# PrivTree: A Differentially Private Algorithm for Hierarchical Decompositions

Jun Zhang[1] Xiaokui Xiao[1] Xing Xie[2]

[1]Nanyang Technological University
{jzhang027, xkxiao}@ntu.edu.sg

[2]Microsoft Research
xingx@microsoft.com

## ABSTRACT

Given a set $D$ of tuples defined on a domain $\Omega$, we study differentially private algorithms for constructing a histogram over $\Omega$ to approximate the tuple distribution in $D$. Existing solutions for the problem mostly adopt a *hierarchical decomposition* approach, which recursively splits $\Omega$ into sub-domains and computes a noisy tuple count for each sub-domain, until all noisy counts are below a certain threshold. This approach, however, requires that we (i) impose a limit $h$ on the recursion depth in the splitting of $\Omega$ and (ii) set the noise in each count to be proportional to $h$. The choice of $h$ is a serious dilemma: a small $h$ makes the resulting histogram too coarse-grained, while a large $h$ leads to excessive noise in the tuple counts used in deciding whether sub-domains should be split. Furthermore, $h$ cannot be directly tuned based on $D$; otherwise, the choice of $h$ itself reveals private information and violates differential privacy.

To remedy the deficiency of existing solutions, we present *PrivTree*, a histogram construction algorithm that adopts hierarchical decomposition but completely eliminates the dependency on a pre-defined $h$. The core of PrivTree is a novel mechanism that (i) exploits a new analysis on the Laplace distribution and (ii) enables us to use only *a constant amount of noise* in deciding whether a sub-domain should be split, without worrying about the recursion depth of splitting. We demonstrate the application of PrivTree in modelling spatial data, and show that it can be extended to handle sequence data (where the decision in sub-domain splitting is not based on tuple counts but a more sophisticated measure). Our experiments on a variety of real datasets show that PrivTree considerably outperforms the states of the art in terms of data utility.

## Keywords

Differential privacy; hierarchical decompositions

## 1. INTRODUCTION

Releasing sensitive data while preserving privacy is a problem that has attracted considerable attention in recent years. The state-of-the-art paradigm for addressing the problem is *differential privacy* [16], which requires that the data released reveals little information about whether any particular individual is present or absent from the data. To fulfill such a requirement, a typical approach adopted by the existing solutions is to publish a noisy version of the data in place of the original one.

In this paper, we consider a fundamental problem that is frequently encountered in differentially private data publishing: Given a set $D$ of tuples defined over a domain $\Omega$, we aim to decompose $\Omega$ into a set $S$ of sub-domains and publish a noisy count of the tuples contained in each sub-domain, such that $S$ and the noisy counts approximate the tuple distribution in $D$ as accurately as possible. Applications of the problem include:

- Private modelling of spatial data [12, 41] often requires generating a multi-dimensional histogram of the input data.
- For differentially private data mining (e.g., $k$-means [48] and regression analysis [29]), one of the general approaches is to first coarsen the input data and inject noise into it, and then use the modified data to derive mining results.
- Existing algorithms for sequence data publishing [7] require identifying frequent patterns (e.g., prefixes) in a given set $D$ of sequences. This is equivalent to asking for a decomposition of the sequence domain $\Omega$ into a set of disjoint sub-domains, such that (i) each sub-domain includes all sequences in $D$ containing a particular pattern, and (ii) the number of sequences included in each sub-domain is larger than a given threshold.

To address the above decomposition problem, the prior art mostly adopts a *hierarchical* approach, which (i) recursively splits $\Omega$ into sub-domains and computes a noisy tuple count for each of them, and (ii) stops splitting a sub-domain when its noisy count is smaller than a threshold. This approach, albeit intuitive, requires a pre-defined limit $h$ on the maximum depth of recursion when splitting $\Omega$. The reason is that, to ensure differential privacy, the amount of noise injected in each tuple count has to be proportional to the maximum recursion depth, and hence, $h$ must be fixed in advance so that the algorithm can decide the correct noise amount to use.

Nevertheless, the choice of $h$ is a serious dilemma: for the algorithm to produce fine-grained sub-domains of $\Omega$, $h$ cannot be small; yet, increasing $h$ would lead to noisier tuple counts, and thus more errors in deciding whether a sub-domain should be split. As a consequence, no choice of $h$ could result in an accurate approximation of the input data. Furthermore, we cannot tune $h$ directly on the input dataset; otherwise, the choice of $h$ itself reveals private information and violates differential privacy. To mitigate these issues, existing work relies on heuristics to select an appropriate value of $h$, and to generate fine-grained decompositions even when $h$ is small. As we show in our experiments, however, those heuris-

*SIGMOD'16, June 26-July 01, 2016, San Francisco, CA, USA*
© 2016 ACM. ISBN 978-1-4503-3531-7/16/06...$15.00
DOI: http://dx.doi.org/10.1145/2882903.2882928

tics are rather ineffective when the input data follows a skewed distribution (which is often the case in practice).

**Contributions.** Motivated by the limitations of existing solutions, we present *PrivTree*, an algorithm for the decomposition problem that adopts the hierarchical approach but completely eliminates the dependency on a pre-defined $h$. In particular, PrivTree requires only *a constant amount of noise* in deciding whether a sub-domain should be split, which enables it to generate fine-grained decompositions without worrying about the recursion depth. Such a surprising improvement is obtained with a novel mechanism for differential privacy that exploits a non-trivial analysis on the Laplace noise [17] to derive an extremely tight privacy bound. Its central insight is that, in the context of hierarchical decomposition, it is possible to publish a sequence $S$ of $0/1$ values using $O(1)$ noise, regardless of the *sensitivity* of $S$ [17]. In contrast, the standard Laplace mechanism [17] requires that noise amount must be proportional to $S$'s sensitivity.

To demonstrate the applications of PrivTree, we apply it to the private modeling of spatial data, and present a non-trivial extension to tackle sequence data, for which we adopt an advanced Markov model and utilize a sophisticated measure (instead of tuple counts) to decide whether a sub-domain should be split. We experimentally evaluate our algorithms on a variety of real data, and show that they considerably outperform the states of the art in terms of data utility.

In addition, we present an in-depth analysis on the connection between PrivTree and the *support vector technique (SVT)* [18, 21, 28], a technique widely adopted for data publishing under differential privacy. We show that there exists a variant of SVT [28] that could have been used to implement PrivTree, if its privacy guarantees are as claimed in previous work [28]. Nevertheless, we prove that the SVT variant [28] does not satisfy differential privacy, which makes it inapplicable in our context.

In summary, we make the following contributions in this paper:

1. We propose PrivTree, a differentially private algorithm for hierarchical decomposition that eliminates the dependency on a pre-defined threshold of the recursion depth.
2. We present applications of PrivTree in modeling spatial and sequence data. (Sections 3 and 4)
3. We analyze the connection between PrivTree and the SVT, and point out a misclaim about the latter in [28]. (Section 5)
4. We conduct extensive experiments to demonstrate the superiority of PrivTree over the states of the art. (Section 6)

## 2. PRELIMINARIES

In this section, we introduce the concepts behind *differential privacy* [16], and define the problem of *spatial decomposition* [14, 46], which we will address with our PrivTree algorithm in Section 3.

### 2.1 Differential Privacy

Let $D$ be a sensitive dataset with $n$ tuples, and $\mathcal{A}$ be a data publishing algorithm that takes $D$ as input and releases a set of information $\mathcal{A}(D)$. Differential privacy requires that $\mathcal{A}(D)$ should be insensitive to the presence or absence of any particular tuple in $D$, so that an adversary cannot infer much private information from $\mathcal{A}(D)$. More formally, differential privacy is defined based on the concept of *neighboring datasets*, as shown in the following.

DEFINITION 2.1 (NEIGHBORING DATASETS [16]). *Two datasets are neighboring if one of them can be obtained by inserting a tuple into the other.*

DEFINITION 2.2 ($\varepsilon$-DIFFERENTIAL PRIVACY [16]). *An algorithm $\mathcal{A}$ satisfies $\varepsilon$-differential privacy if, for any two neighboring datasets $D$ and $D'$ and for any possible output $O$ of $\mathcal{A}$,*

$$\ln\left(\frac{\Pr\left[\mathcal{A}(D)=O\right]}{\Pr\left[\mathcal{A}(D')=O\right]}\right) \le \varepsilon,$$

*where $\Pr[\cdot]$ denotes the probability of an event.*

There exist several mechanisms [17, 24, 38] for achieving differential privacy, among which the most fundamental one is the *Laplace mechanism*. Specifically, the Laplace mechanism considers a function $f$ that takes $D$ as input and outputs a vector of real numbers, and it aims to release $f(D)$ with differential privacy. To achieve this objective, it adds i.i.d. noise into each value in $f(D)$, such that the noise $\eta$ follows a *Laplace distribution* with the following probability density function:

$$\Pr[\eta = x] = \frac{1}{2\lambda}e^{-|x|/\lambda}. \quad (1)$$

We denote the above distribution as *Lap*$(\lambda)$, and refer to $\lambda$ as the *scale* (since the standard deviation of *Lap*$(\lambda)$ is proportional to $\lambda$). Dwork et al. [17] prove that the Laplace mechanism achieves $(S(f)/\lambda)$-differential privacy, where $S(f)$ is the *sensitivity* of $f$ defined as follows:

DEFINITION 2.3 (SENSITIVITY [17]). *Let $f$ be a function that maps a dataset $D$ into a vector of real numbers. The global sensitivity of $f$ is defined as*

$$S(f) = \max_{D,D'} \left\| f(D) - f(D') \right\|_1,$$

*where $D$ and $D'$ are any two neighboring datasets, and $\|\cdot\|_1$ denotes the $L_1$ norm.*

Intuitively, $S(f)$ measures the maximum possible change in $f$'s output when we insert or remove one arbitrary tuple in $f$'s input.

An important property of differential private algorithms is that their composition also ensures differential privacy:

LEMMA 2.1 (COMPOSITION RULE [37]). *Let $\mathcal{A}_1, \ldots, \mathcal{A}_k$ be $k$ algorithms, such that $\mathcal{A}_i$ satisfies $\varepsilon_i$-differential privacy ($i \in [1, k]$). Then, the sequential composition $(\mathcal{A}_1, \ldots, \mathcal{A}_t)$ satisfies $(\sum_{i=1}^{k} \varepsilon_i)$-differential privacy.*

This lemma is particularly useful in proving that an algorithm ensures differential privacy: we can first decompose the algorithm into a few sequential components, and then analyze each component separately; after that, we can apply Lemma 2.1 to establish the overall privacy guarantee of the algorithm.

### 2.2 Spatial Decompositions

Let $D$ be a set of data points in a multi-dimensional space $\Omega$. A *spatial decomposition* [14, 46] of $D$ consists of a tree-structured decomposition of $\Omega$ into its sub-domains, along with a partitioning of the data points among the leaves of the decomposition tree. For example, Figure 1 illustrates a spatial decomposition of a two-dimensional dataset $D$ that contains 12 data points. The decomposition tree has 9 nodes, namely, $v_1, v_2, \ldots, v_9$, each of which is associated with a sub-domain of $\Omega$ (denoted as "*dom*" and visualized as a black rectangle in Figure 1). We refer to each sub-domain as a *region*. The root of the tree, $v_1$, corresponds to a region that covers the entire $\Omega$; this region is recursively divided into four equal-size sub-regions in the lower levels of the tree, until each leaf node contains a sufficiently small number of data points.

The spatial decomposition in Figure 1 is referred to as a *quadtree* [14, 46], and is widely adopted in spatial databases for efficient

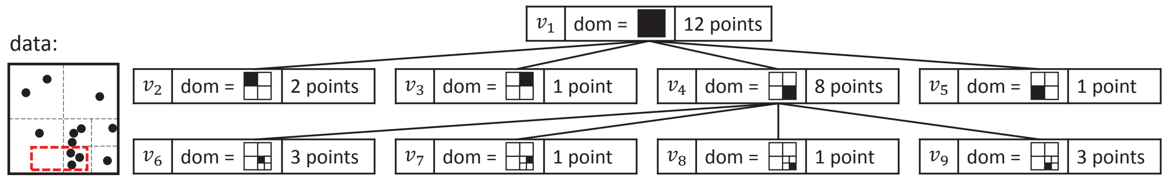


**Figure 1: An illustration of a spatial decomposition tree.**

query processing. In particular, suppose that we are to use the quadtree to answer *range count queries*, i.e., queries that ask for the number of data points contained in a rectangle $q$. In that case, we can pre-compute, for each node $v$ in the quadtree, the number of data points contained in $v$'s region. Then, we can answer any range count query $q$ with a top-down traversal from the root node of the quadtree. Specifically, at the beginning of the traversal, we initialize the query answer as $ans = 0$. After that, for each node $v$ that we traverse, we examine $v$'s region $dom(v)$, and differentiate four cases:

1. If $dom(v)$ is disjoint from $q$, we ignore $v$;
2. If $dom(v)$ is fully contained in $q$, we increase $ans$ by the point count pre-computed for $v$;
3. If $dom(v)$ partially intersects $q$ and $v$ is not a leaf node, then we visit every child of $v$ with a region not disjoint from $q$;
4. If $dom(v)$ partially intersects $q$ and $v$ is a leaf node, then we inspect the data points in $dom(v)$, and add to $ans$ the number of points contained in $q$.

After the traversal terminates, we return $ans$ as the result. For instance, consider a range count query $q$ that corresponds to the dashed-line rectangle in Figure 1. To answer $q$, we only need to examine four nodes, namely, $v_1, v_4, v_5, v_9$; the other nodes are all ignored since their regions are disjoint from $q$.

