of S_d with depth at least $\Omega(t)$.*

Proof The proof is almost identical to the utility analysis of algorithm `TreeLog`, with the following exception. As we next explain, the assumption that $f_{d+1}(S) \geq \Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$ means that in the database D (constructed in Step 4) there is at least one element with large multiplicity. Hence, instead of applying the recursion on D (as we did in algorithm `TreeLog`), we could use the Choosing Mechanism (Lemma 5 with confidence $\beta = \delta$) in order to identify such an element, which would be an interior point of D with large score.

The assumption that $f_{d+1}(S) \geq \Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$ means that if we were to generate the path π *deterministically* (as is done in algorithm `ConstructPaths`) then the resulting database D would be guaranteed to contain at least one element with large multiplicity. Let π_{deter} denote the path that would be obtained by following the heaviest nodes deterministically. We have established that if $\pi = \pi_{\text{deter}}$, then D contains an element with large multiplicity. Otherwise, if $\pi \neq \pi_{\text{deter}}$, then π "breaks off" from the heaviest path at some node v. Since the path π_{deter} proceeds to a different child of v than π, we have that "a lot" of elements fall off of the path π at the node v. Therefore, in that case we again get that the resulting database D contains at least one element with large multiplicity. $\blacksquare$

We now proceed with the analysis of algorithm `LevelUp`. Recall that this algorithm gets a recursive level d and a level i in T_d which is an interior point of the $(d+1)$th database that

---

Algorithm 4 `OneRandomPath`

Input: Parameters ε, δ and a database $S \in X^n$ where X is a totally ordered domain. We assume that $|X|$ is a power of 2 (otherwise extend X to the closest power of 2).

Additional inputs: Parameter $t = 2k = \frac{2\lambda}{\varepsilon} \cdot \log\left(\frac{1}{\delta} \cdot \log^* |X|\right)$, where λ is a global constant, current level $d \in [\log^* |X|]$.

1. Apply algorithm `ConstructPaths` on S and let (X_d, S_d, T_d, π_d) denote its dth outputs.

2. Sort S_d and let $\hat{S}_d \in X^{n-2t}$ be a database containing all elements of S_d except for the t largest and t smallest elements.

3. Let T be a complete binary tree with $|X|$ leaves that correspond to elements of X. A leaf u has weight $w(u) = |\{x \in \hat{S}_d : x = u\}|$. A node v has weight $w(v)$ that equals the sum of the weights of its children.

4. Sample a path π in T, starting from the root and constructed by the following process:

   (a) Let v be the current node in π.

   (b) If v is a leaf, or if $w(v) \leq t$, then v is the last node in π.

   (c) Else, let v_0, v_1 be the two children of v in T. If one of them has weight 0 then proceed to the other child. Otherwise, proceed to v_b (for $b \in \{0, 1\}$) with probability proportional to $\exp(\varepsilon \cdot w(v_b))$.

5. Initialize $D = \emptyset$. Add elements to D by following the path π starting from the root:

   (a) If $|D| \geq n - 3t$ then goto Step 6.

   (b) Let v be the current node in the path, and let ℓ denote its level in T (the root is in level 0 and the leaves are in level $\log |X|$).

   (c) If v is the last node in π then add $(n - 3t - |D|)$ copies of ℓ to D and goto Step 6.

   (d) Else, let v_{next} be the next node in π, and let v_{other} be the other child of v in T. Add $\min\{w(v_{\text{other}}), n - 3t - |D|\}$ copies of ℓ to D, and goto Step 5a with v_{next} as the current node.

6. Use the Choosing Mechanism to choose an element $i \in [\log |X|]$ with large multiplicity in D.

7. Use the Choosing Mechanism to choose a node v^* at level i of T_d with large weight $w(v^*)$.

8. Let v_{left} and v_{right} be the left-most and right-most leaves, respectively, of the sub-tree rooted at v^*. Also let $v_{\text{inner-left}}$ be the right-most leaf of the sub-tree rooted at the left child of v^*, and let $v_{\text{inner-right}}$ be the left-most leaf of the sub-tree rooted at the right child of v^*.