The efficiency of quadtrees results from its adaptiveness to the underlying distribution, i.e., it grows deep into the dense regions of $\Omega$ where there are a large number of data points (e.g., the region of $v_4$ in Figure 1), and it ignores those regions that are sparse (e.g., the regions of $v_2, v_3, v_5$). Such adaptiveness has motivated existing work [12] to utilize quadtrees for generating private synopses of spatial data. Specifically, the technique in [12] first applies a differentially private algorithm to generate a quadtree, and then employs the Laplace mechanism to inject noise into the point count of each node. The quadtree and the noisy counts can then be used to answer any range-count query $q$ using the top-down traversal algorithm mentioned above, with two minor modifications. First, whenever we visit a node $v$ whose region is fully contained in $q$, we add the noisy count associated with $v$ (instead of the exact count) to the query answer $ans$. Second, if $v$ is a leaf node whose region $dom(v)$ partially intersects $q$, then we multiply the noisy count of $v$ by $\frac{|q \cap dom(v)|}{|dom(v)|}$ before adding it to $ans$, where $|\cdot|$ denotes the area of a region. That is, given only the noisy count of $v$, we estimate the number of data points in $dom(v)$ that are contained in $q$, by assuming that the points follow a uniform distribution. The rationale of this approach is that, given the adaptiveness of the quadtree, each leaf node $v$ should cover a region where the data distribution is not highly skewed; otherwise, $v$ should contain a dense sub-region, in which case the quadtree construction algorithm should have further split $v$ (instead of making $v$ a leaf node). This makes it relatively accurate to adopt a uniform assumption when estimating the contribution of $v$ to the answer of $q$. In Section 3, we will present a more detailed analysis of the above quadtree approach, and then use it to motivate our PrivTree algorithm.

**Table 1: Table of notations**

| Notation | Description |
|---|---|
| $n, d$ | the cardinality and dimensionality of the input dataset $D$ |
| $Lap(\lambda)$ | a random variable following the Laplace distribution with 0 mean and $\lambda$ scale |
| $dom(v)$ | the sub-domain of a node $v$ |
| $depth(v)$ | the hop distance from a node $v$ to the root of the tree |
| $\theta$ | the threshold used to decide if a node should be split |
| $c(v), \hat{c}(v)$ | the point count of a node $v$, and its noisy version |
| $b(v), \hat{b}(v)$ | the biased count of a node $v$, and its noisy version |
| $\rho(v)$ | the privacy risk of a node $v$ (see Equation (5)) |
| $\rho^{\top}(v)$ | an upper bound of $\rho$ (see Equation (7)) |
| $\beta$ | the fanout of the spatial decomposition tree |
| $\delta$ | the decaying factor used by PrivTree |
| $\mathcal{I}$ | the set of distinct items in a given set $D$ of sequences |

## 3. PRIVATE SPATIAL DECOMPOSITIONS

This section presents our solution for constructing private spatial decompositions. We first revisit the private quadtree approach (in Section 2.2) and discuss its limitations; after that, we elaborate our PrivTree algorithm, analyze its guarantees, and discuss its extensions. Table 1 shows the notations that we frequently use.

### 3.1 Private Quadtrees Revisited

Algorithm 1 presents a generic version of the private quadtree approach mentioned in Section 2.2. The algorithm takes as input four parameters: (i) a set $D$ of spatial points defined over a multi-dimensional domain $\Omega$, (ii) the scale $\lambda$ of the Laplace noise to be used in the construction of the quadtree, (iii) the threshold $\theta$ used to decide whether a quadtree node should be split, and (iv) the threshold $h$ on the maximum height of the decomposition tree. The output of the algorithm is a quadtree $\mathcal{T}$ where each node $v$ comes with two pieces of information: the sub-domain of $\Omega$ corresponding to $v$ (denoted as $dom(v)$), and a noisy version of the point count in $dom(v)$ (denoted as $\hat{c}(v)$). We define the *depth* of $v$ as the hop distance between $v$ and the root of $\mathcal{T}$, and denote it as $depth(v)$.

The algorithm starts by creating the root note $v_1$ of $\mathcal{T}$, after which it sets $dom(v_1) = \Omega$ and marks $v_1$ as *unvisited* (Lines 1-2). The subsequent part of the algorithm consists of a number of iterations (Lines 3-9). In each iteration, we examine if there is an unvisited node $v$ in $\mathcal{T}$. If such $v$ exists, we mark $v$ as visited, and employ the Laplace mechanism to generate a noisy version $\hat{c}(v)$ of the number of points contained in $dom(v)$ (Lines 4-6). After that, we split $v$ if the following two conditions simultaneously hold. First, $\hat{c}(v) > \theta$, i.e., $dom(v)$ is likely to contain a sufficiently large number of points. Second, the height of the tree is smaller than $h$, which, as we discuss shortly, ensures that the noisy counts generated by the algorithm would not violate differential privacy. If both of the above conditions are met, then we generate $v$'s children and insert them into $\mathcal{T}$ as unvisited nodes (Lines 7-9); otherwise, $v$ becomes a leaf node of $\mathcal{T}$. When all of the nodes in $\mathcal{T}$ become visited, the algorithm terminates and returns $\mathcal{T}$.

**Algorithm 1: SimpleTree** ($D$, $\lambda$, $\theta$, $h$)

```
1  initialize a quadtree $\mathcal{T}$ with a root node $v_1$;
2  set $dom(v_1) = \Omega$, and mark $v_1$ as unvisited;
3  while there exists an unvisited node $v$ do
4      mark $v$ as visited;
5      compute the number $c(v)$ of points in $D$ that are contained in $dom(v)$;
6      compute a noisy version of $c(v)$: $\hat{c}(v) = c(v) + Lap(\lambda)$;
7      if $\hat{c}(v) > \theta$ and $depth(v) < h - 1$ then
8          split $v$, and add its children to $\mathcal{T}$;
9          mark the children of $v$ as unvisited;
10 return $\mathcal{T}$
```

**Privacy and Utility Analysis.** Algorithm 1 ensures $\varepsilon$-differential privacy if $\lambda \geq h/\varepsilon$. To understand this, suppose that we insert an arbitrary point $t$ into $D$. Then, $\mathcal{T}$ has only $h$ nodes whose exact point counts are affected by the insertion of $t$, i.e., the $h$ nodes whose sub-domains contain $t$. In addition, the point count of those nodes should change by one after $t$'s insertion. This indicates that the sensitivity (see Definition 2.3) of all point counts in $\mathcal{T}$ equals $h$, and hence, adding i.i.d. Laplace noise of scale $\lambda \geq h/\varepsilon$ into the counts would achieve $\varepsilon$-differential privacy.

As we mention in Section 1, however, requiring $\lambda \geq h/\varepsilon$ makes it rather difficult for Algorithm 1 to generate high-quality quadtrees. Specifically, if we set $h$ to a small value, the resulting quadtree $\mathcal{T}$ would not adapt well to the data distribution in $D$, due to the restriction on the tree height; meanwhile, increasing $h$ would also increase the amount of noise in each $\hat{c}(v)$, which makes Algorithm 1 more error-prone in deciding whether a node should be split, thus degrading the quality of $\mathcal{T}$. In other words, any choice of $h$ inevitably leads to inferior data utility. Furthermore, we cannot directly tune $h$ by (i) testing the performance of Algorithm 1 on $D$ under different settings of $h$, and then (ii) selecting the one that yields the best result. The reason is that such a tuning process violates differential privacy: when we change the input data from $D$ to a neighboring dataset $D'$, the tuning process may select a different $h$, in which case Algorithm 1 would use different noise scales for $D$ and $D'$, invalidating its privacy guarantee.

To alleviate the above issue, existing work [12, 41, 42, 48] resorts to heuristics to choose $h$ without violating differential privacy, and to enhance the performance of Algorithm 1, e.g., by avoiding the generation of noisy counts for certain levels of the decomposition tree (so that $h$ can be reduced), and by exploiting correlations among the noisy counts to improve their accuracy [25]. However, none of those heuristics is able to thoroughly address the limitations of Algorithm 1. As we shown in our experiments in Section 6, existing approaches tend to provide inferior data utility, especially when the input data follows a skewed distribution.

## 3.2 Rationale Behind Our Solution

To remedy the deficiency of Algorithm 1, we aim to eliminate the requirement that $\lambda \geq h/\varepsilon$, and make $\lambda$ a constant instead. This would not only resolve the dilemma in choosing $h$, but also unleash the potential of quadtrees as the tree height is no longer restricted. Towards this end, we first make a simple observation: after we finish constructing the quadtree $\mathcal{T}$ in Algorithm 1, we could remove all noisy counts associated with the intermediate nodes, and release only the noisy counts for the leaf nodes as well as the sub-domains of all nodes. The released tree, denoted as $\mathcal{T}'$, could still be used for query processing, since we can re-generate an alternative count for each intermediate node $v$ in $\mathcal{T}'$ by summing up the published noisy counts of the leaf nodes under $v$. Intuitively, $\mathcal{T}'$ reveals less information than $\mathcal{T}$ does, and hence, we might use less noise in $\mathcal{T}'$ to achieve $\varepsilon$-differential privacy.

However, the above intuition does not hold in general, as $\mathcal{T}'$ and $\mathcal{T}$ require the same amount of noise to enforce the same privacy guarantee. To explain, consider two neighboring datasets $D$ and $D'$, such that $D$ is obtained by inserting a point $t$ into $D'$. Let $v_1, v_2, \ldots, v_h$ be the $h$ nodes in $\mathcal{T}'$ whose sub-domains contain $t$. Then, these $h$ nodes should form a path from the root of $\mathcal{T}$ to a leaf node. Furthermore, for any $v_i$, the point count of $v_i$ is decreased by one when we change the input dataset from $D$ to $D'$. We use $c(v_i)$ to denote $v_i$'s exact point count on $D$.

Without loss of generality, assume that $v_h$ is a leaf node (i.e., $v_1, \ldots, v_{h-1}$ are all intermediate nodes). Let $\Pr[D \to \mathcal{T}']$ (resp. $\Pr[D' \to \mathcal{T}']$) denote the probability that we obtain $\mathcal{T}'$ from $D$ (resp. $D'$) given fixed $\lambda$, $\theta$, and $h$. Then,

$$\ln\left(\frac{\Pr[D \to \mathcal{T}']}{\Pr[D' \to \mathcal{T}']}\right) = \sum_{i=1}^{h-1} \ln\left(\frac{\Pr[c(v_i) + Lap(\lambda) > \theta]}{\Pr[c(v_i) - 1 + Lap(\lambda) > \theta]}\right) + \ln\left(\frac{\Pr[c(v_h) + Lap(\lambda) = \hat{c}(v_h)]}{\Pr[c(v_h) - 1 + Lap(\lambda) = \hat{c}(v_h)]}\right).$$

By Equation (1), for any $c(v_h) < \hat{c}(v_h)$,

$$\ln\left(\frac{\Pr[c(v_h) + Lap(\lambda) = \hat{c}(v_h)]}{\Pr[c(v_h) - 1 + Lap(\lambda) = \hat{c}(v_h)]}\right) = \frac{1}{\lambda}. \quad (2)$$

In addition, for any $v_i$ ($i \in [1, h-1]$) with $c(v_i) \leq \theta$,

$$\ln\left(\frac{\Pr[c(v_i) + Lap(\lambda) > \theta]}{\Pr[c(v_i) - 1 + Lap(\lambda) > \theta]}\right) = \frac{1}{\lambda}. \quad (3)$$

Therefore, when $c(v_i) \leq \theta$ holds for every $v_i$ ($i \in [1, h-1]$),

$$\ln\left(\frac{\Pr[D \to \mathcal{T}']}{\Pr[D' \to \mathcal{T}']}\right) = \frac{h}{\lambda}. \quad (4)$$

By Definition 2.2, this indicates that $\lambda$ must be at least $h/\varepsilon$ to ensure that $\mathcal{T}'$ achieves $\varepsilon$-differential privacy.

In summary, $\mathcal{T}'$ is no better than $\mathcal{T}$ in terms of the amount of noise required, because of the negative result in Equations (2) and (3). In other words, *in the worst case*, releasing the boolean result of $c(v_i) + Lap(\lambda) > \theta$ incurs the same privacy cost as releasing $c(v_i) + Lap(\lambda)$ directly. That said, if $c(v_i) > \theta$ for some $v_i$, then Equation (3) does not hold, in which case $\mathcal{T}'$ could entail a smaller privacy cost than $\mathcal{T}$ does. To illustrate this, we denote the l.h.s. of Equation (3) as a function $\rho$ of $c(v_i)$, i.e.,

$$\rho(x) = \ln\left(\frac{\Pr[x + Lap(\lambda) > \theta]}{\Pr[x - 1 + Lap(\lambda) > \theta]}\right), \quad (5)$$

and we plot $\rho$ in Figure 2. (Note that the y-axis of Figure 2 is in a logarithmic scale.) Observe that, when $x = c(v) \geq \theta + 1$, $\rho(x)$ decreases exponentially with the increase of $x$. This indicates that $\ln\left(\frac{\Pr[D \to \mathcal{T}']}{\Pr[D' \to \mathcal{T}']}\right)$ could be much smaller than $h/\lambda$, if $c(v_i) \geq \theta + 1$ holds for all $i \in [1, h-1]$. For example, if $c_{h-1} \geq \theta + 1$ and $c(v_i) - c(v_{i+1})$ is at least a constant for all $i \in [1, h-2]$, then

$$\sum_{i=1}^{h-1} \rho\Big(c(v_i)\Big) = \Theta\left(\frac{1}{\lambda}\right), \quad (6)$$

due to the exponential decrease of $\rho(v_i)$. In that case, we have $\ln\left(\frac{\Pr[D \to \mathcal{T}']}{\Pr[D' \to \mathcal{T}']}\right) = \Theta(1/\lambda)$ instead of $\ln\left(\frac{\Pr[D \to \mathcal{T}']}{\Pr[D' \to \mathcal{T}']}\right) = h/\lambda$, which would enable us to set $\lambda$ as a constant independent of $h$.