9. Use the Exponential Mechanism to return $y \in \{v_{\text{left}}, v_{\text{right}}, v_{\text{inner-left}}, v_{\text{inner-right}}\}$ with large quality

$$q(S_d, y) = \min\left\{ |\{x \in S_d : x \leq y\}|, \; |\{x \in S_d : x \geq y\}| \right\}.$$

Algorithm 5 `LevelUp`

Input: Parameters ε, δ and a database $S \in X^n$ where X is a totally ordered domain. We assume that $|X|$ is a power of 2.

Additional inputs: Parameter $t = 2k = \frac{2\lambda}{\varepsilon} \cdot \log\left(\frac{1}{\delta} \cdot \log^* |X|\right)$, where λ is a global constant, the current recursive level $d \in [\log^* |X|]$, and a level i in the tree T_d (which is an interior point in S_{d+1}).

1. Apply algorithm `ConstructPaths` on S with the parameter t and let (X_d, S_d, T_d, π_d) denote its dth output.

2. Sort S_d and let $\hat{S}_d$ be a database containing all elements of S_d except for the t largest and t smallest elements.

3. A leaf u of the tree T_d has weight $w(u) = |\{x \in \hat{S}_d : x = u\}|$. A node v has weight $w(v)$ that equals the sum of the weights of its children. Use the Choosing Mechanism to choose a node v^* at level i of T_d with large weight $w(v^*)$.

4. Let v_{left} and v_{right} be the left-most and right-most leaves, respectively, of the sub-tree rooted at v^*. Also let $v_{\text{inner-left}}$ be the right-most leaf of the sub-tree rooted at the left child of v^*, and let $v_{\text{inner-right}}$ be the left-most leaf of the sub-tree rooted at the right child of v^*.

5. Use the Exponential Mechanism to return $y \in \{v_{\text{left}}, v_{\text{right}}, v_{\text{inner-left}}, v_{\text{inner-right}}\}$ with large quality
$$q(S_d, y) = \min\left\{ |\{x \in S_d : x \leq y\}|, |\{x \in S_d : x \geq y\}| \right\}.$$

---

algorithm `ConstructPaths` generates, and translates i into an interior point of the dth database that `ConstructPaths` generates. The following lemma specifies the utility properties of the algorithm.

Lemma 21 *Let $t = 2k = \frac{2\lambda}{\varepsilon} \cdot \log\left(\frac{1}{\delta} \cdot \log^* |X|\right)$, and let $S \in X^n$ be a database. Let $\{(X_d, S_d, T_d, \pi_d)\}_{d=1}^{\log^* |X|}$ denote the outcomes of the execution of `ConstructPaths` with parameter t on S. Let algorithm `LevelUp` be executed on S with a parameters t, d, i such that i is an interior point of the database S_{d+1} with depth at least $\Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$. Then, with probability at least $1 - \delta$, the algorithm returns an interior point of S_d with depth at least $\Omega(t)$.*

Proof The lemma follows from similar arguments to those given in Section 3. In more details, suppose that i is an interior point of the database S_{d+1}, with depth at least $\Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$. This means that at least $\Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$ points from $\hat{S}_d$ fall off the path π_d on or before level i of the tree T_d, and at least $\Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$ points from $\hat{S}_d$ fall off the path π_d on or after level i.

Let $v_{\pi_d}^i$ denote the node of π_d at level i of the tree. Since at least $\Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$ points from $\hat{S}_d$ fall off from π_d on or after level i, we get that the weight of $v_{\pi_d}^i$ is at least $\Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$. Hence, by the properties of the Choosing Mechanism (Lemma 5 with $\beta = \delta/2$), the node v^* identified in Step 3 has weight at least $\Omega\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$ with probability at least $(1 - \delta/2)$. In that case, similarly to the analysis in Section 3, at least one of $\{v_{\text{left}}, v_{\text{right}}, v_{\text{inner-left}}, v_{\text{inner-right}}\}$ is an interior point of $\hat{S}_d$, and hence, an interior point of S_d with depth at least $\Omega(t)$. It follows that the Exponential Mechanism (Lemma 4 with $\beta = \delta/2$) identifies such an interior point of S_d with depth at least $\Omega(t)$ with probability at least $(1 - \delta/2)$. ∎

The following lemma specifies the privacy properties of algorithm `LevelUp`.