The above analysis leads to an interesting question: can we ensure that Equation (6) holds for any input dataset? In Section 3.3, we will give an affirmative answer to this question. The basic idea of our method is to add a bias term to each $c(v_i)$, so that $c(v_i) - c(v_{i+1})$ ($i \in [1, h-2]$) is larger than a constant of choice. In addition, the bias term is independent of the input data, which

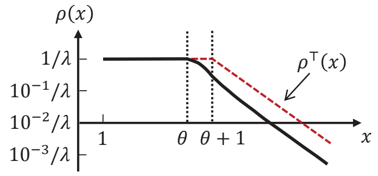


**Figure 2: An illustration of $\rho(x)$ and $\rho^{\top}(x)$.**

guarantees that its usage does not leak any private information. The derivation of the bias term requires a careful analysis of $\rho(x)$. To simplify our analysis, we devise a simple upper bound of $\rho(x)$:

LEMMA 3.1. *Let $\rho^{\top}$ be a function such that*

$$\rho^{\top}(x) = \begin{cases} 1/\lambda, & \text{if } x < \theta + 1 \\ \frac{1}{\lambda}\exp\left(\frac{\theta+1-x}{\lambda}\right), & \text{otherwise} \end{cases} \tag{7}$$

*Then, $\rho(x) \le \rho^{\top}(x)$ for any $x$.*

Figure 2 shows $\rho^{\top}$ with a dashed line. Observe that it closely captures the exponential decrease of $\rho$ when $c(v) \ge \theta + 1$.

## 3.3 The PrivTree Algorithm

Algorithm 2 presents our PrivTree technique for private spatial decomposition. As with Algorithm 1, PrivTree asks for a spatial dataset $D$, the scale $\lambda$ of Laplace noise to be used, and a threshold $\theta$ for deciding whether a node should be split. However, it does not request a threshold $h$ on the maximum tree height; instead, it requires a positive number $\delta$, the usage of which will be clarified shortly. The output of PrivTree is a quadtree $\mathcal{T}$, with the point count associated with each node removed. That is, $\mathcal{T}$ reveals the sub-domain of each node $v$, but conceals all information about $c(v)$. In Section 3.4, we will explain how we obtain the point count of each node, as well as our choices of $\theta$ and $\delta$.

In a nutshell, PrivTree is similar to Algorithm 1 in that it also (i) generates $\mathcal{T}$ by recursively splitting a root node $v_1$ whose region $dom(v_1)$ covers the whole data space $\Omega$, and (ii) decides whether a node $v$ should be split based on a noisy point count of $v$. However, the method for obtaining noisy counts marks the crucial difference between the two algorithms. Specifically, given a node $v$, PrivTree does not generate its noisy count by directly adding Laplace noise to $c(v)$. Instead, PrivTree first computes a biased count $b(v) = c(v) - depth(v) \cdot \delta$, and checks if it is smaller than $\theta - \delta$; if it is, then PrivTree increases it to $\theta - \delta$. In other words,

$$b(v) = \max\Big\{\theta - \delta,\ \ c(v) - depth(v)\cdot\delta\Big\}. \tag{8}$$

After that, PrivTree produces a noisy count $\hat{b}(v) = b(v) + Lap(\lambda)$, and splits $v$ if $\hat{b}(v)$ is larger than the given threshold $\theta$. Notice that PrivTree does not restrict the height of $\mathcal{T}$, as the decision to split any node $v$ solely depends on $\hat{b}(v)$.

**Privacy Analysis.** Consider any quadtree $\mathcal{T}$ output by PrivTree, and any two neighboring datasets $D$ and $D'$, such that $D$ is obtained by inserting a point $t$ into $D'$. In what follows, we show that setting $\lambda = \Theta(1/\varepsilon)$ is sufficient for $\varepsilon$-differential privacy, i.e.,

$$-\varepsilon \le \ln\left(\frac{\Pr[D \to \mathcal{T}]}{\Pr[D' \to \mathcal{T}]}\right) \le \varepsilon. \tag{9}$$

The proof for the first inequality in Equation (9) is relatively straightforward. For any node $v$ in $\mathcal{T}$, let $c(v)$ be $v$'s point count on $D$, and $b(v)$ be the biased version of $c(v)$ generated from Equation (8). Let $c'(v)$ and $b'(v)$ be the counterparts of $c(v)$ and $b(v)$, respectively, given $D'$ as the input. Then, we have $c(v) = c'(v)$ for all nodes $v$ in $\mathcal{T}$, except for the nodes whose sub-domains contain $t$. Note that those nodes should form a path from the root of $\mathcal{T}$ to a leaf. Let $k$ be the length of the path, and $v_i$ be $i$-th node in the path, with $v_1$ denoting the root of $\mathcal{T}$. We have $c(v_i) = c'(v_i) + 1$ and

$$b(v_i) = \begin{cases} b'(v_i) + 1, & \text{if } b(v_i) \ge \theta - \delta + 1 \\ b'(v_i), & \text{otherwise} \end{cases} \tag{10}$$

Then, by Equation (1),

$$\begin{aligned} \ln\left(\frac{\Pr[D \to \mathcal{T}]}{\Pr[D' \to \mathcal{T}]}\right) &= \sum_{i=1}^{k-1} \ln\left(\frac{\Pr[b(v_i) + Lap(\lambda) > \theta]}{\Pr[b'(v_i) + Lap(\lambda) > \theta]}\right) \\ &\quad + \ln\left(\frac{\Pr[b(v_k) + Lap(\lambda) \le \theta]}{\Pr[b'(v_k) + Lap(\lambda) \le \theta]}\right) \\ &\ge 0 - \frac{1}{\lambda} \ \ = -\frac{1}{\lambda}. \end{aligned}$$

This indicates that $\lambda \ge 1/\varepsilon$ ensures the first inequality in Equation (9).

Next, we prove the second inequality in Equation (9) by analyzing $\ln\left(\frac{\Pr[b(v_i)+Lap(\lambda)>\theta]}{\Pr[b'(v_i)+Lap(\lambda)>\theta]}\right)$, which we refer to as the *privacy cost* of $v_i$. The high-level idea of our proof is as follows. First, due to the way that we generate biased counts, each node $v_i$'s bias count $b(v_i)$ is at least a constant $\delta$ smaller than that of its parent $v_{i-1}$, as long as $b(v_i) \ge \theta + 1$. Based on this observation and Lemma 3.1, we show that all nodes $v_i$ with $b(v_i) \ge \theta + 1$ incur a total privacy cost of $\Theta(1/\varepsilon)$. After that, we prove that the total privacy cost of the remaining nodes is also $\Theta(1/\varepsilon)$.

By the definition of $v_1, \ldots, v_k$, we have $c(v_i) \ge c(v_{i+1})$ and $depth(v_i) = depth(v_{i+1}) - 1$ for any $i \in [1, k-1]$. This indicates that $b(v_i) \ge b(v_{i+1}) \ge \theta - \delta$, due to Equation (8). Without loss of generality, assume that there exists $m \in [1, k-1]$, such that $b(v_m) \ge \theta - \delta + 1$ and $b(v_{m+1}) = \theta - \delta$. Then,

$$\begin{cases} b(v_{i-1}) \ \ge\ b(v_i) + \delta \ \ge\ \theta + 1, & \text{if } i \in [2, m] \\ b(v_i) \ =\ \theta - \delta, & \text{otherwise} \end{cases} \tag{11}$$

Combining Equations (10) and (11), we have $b'(v_i) = b(v_i)$ when $i > m$, and $b'(v_i) = b(v_i) - 1$ otherwise. Therefore,

$$\begin{aligned} \ln\left(\frac{\Pr[D \to \mathcal{T}]}{\Pr[D' \to \mathcal{T}]}\right) &= \sum\nolimits_{i=1}^{k-1} \ln\left(\frac{\Pr[b(v_i) + Lap(\lambda) > \theta]}{\Pr[b'(v_i) + Lap(\lambda) > \theta]}\right) \\ &\quad + \ln\left(\frac{\Pr[b(v_k) + Lap(\lambda) \le \theta]}{\Pr[b'(v_k) + Lap(\lambda) \le \theta]}\right) \\ &\le \sum\nolimits_{i=1}^{k-1} \ln\left(\frac{\Pr[b(v_i) + Lap(\lambda) > \theta]}{\Pr[b'(v_i) + Lap(\lambda) > \theta]}\right) \\ &= \sum\nolimits_{i=1}^{m} \ln\left(\frac{\Pr[b(v_i) + Lap(\lambda) > \theta]}{\Pr[b(v_i) - 1 + Lap(\lambda) > \theta]}\right) \\ &= \sum\nolimits_{i=1}^{m} \rho\Big(b(v_i)\Big), \end{aligned}$$

**Algorithm 2: PrivTree** ($D$, $\lambda$, $\theta$, $\delta$)

```
1  initialize a quadtree $\mathcal{T}$ with a root node $v_1$;
2  set $dom(v_1) = \Omega$, and mark $v_1$ as unvisited;
3  while there exists an unvisited node $v$ do
4      mark $v$ as visited;
5      compute a biased point count for $v$ with decaying factor $\delta$:
       $b(v) = c(v) - depth(v) \cdot \delta$;
6      adjust $b(v)$ if it is excessively small: $b(v) = \max\{b(v), \theta - \delta\}$;
7      compute a noisy version of $b(v)$:  $\hat{b}(v) = b(v) + Lap(\lambda)$;
8      if $\hat{b}(v) > \theta$ then
9          split $v$, and add its children to $\mathcal{T}$;
10         mark the children of $v$ as unvisited;
11 return $\mathcal{T}$ with all point counts removed
```

where $\rho(\cdot)$ is as defined in Equation (5). By Lemma 3.1 and Equation (11),

$$\begin{aligned}\sum_{i=1}^{m} \rho\Big(b(v_i)\Big) &\leq \sum_{i=1}^{m} \rho^{\top}\Big(b(v_i)\Big)\\ &= \rho^{\top}\Big(b(v_m)\Big) + \sum_{i=1}^{m-1} \frac{1}{\lambda} \exp\left(\frac{\theta + 1 - b(v_i)}{\lambda}\right)\\ &\leq \frac{1}{\lambda} + \frac{1}{\lambda} \cdot \frac{1}{1 - \exp(-\delta/\lambda)}\\ &= \frac{1}{\lambda} \cdot \frac{2e^{\delta/\lambda} - 1}{e^{\delta/\lambda} - 1}.\end{aligned}$$

Therefore, if we set $\delta = \gamma \cdot \lambda$, where $\gamma$ is a constant, then

$$\ln\left(\frac{\Pr[D \to \mathcal{T}]}{\Pr[D' \to \mathcal{T}]}\right) = \sum_{i=1}^{m} \rho\Big(b(v_i)\Big) \leq \frac{1}{\lambda} \cdot \frac{2e^{\gamma} - 1}{e^{\gamma} - 1} = \Theta\left(\frac{1}{\lambda}\right).$$

Summing up the above analysis, we have the following theorem:

THEOREM 3.1. *PrivTree satisfies $\varepsilon$-differential privacy if $\lambda \geq \frac{2e^{\gamma}-1}{e^{\gamma}-1} \cdot \frac{1}{\varepsilon}$ and $\delta = \gamma \cdot \lambda$ for some $\gamma > 0$.*

## 3.4 Noisy Counts and Parameterization

**Generation of Noisy Counts.** Recall that PrivTree outputs a quadtree with the point count for each node removed. However, if a quadtree with noisy counts is needed, we can easily obtain it by adding a postprocessing step to PrivTree. In particular, given a dataset $D$, we first invoke PrivTree to produce a $\frac{\varepsilon}{2}$-differentially private quadtree $\mathcal{T}$. After that, for each leaf node $v$ of $\mathcal{T}$, we publish a noisy version of $v$'s point count using Laplace noise of scale $2/\varepsilon$. It can be verified that this postprocessing step satisfies $\frac{\varepsilon}{2}$-differential privacy. Then, by Lemma 2.1, the generation of $\mathcal{T}$ and the noisy counts as a whole achieves $\varepsilon$-differential privacy. Finally, we compute a noisy count for each intermediate node $v$ in $\mathcal{T}$, by taking the sum of the noisy counts of all leaf nodes under $v$.

**Choice of $\delta$.** As shown in Theorem 3.1, when $\delta = \gamma \cdot \lambda$, PrivTree needs to use a noise scale $\lambda \geq \frac{2e^{\gamma}-1}{e^{\gamma}-1} \cdot \frac{1}{\varepsilon}$ to achieve $\varepsilon$-differential privacy. Intuitively, the choice of $\delta$ is a balancing act between the amount of bias and the amount of noise in each biased noisy count $\hat{b}(v)$ used by PrivTree. In particular, if $\delta$ is small with respect to $\lambda$, then the bias term in each $\hat{b}(v)$ is small, but the noise amount in $\hat{b}(v)$ would need to be large, since $\frac{2e^{\gamma}-1}{e^{\gamma}-1} \cdot \frac{1}{\varepsilon}$ increases when $\gamma = \delta/\lambda$ decreases. In contrast, if $\delta$ is large with respect to $\lambda$, then each $\hat{b}(v)$ would have small noise but a large bias.

That said, we observe that there is a more important factor in choosing $\delta$. To explain, consider a node $v$ with a biased count $b(v) = \theta - \delta$. Ideally, we would like PrivTree to avoid splitting such a node $v$, as its point count is likely to be small. Nevertheless, if $b(v) + Lap(\lambda) > \theta$, then PrivTree would split $v$ and insert its children into the quadtree. In turn, each of $v$'s children also has a certain probability to be split, and so on. If $\delta$ is excessively small, then each offspring of $v$ has a relatively large splitting probability, in which case the splitting process may not *converge*, i.e., PrivTree may keep generating offsprings of $v$ and does not terminate.

To address the above issue, we set $\delta$ in such a way to ensure that if $b(v) = \theta - \delta$, then in expectation, only 2 nodes would be generated from the subtree under $v$ (including $v$ itself). Specifically, we set $\delta = \lambda \cdot \ln\beta$, where $\beta$ denote the *fanout* of $\mathcal{T}$, i.e., the number of children that each intermediate node in $\mathcal{T}$ has. (For example, $\beta = 4$ if $\mathcal{T}$ is a two-dimensional quadtree.) By Equation (1), this setting of $\delta$ guarantees that any node $v$ with $b(v) = \theta - \delta$ has $\frac{1}{2\beta}$ probability to be split. Formally, we have the following lemma:

LEMMA 3.2. *Let $\mathcal{T}$ be the output of PrivTree given $\delta = \lambda \cdot \ln\beta$, and $\mathcal{T}^*$ be the output of PrivTree when it sets $\hat{b}(v) = c(v)$ for each node $v$ (i.e., no noise or bias is introduced in the split decisions). Then, $\mathbb{E}[|\mathcal{T}|] \leq 2 \cdot |\mathcal{T}^*|$ whenever $|T^*| > 1$, where $|\cdot|$ denotes the number of nodes in a tree.*

The above setting of $\delta$ leads to the following corollary:

COROLLARY 1. *PrivTree satisfies $\varepsilon$-differential privacy if $\lambda \geq \frac{2\beta-1}{\beta-1} \cdot \frac{1}{\varepsilon}$ and $\delta = \lambda \cdot \ln\beta$, where $\beta$ is the fanout of $\mathcal{T}$.*

**Choice of $\theta$.** Intuitively, the threshold $\theta$ serves the purpose of ensuring that each leaf node $v$ of $\mathcal{T}$ contains a sufficiently large point count $c(v)$, so that if we choose to output a noisy version of $c(v)$, it would not overwhelmed by the Laplace noise injected. The choice of $\theta$, however, is complicated by the fact that PrivTree adds a negative bias to the point count of a node when it decides whether or not to split the node. In particular, due to the negative bias, even $\theta = 0$ could ensure that a node $v$ with $b(v) > \theta$ has a sufficient large point count. Therefore, we use $\theta = 0$ in our implementation of PrivTree, and we observe that it leads to reasonably good results in our experiments.

## 3.5 Extensions

Although we have presented PrivTree in the context of spatial decomposition, we note that it could be extended in several different aspects for other applications. First, the decomposition tree used by PrivTree does not have to be a quadtree, but can be any other tree structure instead. For example, suppose that we are given a multi-dimensional dataset $D$ containing both numeric and categorical attributes, and that each categorical attribute has a taxonomy. Then, we can still apply PrivTree on $D$ to generate a private synopsis of $D$, by splitting each numeric dimension of $D$ according to a binary tree and each categorical dimension based on its taxonomy.

Second, when PrivTree decides whether or not a node $v$ should be split, the decision does not have to be based on the count of tuples contained in *dom*$(v)$, but can also be based on any other score function $\mu(v)$ that is *monotonic*, i.e., $\mu(v) \leq \mu(u)$ whenever $v$ is a child node of another node $u$. The rationale is that, as long as $\mu$ is monotonic, we can add a bias to the score of each node $v$ to ensure that it is at least a constant smaller that the score of $v$'s parent. Then, we can apply Lemma 3.1 to show that PrivTree guarantees differential privacy, given that $\lambda$ is properly set based on the sensitivity of the score function $\mu$. In Section 4, we will apply this idea to extend PrivTree for private modeling of sequence data.

Finally, although the privacy analysis of PrivTree (in Section 3.3) assumes that the presence or absence of a tuple $t$ only affects one leaf node and its ancestors, it can be extended to the case when multiples leaf nodes and their ancestors are impacted. In particular, if at most $x$ leaf nodes can be affected, then we can apply PrivTree with the noise scale $\lambda$ enlarged $x$ times. The intuition is that each affected leaf node, along its ancestors, incurs one unit of privacy cost, which in turn requires one unit of noise to mitigate; as such, when there are $x$ affected leaf nodes, we need $x$ units of noise for sufficient privacy protection.

## 4. PRIVATE MARKOV MODELS

This section presents an extension of PrivTree for constructing Markov models on sequence data. We first introduce the basic concepts of sequences and Markov models in Section 4.1. After that, we elaborate our PrivTree extension in Section 4.2, and compare it with existing solutions in Section 4.3.

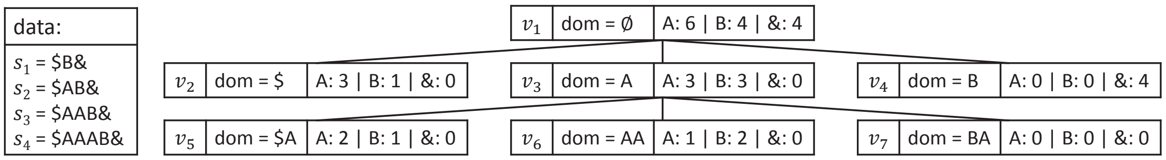

data:
$s_1$ = $B&
$s_2$ = $AB&
$s_3$ = $AAB&
$s_4$ = $AAAB&
$v_1$ | dom = ∅ | A: 6 | B: 4 | &: 4
$v_2$ | dom = $ | A: 3 | B: 1 | &: 0
$v_3$ | dom = A | A: 3 | B: 3 | &: 0
$v_4$ | dom = B | A: 0 | B: 0 | &: 4
$v_5$ | dom = $A | A: 2 | B: 1 | &: 0
$v_6$ | dom = AA | A: 1 | B: 2 | &: 0
$v_7$ | dom = BA | A: 0 | B: 0 | &: 0

**Figure 3: An illustration of a prediction suffix tree (PST).**

## 4.1 Sequence Data and Markov Models

Given a finite alphabet $\mathcal{I}$, a *sequence* $s$ of length $l$ over $\mathcal{I}$ is an ordered list $x_1 x_2 \cdots x_l$, where each $x_i$ ($i \in [1, l]$) is a symbol in $\mathcal{I}$. For convenience, we abuse notation and write $s = \$x_1 x_2 \cdots x_l\&$, where \$ and & are two special symbols that mark the beginning and the end of a sequence, respectively. Sequences are frequently used to represent user behavioral data, such as trajectories, web navigation traces, and product purchasing histories.

Markov models are a type of stochastic models commonly used to characterize sequence data. They assume the *Markov property* [44], i.e., a symbol $x$ in a sequence $s$ is decided by a few symbols that immediately proceeds $x$ in $s$, but not any others. A Markov model over $D$ is often represented as *prediction suffix tree (PST)* [3, 44], where each node $v$ is associated with a *predictor string dom*$(v)$, as well as a *prediction histogram hist*$(v)$. In particular, *dom*$(v)$ consists of symbols in $\mathcal{I} \cup \{\$\}$, while *hist*$(v)$ contains a count for each symbol $x$ in $\mathcal{I} \cup \{\&\}$. The count, denoted as *hist*$(v)[x]$, is computed as follows. We first inspect all occurrences of *dom*$(v)$ in the sequences in $D$, and count the number $y$ of occurrences when *dom*$(v)$ is immediately followed by the symbol $x$; after that, we set *hist*$(v)[x] = y$. In other words, *hist*$(v)[x]$ indicates how often an appearance of *dom*$(v)$ would immediately lead to an appearance of $x$ in a sequence.

For example, Figure 3 illustrates a PST constructed over a set $D$ containing four sequences $s_1, s_2, s_3, s_4$, over an alphabet $\mathcal{I} = \{A, B\}$. The node $v_6$ has a predictor string *dom*$(v_6) = AA$, and prediction histogram *hist*$(v_6)$ that contains a count for each element in $\{A, B, \&\}$. The counts in the histogram sum up to 3, since the string $AA$ appears 3 times in $D$, i.e., once in $s_3$ and twice in $s_4$. In addition, *hist*$(v_6)[A] = 1$ because, among the 3 occurrences of $AA$, only one is immediately followed by $A$ (i.e., the first occurrence of $AA$ in $s_4$). The root node $v_1$ has an empty predictor string *dom*$(v_1) = \emptyset$, and its prediction histogram counts the occurrences of each individual symbol in $\mathcal{I} \cup \{\&\}$, e.g., *hist*$(v_1)[A] = 6$ since the symbol $A$ appears 6 times in total in $D$.

The nodes in a PST are organized in such a way that each node $v$ has $|\mathcal{I}| + 1$ children. Furthermore, for each child $v'$ of $v$, *dom*$(v')$ is obtained by adding a symbol in $\mathcal{I} \cup \{\$\}$ in the beginning of *dom*$(v)$. That is, *dom*$(v)$ is a suffix of *dom*$(v')$. For example, in the PST in Figure 3, $v_3$ is a parent of $v_5$; accordingly, *dom*$(v_5) = \$A$ is obtained by adding the symbol \$ to the beginning of *dom*$(v_3) = A$. The intuition here is that (i) each node $v$ in a PST provides a way to predict the "next symbol" in a sequence based on a "predicate" *dom*$(v)$, and (ii) when we split $v$, each child node would have a longer "predicate" that provides a more specific predication.

A PST $\mathcal{T}$ can be used to support a wide range of queries, such as estimating the number of times that a query string $s_q$ appears in the sequences in $D$. Specifically, given $s_q = x_1 x_2 \ldots x_l$, we first inspect the root node $v_1$'s prediction histogram *hist*$(v_1)$, and then initialize a temporary answer $ans = hist(v_1)[x_1]$. After that, we examine $x_i$ ($i \in [2, l]$) in ascending order of $i$. For each $x_i$, we consider the length-$(i-1)$ prefix of $s_q$, i.e., $s_i^* = x_1 x_2 \ldots x_{i-1}$. We identify the node $v$ in $\mathcal{T}$ whose predictor string is the longest suffix of $s_i^*$. Then, we compute the sum of the counts in $v$'s prediction histogram *hist*$(v)$, referred to as the *magnitude* of the histogram and denoted as $\|hist(v)\|_1$. After that, we set

$$ans = ans \cdot \frac{hist(v)[x_i]}{\|hist(v)\|_1}, \tag{12}$$

i.e., we multiple $ans$ by the probability that the "next symbol" equals $x_i$, as predicted by *hist*$(v)$. When all $x_i$ ($i \in [1, l]$) are examined, we return $ans$ as the query answer.

For example, consider a query sequence $s_q = AB$ on the PST in Figure 3. We first visit the root node $v_1$, and initialize $ans = hist(v_1)[A] = 6$. After that, we consider the length-1 prefix of $s_q$, i.e., $s_2^* = A$. We identify $v_3$ as the node whose predictor string is the longest suffix of $s_2^*$, and we set $ans = ans \cdot \frac{hist(v_3)[B]}{\|hist(v_3)\|_1} = 3$. Finally, we return $ans = 3$ as the answer.

In addition to the aforementioned query type, we can also utilize a PST $\mathcal{T}$ to generate a *synthetic sequence dataset*, by sampling sequences from $\mathcal{T}$ one by one. Specifically, to generate a sequence, we start from an initial sequence $s_0 = \$$ and insert symbols into $s_0$ iteratively. In the $i$-th iteration ($i \geq 1$), we inspect the sequence $s_{i-1}$, and identify the node $v$ in $\mathcal{T}$ whose predictor string is the longest suffix of $s_{i-1}$. Then, we sample a symbol $x_i$ from the symbol distribution represented by *hist*$(v)$, i.e., $\Pr[x_i = x] = \frac{hist(v)[x]}{\|hist(v)\|_1}$. After that, we insert $x_i$ to the end of $s_{i-1}$, and denote the resulting sequence as $s_i$. If $x_i$ happens to be &, then we return $s_i$ as the result.

## 4.2 Extension of PrivTree

To construct a PST $\mathcal{T}$ on a sequence dataset $D$, we can start from a root node $v_1$ with a predictor string *dom*$(v_1) = \emptyset$, and then recursively split $v_1$. This motivates us to adopt PrivTree for the generation of differentially private PSTs. However, we can no longer use a node $v$'s noisy count $c(v)$ to decide whether $v$ should be split, since $c(v)$ is undefined on a PST. Instead, as discussed in Section 3.5, we can redefine $c(v)$ as a score function that measures the suitability of $v$ for splitting. In the non-private setting, existing work [44] typically avoids splitting a node $v$ if any of the following conditions is satisfied:

C1. *dom*$(v)$ *starts with* \$. In this case, no more symbol can be added to the beginning of *dom*$(v)$; thus, $v$ cannot be split.

C2. *The magnitude of hist*$(v)$ *is small.* The rationale is that, when $\|hist(v)\|_1$ is small, further splitting $v$ results in child nodes $v'$ whose prediction histograms *hist*$(v')$ have even smaller magnitudes. In that case, the symbol distribution captured by *hist*$(v')$ is obtained from a tiny sample set of sequences, which leads to poor prediction accuracy.

C3. *The entropy*[1] *of hist*$(v)$ *is small.* This is because when *hist*$(v)$ has a small entropy, there is little uncertainty in the symbol prediction given by *hist*$(v)$; as such, there is little benefit in splitting $v$. (See $v_4$ in Figure 3 for an example.)

Suppose that we are to adopt the above conditions into PrivTree.

[1]Here we treat *hist*$(v)$ as a probability distribution.

Condition C1 can be straightforwardly applied, since it only depends on *dom*$(v)$ and does not rely on $D$, i.e., it does not leak private information. In contrast, conditions C2 and C3 cannot be directly adopted since the counts in *hist*$(v)$ depend on $D$. To address this issue, we aim to design a score function $c(\cdot)$ for PrivTree with the following two properties:

P1. $c(\cdot)$ is monotonic, i.e., $c(v) \leq c(u)$ for any node $v$ and its parent $u$. This, as discussed in Section 3.5, is required to ensure that PrivTree satisfies differential privacy.

P2. If a node $v$'s prediction histogram has a small magnitude or a small entropy, then $c(v)$ tends to be small. This is motivated by conditions C2 and C3 mentioned above.