Lemma 22 *Fix two neighboring databases $S, S' \in X^n$, and consider running algorithm `LevelUp` on S and on S' with parameter $d < d^*(S)$. Then for every set of outcomes T we have*

$$\Pr[\texttt{LevelUp}(S) \in T] \leq e^{2\varepsilon} \cdot \Pr[\texttt{LevelUp}(S') \in T] + 2\delta.$$

Proof Let S_d and S'_d denote the databases defined in Step 1 of the executions of `LevelUp` on S and on S', respectively. Since $d < d^*(S)$, by Lemma 16 we have that S_d and S'_d are neighboring databases. Therefore, the databases $\hat{S}_d$ and $\hat{S}'_d$ defined in Step 2 of the executions are also neighboring. We then access one of these neighboring databases using the Choosing Mechanism (in Step 3) and using the Exponential Mechanism (in Step 5), and hence, Lemma 22 follows from the privacy properties of these two private mechanisms (and from composition). ∎

Algorithm 6 `HeavyPaths`

Input: Parameters ε, δ and a database $S \in X^n$ where X is a totally ordered domain. We assume that $|X|$ is a power of 2.

1. Let $t = 2k = \frac{2\lambda}{\varepsilon} \cdot \log\left(\frac{1}{\delta} \cdot \log^* |X|\right)$, where λ is a global constant. Apply algorithm `StoppingPoint` on S with parameter t and let d^* denote its output.

2. Run algorithm `OneRandomPath` on S with parameters t and $d = d^* - 1$. Let y_{d^*-1} denote its outcome.

3. For $d = d^* - 2$ down to 1 do:

   (a) Run algorithm `LevelUp` on S with parameters d and t, and with the point y_{d+1}. Denote the outcome as y_d.

4. Return y_1.

We are now ready to present and analyze algorithm `HeavyPaths` that runs algorithms `StoppingPoint`, `OneRandomPath`, and `LevelUp` in order to (privately) identify an interior point of its input database. The following lemma specifies the utility guarantees of the algorithm.

Lemma 23 *Let algorithm `HeavyPaths` be executed on a database $S \in X^n$ of size*

$$n = \Omega\left(\frac{\log^* |X|}{\varepsilon} \cdot \log\left(\frac{1}{\delta} \cdot \log^* |X|\right)\right).$$

Then the algorithm returns an interior point of S with probability at least $1 - O\left(\delta \cdot \log^* |X|\right)$.

Proof Let $\{(X_d, S_d, T_d, \pi_d)\}_{d=1}^{\log^* |X|}$ denote the outcomes of the execution of `ConstructPaths` with parameter t on S. First, by Lemma 17, with probability at least $(1 - \delta)$ we have that the value d^* computed using algorithm `StoppingPoint` is such that $f_{d^*}(S) \geq \Omega(k)$. Then, by Lemma 20, with probability at least $(1 - \delta)$, the point y_{d^*-1} computed by algorithm `OneRandomPath` is an interior point of S_{d^*-1} with depth at least $\Omega(t)$. Therefore, by induction using Lemma 21, with probability at least $(1 - \delta \cdot \log^* |X|)$, for every $1 \leq d \leq d^* - 2$ we have that y_d (computed by

algorithm `LevelUp`) is an interior point of the database S_d with depth at least $\Omega(t)$. This concludes the proof as $S_1 = S$ and therefore y_1 is an interior point of S. ∎

We now proceed with the privacy analysis of algorithm `HeavyPaths`. For two random variables Z_0, Z_1 we write $Z_0 \approx_{(\varepsilon,\delta)} Z_1$ to mean that for any event T and for any $b \in \{0,1\}$ it holds that $\Pr[Z_b \in T] \leq e^\varepsilon \cdot \Pr[Z_{1-b} \in T] + \delta$.