Our construction of $c(\cdot)$ is based on the following observation: if a prediction histogram has a small entropy, it often has one symbol count that dominates the others, because a small entropy implies that the distribution of symbols in the histogram is skewed. ($v_4$ in Figure 3 shows an example.) Motivated by this, we define $c(v)$ as

$$c(v) = \|\textit{hist}(v)\|_1 - \max_{x \in \mathcal{I} \cup \{\&\}} \textit{hist}(v)[x], \tag{13}$$

i.e., $c(v)$ equals the magnitude of *hist*$(v)$ minus the largest count in *hist*$(v)$. The intuition is that if the magnitude of *hist*$(v)$ is small, then $c(v)$ must be small, regardless of the largest count in *hist*$(v)$; on the other hand, if the entropy of *hist*$(v)$ is small, then the largest count in *hist*$(v)$ tends to be close to the magnitude of *hist*$(v)$ (since the count often dominates all other counts in *hist*$(v)$), which results in a small $c(v)$ as well. Thus, $c(v)$ fulfills property P2. The following lemma show that $c(v)$ also satisfies property P1.

LEMMA 4.1. *$c(\cdot)$ is a monotonic function.*

In summary, we can construct a private PST on a sequence dataset $D$ using PrivTree (i.e., Algorithm 2), with three minor changes. First, in Line 1 of Algorithm 2, $\mathcal{T}$ is a PST with a fanout $|\mathcal{I}|+1$ (instead of a quadtree), and $v_1$ is a PST node with a predictor string *dom*$(v_1) = \emptyset$ and a prediction histogram *hist*$(v_1)$. Second, in Line 5, $c(v)$ is as defined in Equation (13). Third, in Line 11, we return $\mathcal{T}$ after removing the biased score $\hat{b}(v)$ and the prediction histogram *hist*$(v)$ of each node $v$.

After we obtained the PST $\mathcal{T}$ (without prediction histograms) from the modified PrivTree, we can postprocess $\mathcal{T}$ to recover the prediction histograms. Specifically, for each leaf node $v$ in $\mathcal{T}$, we derive the prediction histogram *hist*$(v)$ from $D$, and then compute a noisy version of *hist*$(v)$, denoted as $\widehat{\textit{hist}}(v)$, by adding Laplace noise into each histogram count. After that, for any non-leaf node $v'$, we construct a noisy prediction $\widehat{\textit{hist}}(v')$, such that for any symbol $x \in \mathcal{I} \cup \{\&\}$,

$$\widehat{\textit{hist}}(v')[x] = \sum_{v \text{ is a leaf node under } v'} \widehat{\textit{hist}}(v)[x].$$

Finally, if any noisy histogram in $\mathcal{T}$ has a negative count, we reset the count to zero. This is to ensure that each histogram represents a distribution of symbols.

**Privacy Analysis and Parameterization.** To analyze the privacy guarantee of the modified PrivTree, we first introduce an assumption that is also adopted in prior work [6] on sequence data publication under differential privacy: we assume that the length of each sequence in $D$, when taking into account & but not \$, is at most $l^\top$, where $l^\top$ is a known constant. To explain why this assumption is needed, consider that we insert an *infinite* sequence $s$ into $D$ to obtain a neighboring dataset $D'$. In that case, the insertion of $s$ incurs unbounded changes in the histogram counts of the PST, which makes it impossible to achieve differential privacy. In general, if $l^\top$ is unknown, we may choose an appropriate $l^\top$ and *truncate* any sequences $s$ that is excessively long[2]. Specifically, if $s = \$x_1x_2 \ldots x_{l^\top}\&$, then we truncate it to $s = \$x_1x_2 \ldots x_{l^\top}$, i.e., $s$ becomes an open-ended sequence. Note that the removal of & from $s$ does not affect the construction of the PST.

Given the above assumption, we prove the privacy guarantee of the modified PrivTree as follows.

THEOREM 4.1. *Let $\beta = |\mathcal{I}|+1$. The modified PrivTree ensures $\varepsilon$-differential privacy when $\lambda \geq \frac{2\beta-1}{\beta-1} \cdot \frac{l^\top}{\varepsilon}$ and $\delta = \lambda \cdot \ln \beta$.*

In addition, we prove that the postprocessing of PrivTree's output (i.e., adding Laplace noise to the histogram counts of the leaf nodes) also achieves $\varepsilon$-differential privacy.

THEOREM 4.2. *Postprocessing PrivTree's output with Laplace noise of scale $\lambda$ achieves $\varepsilon$-differential privacy, when $\lambda \geq \frac{l^\top}{\varepsilon}$.*

Finally, we clarify how we set $\theta$ and divide the privacy budget $\varepsilon$ between PrivTree and its postprocessing step. First, we set $\theta = 0$, following our analysis in Section 3.4. Second, we set the noise scale in PrivTree and its postprocessing step, such that PrivTree achieves $\frac{\varepsilon}{\beta}$-differential privacy and the postprocessing procedure ensures $\frac{\varepsilon\cdot(\beta-1)}{\beta}$-differential privacy. To explain, recall that in PrivTree, we inject Laplace noise into each node $v$'s score $c(v)$, which equals the sum of $\beta - 1$ counts in $v$'s prediction histogram *hist*$(v)$ (i.e., all counts except the largest one). Meanwhile, in the postprocessing step, we add Laplace noise to each count $y$ in the prediction histograms of $\mathcal{T}$'s leaf nodes. Intuitively, $c(v)$ is roughly $\beta - 1$ times more resilient to noise than $y$. Therefore, we set the privacy budget for the postprocessing step to be $\beta - 1$ times the budget for PrivTree, so as to balance the relative accuracy of $c(v)$ and $y$ after noise injection.

## 4.3 Comparison with Previous Work

There exist two differentially private methods [6, 7] for modeling sequence data, and they both utilize hierarchical decompositions for model construction. However, they considerably differ from PrivTree in three aspects. First, they model sequences based on their prefixes [7] or $n$-grams [6], while PrivTree is based on a PST representation of the *variable length Markov chain model* [44]. Second, their algorithms for hierarchical decompositions are similar in spirit to Algorithm 1, due to which they also require a predefined threshold $h$ on the maximum height of the decomposition tree. Consequently, they suffer from similar deficiencies to those of Algorithm 1, i.e., they cannot generate accurate models because of the dependency on $h$. Third, when constructing a decomposition tree, the methods in [6, 7] decide whether a node be split based only on a count associated with the node, whereas PrivTree adopts a more advanced strategy that takes into account three conditions commonly considered in the non-private setting. The above differences make PrivTree an effective approach for modeling sequence data, as we demonstrate in our experiments in Section 6.

## 5. CONNECTIONS TO SVT

In this section, we investigate the connection between PrivTree and the *sparse vector techniques (SVTs)* [18, 21, 28], which are a type of differentially private algorithms widely adopted in the literature. They take as input a sequence of queries and a threshold

[2]Such $l^\top$ can be chosen by first identifying the 90% or 95% quantile of the sequence lengths in $D$, and then computing a differentially private version of the quantile [54].

**Algorithm 3: BinarySVT** ($D$, $Q = \{q_1, q_2, \ldots\}$, $\theta$, $\lambda$)

```
1 compute a noisy version of $\theta$:  $\hat{\theta} = \theta + Lap(\lambda)$;
2 for $i = 1, 2, \ldots$ do
3     compute a noisy version of $q_i(D)$:  $\hat{q}_i(D) = q_i(D) + Lap(\lambda)$;
4     if $\hat{q}_i(D) > \hat{\theta}$ then
5         output $o_i = 1$ and continue;
6     else
7         output $o_i = 0$ and continue;
8 return
```

$\theta$, and output either a set of queries whose results are likely to be larger than $\theta$ [18, 28], or a noisy version of the answers for such a query set [21]. Intuitively, SVTs are similar in spirit to PrivTree since they both aim to identify some elements in a set (e.g., a node set or a query set) with "scores" above a given threshold. Motivated by this, in the following, we examine whether SVTs can be adopted for the hierarchical decomposition problem. Among the three existing variants of SVTs [18,21,28] that satisfy $\varepsilon$-differential privacy[3], we will focus on a variant dubbed *the binary SVT*, as it is most relevant to our problem. Interested readers are referred to the technical report of this paper for discussions on the other two variants.

Algorithm 3 presents a generic version of the binary SVT. Its input includes (i) a dataset $D$, (ii) a sequence $Q = \{q_1, q_2, \ldots\}$ of queries such that each $q_i$ has sensitivity 1, (iii) a threshold $\theta$, and (iv) a noise scale $\lambda$. Its output is a sequence of binary variables $\{o_1, o_2, \ldots\}$, such that $o_i = 1$ indicates that the result of $q_i$ is larger than $\theta$, and $o_i = 0$ indicates otherwise. The algorithm is fairly simple. It first computes a noisy threshold $\hat{\theta} = \theta + Lap(\lambda)$, and then, for each query $q_i$ in the sequence, it generates a noisy query answer $\hat{q}_i(D)$ using Laplace noise of scale $\lambda$ (Line 1-3). If $\hat{q}_i(D) > \theta$, then algorithm outputs $o_i = 1$; otherwise, the algorithm outputs $o_i = 0$ (Line 4-7). Previous work [28] makes the following claim about the privacy assurance of the algorithm:

CLAIM 1. *Algorithm 3 ensures $\varepsilon$-differential privacy if $\lambda \geq \frac{2}{\varepsilon}$.*

In other words, the noise scale required by the algorithm is $\Theta(\frac{1}{\varepsilon})$ and independent of the number of queries.

If Claim 1 holds, Algorithm 3 could yield highly competitive solutions for the problems that we consider. For example, consider the spatial decomposition problem studied in Section 3. Given a threshold $\theta$ and a set $D$ of spatial points in a multi-dimensional space $\Omega$, we first initialize (i) a quadtree $\mathcal{T}$ containing only a root node $v_1$ with $dom(v_1) = \Omega$, and (ii) a query sequence $Q = \{c(v_1)\}$, i.e., $Q$ contains only one query that asks the number of points in $dom(v_1)$ (we will dynamically append queries to $Q$ during the construction of the quadtree). After the initialization, we invoke the binary SVT to inspect each query in $Q$ one by one; if the binary SVT outputs 1 for a query $c(v)$, then we split the node $v$ in $\mathcal{T}$, and append a query $c(v')$ to the end of $Q$ for each child node $v'$ of $v$. When all queries in $Q$ are inspected, we return the quadtree $\mathcal{T}$ obtained. By Claim 1, $\mathcal{T}$ ensures $\varepsilon$-differential privacy, as long as the binary SVT uses Laplace noise of scale $\lambda \geq \frac{2}{\varepsilon}$ when generating the noisy versions of $c(v_i)$. In contrast, PrivTree requires injecting Laplace noise of scale $\lambda \geq \frac{2\beta-1}{\beta-1} \cdot \frac{1}{\varepsilon} > \frac{2}{\varepsilon}$, which indicates that the solution based on the binary SVT is more favorable.

Unfortunately, we show that Claim 1 does not hold: in the worst case, Algorithm 3 requires $\lambda = \Omega(\frac{k}{\varepsilon})$ to achieve $\varepsilon$-differential privacy, where $k$ denotes the number of queries.

[3]There exist other variants of SVT that satisfy a relaxed version of $\varepsilon$-differential privacy [23]. We do not consider those variants.

LEMMA 5.1. *There exists a sequence $Q$ of $k$ count queries for which Algorithm 3 violates $\varepsilon$-differential privacy if $\lambda \leq \frac{k}{4\varepsilon}$.*

Lemma 5.1 invalidates all solutions based on the binary SVT, including those in previous work [9,28,34]. In concurrent work [11], Chen and Machanavajjhala present a similar analysis on the binary SVT, and also come to the conclusion that it is not differentially private. In the full version of this paper[4], we discuss the other two variants of SVT [18,21], and show that one of them [21] also violates differential privacy, while the other [18] does not yield a competitive solution for our problem (even after we improve it with an optimization that leads to better data utility).

## 6. EXPERIMENTS

This section evaluates PrivTree against the states of the art on differentially private modelling of spatial and sequence data.

### 6.1 Experiments on Spatial Data

**Datasets.** We make use of four real spatial datasets shown in Table 2: road [12, 41], where each point represents the latitude and longitude of a road junction in the states of Washington and New Mexico; Gowalla [41,48], which contains check-in locations shared by users on a location-based social networking website; NYC[5] and Beijing[6], which are 4-dimensional datasets that record the pickup and drop-off locations of NYC and Beijing taxis, respectively. Figure 4 visualizes the points in road and gowalla, as well as the pickup locations in NYC and Beijing. Observe that the data distribution in road (resp. NYC) is more skewed than that in Gowalla (resp. Beijing).

**Methods.** We compare PrivTree against five state-of-the-art methods: UG [41, 42, 48], AG [41], Hierarchy [42], DAWA [30], and Privelet* [50]. UG partitions the data domain into $m^d$ grid cells of equal size, and releases a noisy count for each cell, with $m = (n\varepsilon/10)^{2/(d+2)}$ [48]. AG is an improved version of UG that is specifically designed for two-dimensional data. It first employs a coarsened version of UG to produce a set of grid cells; after that, for each cell whose noisy count is above a threshold, AG further splits it into smaller cells and releases their noisy counts. Hierarchy utilizes a multi-level decomposition tree to generate spatial histograms, with the tree height and fanout heuristically chosen to minimize the mean squared error in answering range count queries. DAWA requires as input a workload of range count queries, and it employs the *matrix mechanism* [31] to generate a histogram that is optimized for the given workload. Privelet* publishes multi-dimensional datasets by utilizing the Haar wavelet transformation to reduce the errors of range count queries.

DAWA and Privelet* both require that the input data should have a discrete domain. Following [30], we discretize the domain of each dataset into a uniform grid with $2^{20}$ cells before feeding it to DAWA and Privelet*. The other parameters of each method (e.g., the height and fanout of the decomposition tree, and the grid granularity) are set as suggested in the original papers. For PrivTree, we set its fanout to 4 (resp. 16) for two-dimensional (resp. four-dimensional) datasets, which is standard for quadtrees.