Lemma 24 *Algorithm* `HeavyPaths` *is* $\left(\bar{\varepsilon}, \bar{\delta}\right)$*-differentially private for*

$$\bar{\varepsilon} = O\left(\varepsilon \cdot \sqrt{\log^* |X| \cdot \log \frac{1}{\delta \cdot \log^* |X|}} + \varepsilon^2 \cdot \log^* |X| + \varepsilon \cdot \log n\right),$$

$$\bar{\delta} = O\left(\delta \cdot (n + \log^* |X|) \cdot e^{\bar{\varepsilon}}\right).$$

Proof Let $\mathcal{B}_d$ denote an algorithm consisting of steps 2 and 3 of algorithm `HeavyPaths`, with d as the parameter from Step 1.

Fix two neighboring databases $S, S' \in X^n$. For every $d \leq d^*(S)$, by Lemmas 22 and 19 we have

`LevelUp`$(S) \approx_{(2\varepsilon,2\delta)}$ `LevelUp`(S'), and `OneRandomPath`$(S) \approx_{(\varepsilon',\delta')}$ `OneRandomPath`(S')

for $\varepsilon' = 6\varepsilon \cdot \log n$ and $\delta' = 4\delta n \cdot e^{4\varepsilon \cdot \log n}$. Therefore, applying composition theorems for differential privacy (see (Dwork et al., 2010); we assume that $\varepsilon \leq 1$) to the execution of `OneRandomPath` and the (at most) $\log^* |X|$ executions of `LevelUp`, we get that $\mathcal{B}_d(S) \approx_{(\hat{\varepsilon},\hat{\delta})} \mathcal{B}_d(S')$, for

$$\hat{\delta} = O\left(\delta \cdot \log^* |X| + \delta n \cdot e^{4\varepsilon \log n}\right), \quad \text{and}$$

$$\hat{\varepsilon} = O\left(\varepsilon \cdot \sqrt{\log^* |X| \cdot \log \frac{1}{\delta \cdot \log^* |X|}} + \varepsilon^2 \cdot \log^* |X| + \varepsilon \cdot \log n\right).$$

Consider the executions of `HeavyPaths` on S and on S', and let d^* and $d^{*\prime}$ denote the values obtained in Step 1 of the execution on S and on S', respectively. By Lemmas 17 and 18, for every set of outcomes T we have that

$$\Pr[\texttt{HeavyPaths}(S) \in T] \leq$$

$$\leq \Pr[d^* > d^*(S)] + \sum_{d \leq d^*(S)} \Pr[d^* = d] \cdot \Pr[\texttt{HeavyPaths}(S) \in T | d^* = d]$$

$$= \Pr[d^* > d^*(S)] + \sum_{d \leq d^*(S)} \Pr[d^* = d] \cdot \Pr[\mathcal{B}_d(S) \in T]$$

$$\leq \delta + \sum_{d \leq d^*(S)} \left(e^\varepsilon \cdot \Pr[d^{*\prime} = d] + \delta\right) \cdot \left(e^{\hat{\varepsilon}} \cdot \Pr[\mathcal{B}_d(S') \in T] + \hat{\delta}\right)$$

$$\leq 4\hat{\delta} \cdot e^{\hat{\varepsilon}} \cdot \log^* |X| + e^{\hat{\varepsilon}+\varepsilon} \cdot \sum_{d \leq d^*(S)} \Pr[d^{*\prime} = d] \cdot \Pr[\mathcal{B}_d(S') \in T]$$

$$\leq 4\hat{\delta} \cdot e^{\hat{\varepsilon}} \cdot \log^* |X| + e^{\hat{\varepsilon}+\varepsilon} \cdot \Pr[\texttt{HeavyPaths}(S') \in T].$$

∎

Theorem 14 now follows by combining Lemma 23 and Lemma 24.