We adopt the implementations of DAWA and Privelet* provided by their respective authors, and we implement all other methods in C++. All experiments are conducted on a windows/linux machine with a 2.4GHz CPU and 16GB main memory.

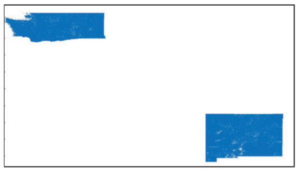
[4]http://arxiv.org/abs/1601.03229

[5]http://publish.illinois.edu/dbwork/open-data/

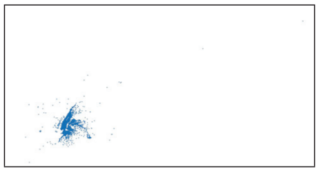
[6]http://research.microsoft.com/apps/pubs/?id=152883

**Table 2: Characteristics of spatial datasets.**

| Name | Dimensionality $d$ | Cardinality $n$ | Description |
|---|---|---|---|
| road | 2 | $1,634,165$ | Coordinates of road intersections in the states of Washington and New Mexico |
| Gowalla | 2 | $107,091$ | Check-in locations shared by users of a location-based social networking website |
| NYC | 4 | $98,013$ | Pickup and drop-off locations of NYC taxis |
| Beijing | 4 | $30,000$ | Pickup and drop-off locations of Beijing taxis |


(a) road

(b) Gowalla


(c) NYC - pickup

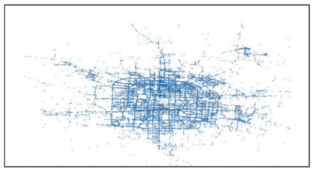
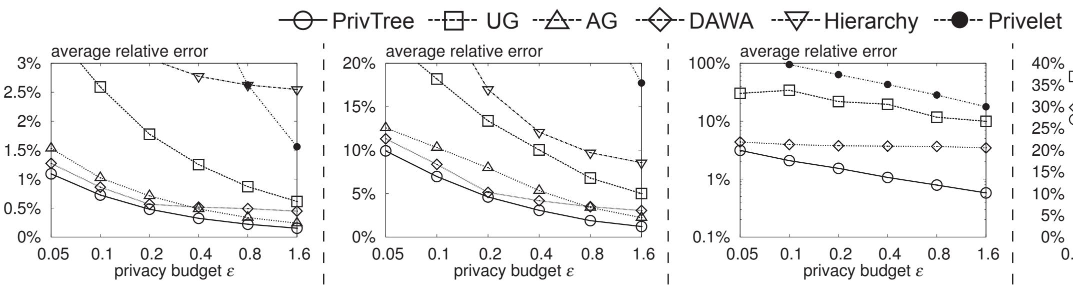
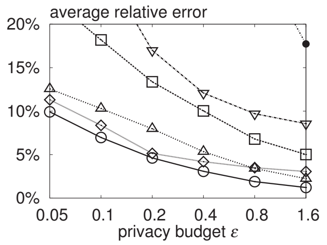

(d) Beijing - pickup

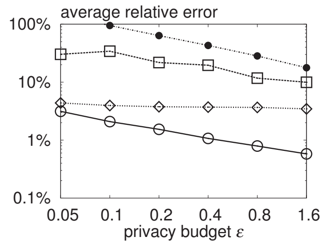
**Figure 4: Visualization of datasets**



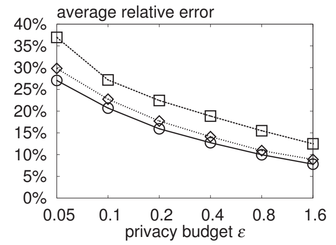
(a) road - small queries.



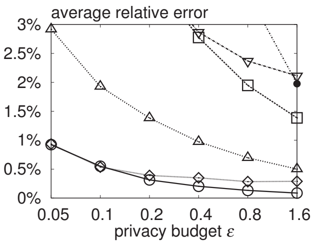
(d) Gowalla - small queries.



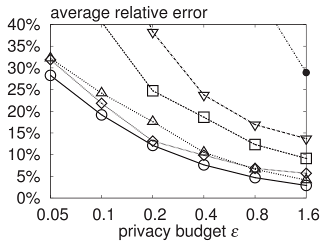
(g) NYC - small queries.



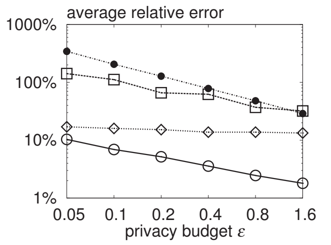
(j) Beijing - small queries.



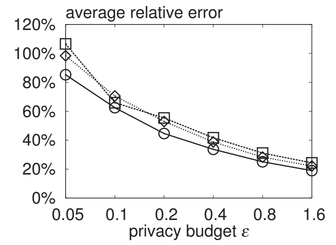
(b) road - medium queries.



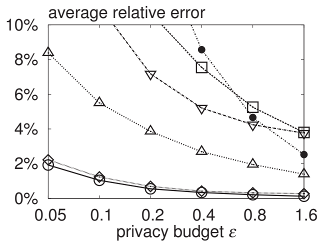
(e) Gowalla - medium queries.



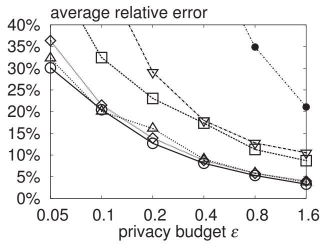
(h) NYC - medium queries.



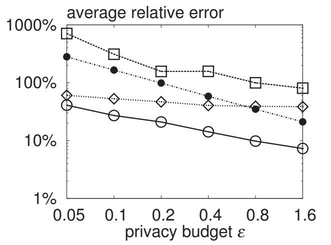
(k) Beijing - medium queries.



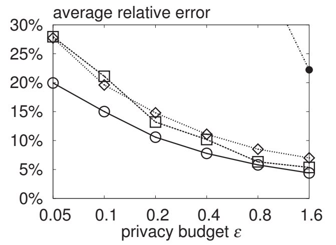
(c) road - large queries.



(f) Gowalla - large queries.



(i) NYC - large queries.



(l) Beijing - large queries.

**Figure 5: Results of range count queries on spatial datasets.**

**Tasks.** We apply each method to create private synopses of every dataset, and we evaluate the quality of each decomposition by the accuracy of its answers to range count queries. In particular, we construct three query sets on each dataset: *small*, *medium*, and *large*, each of which contains $10,000$ randomly generated range count queries. Each query in the small, medium, and large set has a region that cover $[0.01\%, 0.1\%)$, $[0.1\%, 1\%)$, and $[1\%, 10\%)$ of the data domain, respectively. Following prior work [12, 41], we measure accuracy of an answer $\hat{q}(D)$ to a query $q$ by its *relative error*, defined as

$$RE\left(\hat{q}(D)\right) = \frac{|\hat{q}(D) - q(D)|}{\max\left\{q(D), \Delta\right\}}.$$

where $\Delta$ is a smoothing factor set to $0.1\%$ of the dataset cardinality $n$ [41,50]. We repeat each experiment 100 times, and report the average relative error of each method for each query set. For DAWA (which is *query-dependent*), we allow it to generate a synopsis for each query set separately, based on a sample set of 500 queries.

**Results.** Figure 5 illustrates the average relative error of each method on each dataset as a function of the privacy budget $\varepsilon$. On road, PrivTree significantly outperforms UG, AG, Hierarchy, and

Table 3: Characteristics of sequence datasets.

| Name | $\lvert\mathcal{I}\rvert$ | Cardinality | Avg. sequence length | Description | $l^\top$ | # of sequences with length $> l^\top$ |
|---|---|---|---|---|---|---|
| mooc | 7 | $80,362$ | 13.46 | Users' behavior sequences on a MOOC website | 50 | 3,653 |
| msnbc | 17 | $989,818$ | 4.75 | Users' web navigation histories on a news portal | 20 | 31,606 |

Privelet$^*$, regardless of the query set used and the value of $\varepsilon$. In particular, on the large query set, the average relative error of PrivTree is at most $\frac{1}{4}$ (resp. $\frac{1}{10}$) of the error of AG (resp. UG and Hierarchy). This demonstrates the effectiveness of PrivTree in approximating the distribution of the input data. Meanwhile, AG is superior to UG and Hierarchy in all cases, which is consistent with the results in previous work [41]. DAWA is the only method that comes close to PrivTree, but its relative error is never smaller than that of PrivTree, and is 2 to 3 times higher than the latter for the small and medium query sets on road (resp. small query set on Gowalla) when $\varepsilon \geq 0.8$. Furthermore, we note that DAWA is given a sample query set in advance to optimize its query performance, whereas PrivTree is not given such an advantage.

On Gowalla, PrivTree still consistently achieves the best results, but the performance gaps between PrivTree and the other methods are reduced. The reason is that the data distribution in Gowalla is less skewed than that of road (see Figure 4), which makes Gowalla easier to dealt with for all methods. DAWA incurs relatively small errors in all cases, but is noticeably inferior to PrivTree on the small and medium query sets.

On NYC and Beijing, we omit AG and Hierarchy since (i) AG is only applicable on two-dimensional data, and (ii) when applied on a four-dimensional dataset, Hierarchy produces a decomposition tree with at least 2.18 billion leaf nodes [42], which cannot fit in the main memory of our machine. As shown in Figures 5g-5l, PrivTree consistently outperforms all other methods by a large margin on the highly skewed NYC, because its tree construction mechanism enables it to effectively adapt to the skewness of the data, by growing the tree tall (resp. short) in the dense (resp. sparse) regions of the data. On the other hand, on the less skewed Beijing, the accuracies of UG and DAWA are considerably improved. Nevertheless, PrivTree still incurs smaller query errors in all settings. One may notice that the error of DAWA on NYC only decreases around 2 times when $\varepsilon$ increases from 0.05 to 1.6. We find that it is caused by a "private partitioning" step of DAWA [30], as well as the discretization of data domain $\Omega$ that it requires.

In summary, PrivTree provides better data utility than all baselines, especially when the input dataset follows a skewed distribution. This makes PrivTree a more favorable approach for releasing spatial data under differential privacy.

## 6.2 Experiments on Sequence Data

**Datasets.** We use two real sequence datasets: mooc[7] and msnbc [1, 6]. mooc contains $80,362$ learners' behavior sequences on a MOOC platform, and the behaviors are divided into seven categories: working on assignments, watching videos, accessing other course objects, accessing the course wiki, accessing the course forum, navigating to other part of course, and closing the web page. msnbc consists of $989,818$ sequences of URL categories, each of which corresponds to a user's browsing history during a 24-hour period on *msnbc.com*. Table 3 shows the key statistics of mooc and msnbc. Note that the total number $|\mathcal{I}|$ of symbols in mooc (resp. msnbc) is not excessively large. Otherwise (e.g., when $|\mathcal{I}| > 1000$), the domain of the sequence data would be extremely sparse, in which case it is enormously difficult to publish useful information under differential privacy.

**Tasks.** We consider two analytical tasks on each sequence dataset $D$. The first task is to identify the top-$k$ frequent strings in $D$, i.e., the $k$ strings that appear the largest number of times in the sequences in $D$. This task is an important primitive in sequence data mining [20], and is also considered in existing work [6] on sequence data publishing. Following previous work [6], we measure the *precision* of the top-$k$ strings returned by differentially private algorithm, i.e.,

$$\text{precision} = \frac{|K(D) \cap \mathcal{A}(D)|}{k},$$

where $K(D)$ is the exact set of top-$k$ frequent strings in $D$, and $\mathcal{A}(D)$ is the set returned by algorithm $\mathcal{A}$.

The second task is to approximate the distribution of sequence lengths in $D$. In particular, we apply PrivTree and other existing methods to generate synthetic sequence data from $D$. Then, we compare the distribution of sequence lengths in the synthetic data with that in $D$, and we measure their *total variation distance* [13], i.e., half of the $L_1$ distance between the two probability distributions. For each task, we repeat each experiment 100 times and report the average measurements.

**Methods.** For the task of top-$k$ frequent string mining, we compare PrivTree against two differentially private techniques: N-gram [6] and EM [38]. In particular, N-gram is the state-of-the-art solution for sequence data publishing, and it is based on a variable-length $n$-gram model. N-gram requires a pre-defined threshold $n_{max}$ on the maximum length of $n$-grams; we set $n_{max} = 5$, as suggested in [6]. Meanwhile, EM is a standard application of the exponential mechanism [38] in our context. It first initializes a set $R$ that contains $|\mathcal{I}|$ string of length 1, each of which consists of a unique symbol in $\mathcal{I}$. After that, it invokes the exponential mechanism $k$ times. In each invocation, it selects the most frequent string $r$ from $R$ with differential privacy, and then replaces $r$ in $R$ with $|\mathcal{I}|$ strings, each of which is obtained by adding a symbol to the end of $r$. The $k$ strings obtained are then returned as the result. For the task of approximating sequence length distributions, we omit EM since it is inapplicable.

Note that PrivTree, N-gram, and EM all require that the maximum sequence length in the input data is bounded by a constant $l^\top$ that is not excessively large (see Section 4.2 for a discussion on the necessity of $l^\top$). Following previous work [6], we set $l^\top$ to be roughly the 95% quantile of the sequence lengths in the input data, i.e., only around 5% sequences are truncated (see Table 3). To illustrate the effects of truncation, we also include in our experiments a baseline approach dubbed *Truncate*. This approach directly answers all queries on the truncated dataset, without any privacy assurance.

**Results.** Figure 6 shows the precision of each method (for top-$k$ string mining) as a function of the privacy budget $\varepsilon$. The precision of Truncate remains unchanged for all $\varepsilon$, since it does not enforce differential privacy. Among the differentially private methods, PrivTree consistently outperforms N-gram and EM, in most cases by a large margin. Furthermore, in Figure 6d-6f, PrivTree has

[7]https://www.kddcup2015.com/

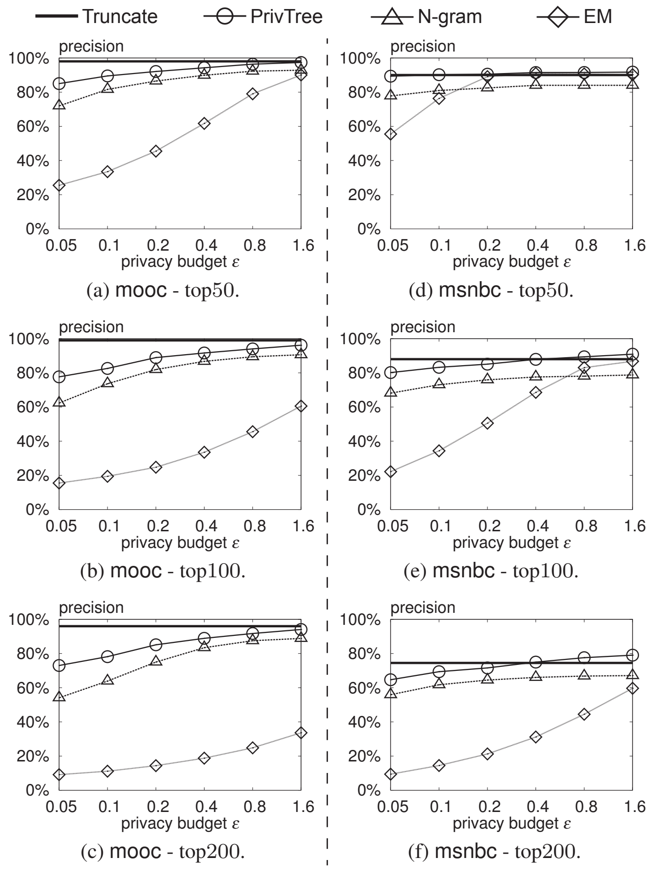


Figure 6: Results of top-$k$ frequent string mining.

an even higher precision than Truncate when $\varepsilon \geq 0.8$. The reason is that the Markov model adopted by PrivTree is able to *recover* some information that is lost due to the truncation of sequences. For example, suppose that the string $aa$ appears in 5 sequences in a dataset $D$ and, in each appearance, it is immediately followed by a symbol $b$. Assume that one of those 5 sequences (denoted as $s$) is truncated, and its suffix $aab$ becomes $aa$ after the truncation. In that case, the Markov model can be used able to accurately recover the truncated symbol of $s$, because, based on the truncated data, it would predict that the "next symbol" after $aa$ is always $b$. Intuitively, such recovering of information is more effective when the amount of noise in the Markov model is small, which explains why PrivTree outperforms Truncate only when $\varepsilon$ is large. In contrast, N-gram never outperforms Truncate, and its precision is lower than that of PrivTree by more than 10% in most settings. In addition, EM yields unattractive precision in almost all cases. Its accuracy degrades with the increases of $k$, since a larger $k$ requires it to inject more noise into the selection procession of top-$k$ frequent strings.

In the last set of experiments, we evaluate the accuracy of the sequence length distribution in the synthetic sequence data generated by each method. Figure 7 illustrates the total variation distance of each sequence length distribution. Observe that PrivTree incurs a small error comparable to that of Truncate, especially when $\varepsilon \geq 0.2$. In contrast, N-gram entails an enormous error in all cases. Based on the results in Figures 6 and 7, we conclude that PrivTree is a more preferable solution than N-gram to modeling sequential data under $\varepsilon$-differential privacy.

## 7. ADDITIONAL RELATED WORK

In Sections 3.1, 4.3, and 6, we have introduced the states-of-the-art solutions [6, 12, 30, 41, 42, 48, 50] for publishing spatial and sequence data under differential privacy. Besides those solutions, there are a few other methods for private modeling of spatial and sequence data. In particular, Xiao et al. [51] present a spatial decomposition algorithm based on the $k$-d tree [4]. It first imposes a uniform grid over the data domain, and then construct a private $k$-d tree over the cells in the grid. This method, however, is shown to be inferior to the UG and AG methods tested in our experiments, in terms of data utility [41]. Chen et al. [7] consider the publication of sequence data under differential privacy, and propose an algorithm that releases a prefix tree of sequences to support count queries and frequent string mining. Nevertheless, subsequent work by Chen et al. [6] shows that the prefix-based method is considerably outperformed by the N-gram approach in our experiment.

In addition, there is a long line of research on processing aggregate queries in a differentially private manner. Specifically, Barak et al. [2] investigate the publication of *marginals* (i.e., projections of a dataset on subsets of its dimensions), and propose a solution based on the Fourier transform. Ding et al. [15] publish multiple data cubes with both privacy and consistency guarantees. The *matrix mechanism* of Li and Miklau [32, 33] and follow-up approaches [22, 30, 52, 53] take into account a query workload, and aim to release a version of the data that maximizes the overall accuracy of the workload. DAWA [30] is the most advanced method among these approaches, but as shown in our experiments, it is outperformed by PrivTree in terms of the relative errors of range count queries on spatial data.

Moreover, there exists extensive work that addresses numerous other tasks under differential privacy, such as regressions [5, 27, 40, 47, 55, 57], clusterings [48], decision trees [19], recommendation systems [37], time-series data analysis [43], combinatorial optimizations [49], frequent itemset mining [35], and graph queries [10, 26, 36, 56]. Finally, recent research has also studied the adoption of differential privacy in various systems [8, 39, 45].

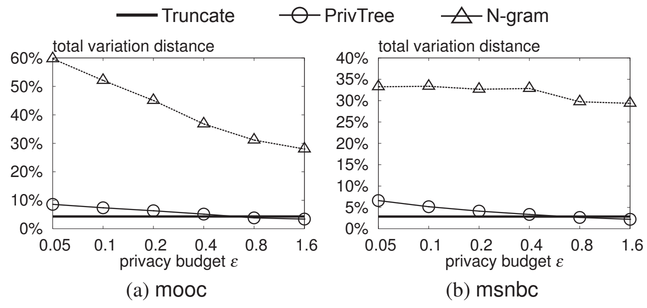


Figure 7: Errors of sequence length distributions.

## 8. CONCLUDING REMARKS

In this paper, we study the problem of hierarchical decomposition under differential privacy, and address the central dilemma of choosing the maximum height $h$ of the decomposition tree. We show that the constraint on $h$ can be removed by introducing a carefully controlled bias in deciding when a node should be split. Based on this result, we propose PrivTree, a general approach for hierarchical decomposition on private data, and we showcase its applications on spatial and sequence data release. Our experimental results demonstrate that PrivTree significantly outperforms the states of the art in terms of data utility. For future work, we plan to extend the idea behind PrivTree to other problems that are based on a lattice-model instead of a tree-model, such as frequent itemset mining.

## Acknowledgments

Xiaokui Xiao was supported by grant ARC19/14 from MOE, Singapore and gifts from MSRA and AT&T.

## 9. REFERENCES

[1] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[2] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar. Privacy, accuracy, and consistency too: a holistic solution to contingency table release. In *PODS*, pages 273–282, 2007.

[3] R. Begleiter, R. El-Yaniv, and G. Yona. On prediction using variable order markov models. *Journal of Artificial Intelligence Research*, pages 385–421, 2004.

[4] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[5] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12:1069–1109, 2011.

[6] R. Chen, G. Acs, and C. Castelluccia. Differentially private sequential data publication via variable-length n-grams. In *CCS*, pages 638–649, 2012.

[7] R. Chen, B. Fung, B. C. Desai, and N. M. Sossou. Differentially private transit data publication: a case study on the montreal transportation system. In *SIGKDD*, pages 213–221. ACM, 2012.

[8] R. Chen, A. Reznichenko, P. Francis, and J. Gehrke. Towards statistical queries over distributed private user data. In *NSDI*, pages 169–182, 2012.

[9] R. Chen, Q. Xiao, Y. Zhang, and J. Xu. Differentially private high-dimensional data publication via sampling-based inference. In *SIGKDD*. ACM, 2015.

[10] S. Chen and S. Zhou. Recursive mechanism: Towards node differential privacy and unrestricted joins. In *SIGMOD*, pages 653–664, 2013.

[11] Y. Chen and A. Machanavajjhala. On the privacy properties of variants on the sparse vector technique. *CoRR*, abs/1508.07306, 2015.

[12] G. Cormode, C. M. Procopiuc, E. Shen, D. Srivastava, and T. Yu. Differentially private spatial decompositions. In *ICDE*, 2012.

[13] A. B. Cybakov. *Introduction to nonparametric estimation*. Springer, 2009.

[14] M. De Berg, M. Van Kreveld, M. Overmars, and O. C. Schwarzkopf. *Computational geometry*. Springer, 2000.

[15] B. Ding, M. Winslett, J. Han, and Z. Li. Differentially private data cubes: optimizing noise sources and consistency. In *SIGMOD*, pages 217–228, 2011.

[16] C. Dwork. Differential privacy. In *ICALP*, pages 1–12, 2006.

[17] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006.

[18] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Theoretical Computer Science*, 9(3-4):211–407, 2013.

[19] A. Friedman and A. Schuster. Data mining with differential privacy. In *KDD*, pages 493–502, 2010.

[20] J. Han, M. Kamber, and J. Pei. *Data mining: concepts and techniques: concepts and techniques*. Elsevier, 2011.

[21] M. Hardt. *A study of privacy and fairness in sensitive data analysis*. PhD thesis, Princeton University, 2011.

[22] M. Hardt, K. Ligett, and F. McSherry. A simple and practical algorithm for differentially private data release. In *NIPS*, pages 2348–2356, 2012.

[23] M. Hardt and G. N. Rothblum. A multiplicative weights mechanism for privacy-preserving data analysis. In *FOCS*, pages 61–70, 2010.

[24] M. Hardt and K. Talwar. On the geometry of differential privacy. In *STOC*, pages 705–714. ACM, 2010.

[25] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *PVLDB*, 3(1):1021–1032, 2010.

[26] V. Karwa, S. Raskhodnikova, A. Smith, and G. Yaroslavtsev. Private analysis of graph structure. *TODS*, 39(3):22, 2014.

[27] D. Kifer, A. D. Smith, and A. Thakurta. Private convex optimization for empirical risk minimization with applications to high-dimensional regression. *Journal of Machine Learning Research - Proceedings Track*, 23:25.1–25.40, 2012.

[28] J. Lee and C. W. Clifton. Top-$k$ frequent itemsets via differentially private fp-trees. In *SIGKDD*, pages 931–940. ACM, 2014.

[29] J. Lei. Differentially private m-estimators. In *NIPS*, 2011.

[30] C. Li, M. Hay, G. Miklau, and Y. Wang. A data-and workload-aware algorithm for range queries under differential privacy. *PVLDB*, 7(5):341–352, 2014.

[31] C. Li, M. Hay, V. Rastogi, G. Miklau, and A. McGregor. Optimizing linear counting queries under differential privacy. In *PODS*, pages 123–134, 2010.

[32] C. Li and G. Miklau. An adaptive mechanism for accurate query answering under differential privacy. *PVLDB*, 5(6):514–525, 2012.

[33] C. Li and G. Miklau. Optimal error of query sets under the differentially-private matrix mechanism. In *ICDT*, pages 272–283, 2013.

[34] H. Li, L. Xiong, X. Jiang, and J. Liu. Differentially private histogram publication for dynamic datasets: an adaptive sampling approach. In *CIKM*, pages 1001–1010, 2015.

[35] N. Li, W. Qardaji, D. Su, and J. Cao. Privbasis: Frequent itemset mining with differential privacy. *PVLDB*, 5(11):1340–1351, 2012.

[36] W. Lu and G. Miklau. Exponential random graph estimation under differential privacy. In *KDD*, pages 921–930, 2014.

[37] F. McSherry and I. Mironov. Differentially private recommender systems: Building privacy into the netflix prize contenders. In *KDD*, pages 627–636, 2009.

[38] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *FOCS*, pages 94–103, 2007.

[39] A. Narayan and A. Haeberlen. Djoin: Differentially private join queries over distributed databases. In *OSDI*, pages 149–162, 2012.

[40] K. Nissim, S. Raskhodnikova, and A. Smith. Smooth sensitivity and sampling in private data analysis. In *STOC*, pages 75–84, 2007.

[41] W. Qardaji, W. Yang, and N. Li. Differentially private grids for geospatial data. In *ICDE*, pages 757–768, 2013.

[42] W. Qardaji, W. Yang, and N. Li. Understanding hierarchical methods for differentially private histograms. *PVLDB*, 6(14):1954–1965, 2013.

[43] V. Rastogi and S. Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *SIGMOD*, pages 735–746, 2010.

[44] D. Ron, Y. Singer, and N. Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine learning*, 25(2-3):117–149, 1996.

[45] I. Roy, S. T. V. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and privacy for mapreduce. In *SIGMOD*, pages 297–312, 2010.

[46] H. Samet. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.

[47] A. Smith. Privacy-preserving statistical estimation with optimal convergence rate. In *STOC*, 2011.

[48] D. Su, J. Cao, N. Li, E. Bertino, and H. Jin. Differentially private $k$-means clustering. *arXiv preprint arXiv:1504.05998*, 2015.

[49] K. Talwar, A. Gupta, K. Ligett, F. McSherry, and A. Roth. Differentially private combinatorial optimization. *SODA*, 2010.

[50] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. *TKDE*, 23(8):1200–1214, 2011.

[51] Y. Xiao, L. Xiong, and C. Yuan. Differentially private data release through multidimensional partitioning. In *Secure Data Management*, pages 150–168. Springer, 2010.

[52] G. Yaroslavtsev, G. Cormode, C. M. Procopiuc, and D. Srivastava. Accurate and efficient private release of datacubes and contingency tables. In *ICDE*, pages 745–756, 2013.

[53] G. Yuan, Z. Zhang, M. Winslett, X. Xiao, Y. Yang, and Z. Hao. Low-rank mechanism: Optimizing batch queries under differential privacy. *PVLDB*, 5(11):1352–1363, 2012.

[54] C. Zeng, J. F. Naughton, and J.-Y. Cai. On differentially private frequent itemset mining. *PVLDB*, 6(1):25–36, 2012.

[55] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. Privbayes: Private data release via bayesian networks. pages 1423–1434, 2014.

[56] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. Private release of graph statistics using ladder functions. In *SIGMOD*, pages 731–745, 2015.

[57] J. Zhang, X. Xiao, Y. Yang, Z. Zhang, and M. Winslett. PrivGene: differentially private model fitting using genetic algorithms. In *SIGMOD*, pages 665–676, 2013.

## APPENDIX

## A. PROOFS

**Proof of Lemma 3.1.** By Equations (1) and (5), for any $x$,

$$\rho(x) = \ln\left(\frac{\int_{\theta-x}^{+\infty}\frac{1}{2\lambda}\exp\left(-\frac{|y|}{\lambda}\right)dy}{\int_{\theta+1-x}^{+\infty}\frac{1}{2\lambda}\exp\left(-\frac{|y|}{\lambda}\right)dy}\right) \le \frac{1}{\lambda}.$$

Recall that $\rho^\top(x) = 1/\lambda$ when $c(v) < \theta + 1$. Therefore, $\rho(x) \le \rho^\top(x)$ holds if $x < \theta + 1$.

Now consider that $x \ge \theta + 1$. In that case,

$$\rho(x) = \ln\left(\frac{1-\frac{1}{2}\exp\left(\frac{\theta-x}{\lambda}\right)}{1-\frac{1}{2}\exp\left(\frac{\theta+1-x}{\lambda}\right)}\right).$$

For convenience, we let $\alpha = \frac{1}{2}\exp\left(\frac{\theta+1-x}{\lambda}\right)$, and define a function $f$ of $\alpha$ as follows:

$$\begin{aligned}f(\alpha) &= \rho(x) - \frac{1}{\lambda}\exp\left(\frac{\theta+1-x}{\lambda}\right)\\ &= \ln\left(\frac{1-\alpha e^{-1/\lambda}}{1-\alpha}\right) - \frac{2\alpha}{\lambda}.\end{aligned}$$

Observe that $\alpha \in (0, 1/2]$ whenever $x \ge \theta + 1$. Therefore, we can prove the lemma by showing that $f(\alpha) \le 0$ for all $\alpha \in (0, 1/2]$. For this purpose, we first compute the second derivative of $f$ with respect to $\alpha$:

$$f''(\alpha) \;=\; (e^{1/\lambda} - 1)\cdot\frac{e^{1/\lambda} + 1 - 2\alpha}{(1-\alpha)^2\cdot(e^{1/\lambda}-\alpha)^2}.$$

Given that $e^{1/\lambda} - 1 > 0$ and $e^{1/\lambda} + 1 - 2\alpha \ge e^{1/\lambda} > 0$, we have $f''(\alpha) > 0$. This indicates that $\max_{\alpha\in(0,1/2]} f(\alpha) = \max\{f(0), f(1/2)\}$. Meanwhile, $f(0) = 0$, and

$$\begin{aligned}f(1/2) &= \ln(2 - e^{-1/\lambda}) - \frac{1}{\lambda}\\ &= \ln\left(e^{1/\lambda} - \left(e^{1/2\lambda} - e^{-1/2\lambda}\right)^2\right) - \frac{1}{\lambda}\\ &< \ln(e^{1/\lambda}) - \frac{1}{\lambda} = 0.\end{aligned}$$

Therefore, $\max_{\alpha\in(0,1/2]} f(\alpha) \le 0$, which proves the lemma. □

**Proof of Theorem 3.1.** The theorem directly follows from the privacy analysis in Section 3.3. □

**Proof of Lemma 3.2.** Let $V$ be the set of all possible nodes in a quadtree built on $D$. We divide the nodes in $V$ into three subsets: (i) the set $V_1$ of nodes that appear as non-leaf nodes in $\mathcal{T}^*$, (ii) the set $V_2$ of the nodes that appear as leaves in $\mathcal{T}^*$, and (iii) the set $V_3$ of the nodes that do not appear in $\mathcal{T}^*$. Let $g(S)$ be the expected number of nodes in a set $S$ that appear in $\mathcal{T}$. Then, we have

$$\begin{aligned}\mathbb{E}[|\mathcal{T}|] &= g(V_1) + g(V_2) + g(V_3)\\ &\le |V_1| + |V_2| + g(V_3) \;=\; |\mathcal{T}^*| + g(V_3).\end{aligned}$$

Therefore, the lemma can be proved by showing $g(V_3) \le |\mathcal{T}^*|$.

Observe that each node in $V_3$ must be the descendant of a node in $V_2$, i.e., a node that appears as a leaf in $\mathcal{T}^*$. Therefore, we can divide the nodes in $V_3$ into $|V_2|$ subsets, such that all nodes in the same subset are descendants of the same node in $V_2$. Consider any such subset $S$ that corresponds to a node $v \in V_2$. Given that $v$ appears as a leaf in $\mathcal{T}^*$, we have $c(v) \le \theta$. In addition, since $|\mathcal{T}^*| > 1$, $depth(v) \ge 1$ holds. Therefore,

$$c(v) - depth(v)\cdot\delta \;\le\; c(v) - \delta \;\le\; \theta - \delta.$$

By Equation (8), we have $b(v) = \theta - \delta$. Furthermore, for any $v' \in S$, we have $c(v') \le c(v)$, which also leads to $b(v') = \theta - \delta$. Therefore, $v$ and $v'$ have the same probability $p_s$ to be split. Given $\delta = \lambda\cdot\ln\beta$, we have

$$\begin{aligned}p_s &= \Pr[\theta - \delta + Lap(\lambda) > \theta] \;=\; \Pr[Lap(\lambda) > \delta]\\ &= \int_{\lambda\cdot\ln\beta}^{\infty}\frac{1}{2\lambda}\exp\left(-\frac{|y|}{\lambda}\right)dy \;=\; \frac{1}{2\beta}.\end{aligned}$$

Assume that $v$ appears in $\mathcal{T}$. Then, given that $v$ is split with $\frac{1}{2\beta}$ probability, each child of $v$ has $\frac{1}{2\beta}$ probability to appear in $\mathcal{T}$. Therefore, in expectation, the number of $v$'s children that appear in $\mathcal{T}$ should equal $\beta\cdot\frac{1}{2\beta} = \frac{1}{2}$. In general, for any $i \ge 1$, there exist $\beta^i$ nodes $v' \in S$ with $depth(v') - depth(v) = i$, and each such $v'$ has $(\frac{1}{2\beta})^i$ probability to appear in $\mathcal{T}$. Hence, the expected number of nodes in $S$ that appear in $\mathcal{T}$ is

$$g(S) \;=\; \sum_{i=1}^{+\infty}\beta^i\cdot\left(\frac{1}{2\beta}\right)^i \;=\; \sum_{i=1}^{+\infty}\frac{1}{2^i} \;=\; 1.$$

Since each $S$ uniquely corresponds to a node in $V_2$, we have

$$g(V_3) \;=\; |V_2|\cdot g(S) \;= |V_2| \;\le\; |\mathcal{T}^*|.$$

Therefore, the lemma is proved. □

**Proof of Corollary 1.** The corollary follows from Theorem 3.1 when $\gamma = \ln\beta$. □

**Proof of Lemma 4.1.** Let $u$ and $v$ be two nodes in $\mathcal{T}$, such that $u$ is the parent of $v$. Then, $hist(v)[x] \le hist(u)[x]$ holds for every symbol $x \in \mathcal{I}\cup\{\&\}$. Let $x_v$ (resp. $x_u$) be the symbol that has the largest count in $hist(v)$ (resp. $hist(u)$). We have

$$\begin{aligned}c(v) &= \|hist(v)\|_1 - hist(v)[x_v] \;\le\; \|hist(v)\|_1 - hist(v)[x_u]\\ &= \sum_{x\neq x_u} hist(v)[x] \;\le\; \sum_{x\neq x_u} hist(u)[x]\\ &= \|hist(u)\|_1 - hist(u)[x_u] \;=\; c(u).\end{aligned}$$

Therefore, $c(\cdot)$ is monotonic. □

**Proof of Theorem 4.1.** Let $D$ and $D'$ be two neighboring datasets, such that $D$ is obtained by inserting a sequence $s$ into $D'$. Assume that $s = \$x_1\ldots x_l$, where $x_i \in I\cup\{\&\}$ for $i \in [1, l]$. To facilitate our proof, we define $l$ datasets $D_1, D_2, \ldots, D_l$, such that $D_i = D'\cup\{s_i\}$ and $s_i = \$x_1x_2\ldots x_i$ is the length-$i$ prefix of $s$ ended at symbol $x_i$. Observe that $D_l = D$. For convenience, we define $D_0 = D'$.

In the following, we will prove that for any $i \in [1, l]$ and any output $\mathcal{T}$ of the modified PrivTree,

$$-\frac{\varepsilon}{l^\top} \;\le\; \ln\left(\frac{\Pr[D_i \to \mathcal{T}]}{\Pr[D_{i-1}\to\mathcal{T}]}\right) \;\le\; \frac{\varepsilon}{l^\top}, \tag{14}$$

where $\Pr[D_i \to \mathcal{T}]$ denotes the probability that PrivTree outputs $\mathcal{T}$ given $D_i$. This would prove the theorem because, given that $l \le l^\top$,

$$\ln\left(\frac{\Pr[D \to \mathcal{T}]}{\Pr[D' \to \mathcal{T}]}\right) = \sum_{i=1}^{l} \ln\left(\frac{\Pr[D_i \to \mathcal{T}]}{\Pr[D_{i-1} \to \mathcal{T}]}\right) \in [-\varepsilon, \varepsilon].$$

Observe that $D_i$ can be obtained by appending a symbol $x_i$ to the end of the sequence $s_{i-1}$ in $D_{i-1}$. Therefore, when we change the input data from $D_{i-1}$ to $D_i$, the only changes in the PST are the histogram counts that $x_i$ contributes to. Observe that if $x_i$ contributes to the prediction histogram $hist(v)$ of a node $v$, then $dom(v)$ must be a suffix of $s_{i-1}$, and the only possible change in $hist(v)$ is that $hist(v)[x_i]$ would be increased by one. Then, by the definition of the PST, all of those nodes $v$ should form a path from the root of the PST to a leaf. In addition, by Equation (13), the score $c(v)$ of each of those nodes $v$ is changed by at most one. In that case, we can prove Equation (14) by reusing the analysis in the proof of Theorem 3.1.

To explain, recall that the correctness of Theorem 3.1 only replies on two conditions. First, the score $c(v)$ of each node $v$ is monotonic. Second, when we change the input data, all of the nodes affected should form a path from the root of the decomposition tree to a leaf, and the score of each of those nodes should change by at most one. Notice that all three conditions are satisfied when we change the input of the modified PrivTree from $D_{i-1}$ to $D_i$. Combining this with the fact that the modified PrivTree uses a noise scale that is $l^\top$ times that of Algorithm 2, it can be verified that Equation (14) holds. Therefore, the theorem is proved. □

**Proof of Theorem 4.2.** Let $\mathcal{T}$ be the output of the modified PrivTree. Let $D$ and $D'$ be two neighboring datasets, such that $D$ is obtained by inserting a sequence $s$ into $D'$. Assume that $s = \$x_1 \ldots x_l$, where $x_i \in I \cup \{\&\}$ for $i \in [1, l]$. Observe that each symbol $x_i$ in $s$ contributes to the prediction histograms of the nodes whose predictor strings $dom(\cdot)$ are suffixes of $\$x_1 \ldots x_{i-1}$. By the definition of PSTs, these nodes form a path from the root of $\mathcal{T}$ to a leaf. This indicates that each $x_i$ gets counted in the histogram of *one* leaf node only. Taking into account $l \le l^\top$, it follows that the sensitivity of releasing the histogram counts of all leaf nodes in $\mathcal{T}$ is $l^\top$. By the property of the Laplace mechanism, the postprocessing step ensures $\varepsilon$-differential privacy if $\lambda \ge \frac{l^\top}{\varepsilon}$. □

**Proof of Lemma 5.1.** Consider three datasets $D_1 = \{a, b\}$, $D_2 = \{a, b, b\}$, and $D_3 = \{b, b\}$, where each tuple is either $a$ or $b$. Observe that $D_1$ is a neighboring dataset of $D_2$, while $D_2$ is a neighboring dataset of $D_3$. Let $q_a$ (resp. $q_b$) be a query that asks for the number of $a$ (resp. $b$) in a dataset. Let $Q$ be a sequence of $k$ queries, such that first $k/2$ queries are all $q_a$, and the remaining $k/2$ queries are all $q_b$.

Suppose that we invoke Algorithm 3 on $D_1, D_2, D_3$, respectively, with $Q$, a noise scale $\lambda$, and a threshold $\theta = 1$. Let $E$ be the event that Algorithm 3 outputs 1 for the first $k/2$ queries in $Q$, and 0 for the remaining $k/2$ queries. In addition, let $\Pr[D \to E]$ denote the probability that $E$ occurs when the input dataset is $D$. If Algorithm 3 satisfies $\varepsilon$-differential privacy, then

$$\frac{\Pr[D_1 \to E]}{\Pr[D_2 \to E]} \le e^{\varepsilon}, \qquad \frac{\Pr[D_2 \to E]}{\Pr[D_3 \to E]} \le e^{\varepsilon}.$$

This indicates that

$$\frac{\Pr[D_1 \to E]}{\Pr[D_3 \to E]} = \frac{\Pr[D_1 \to E]}{\Pr[D_2 \to E]} \cdot \frac{\Pr[D_2 \to E]}{\Pr[D_3 \to E]} \le e^{2\varepsilon}. \quad (15)$$

In what follows, we prove the lemma by showing that Equation (15) does not hold when $\lambda \le k/4\varepsilon$.

Recall that Algorithm 3 generates a noisy threshold $\hat{\theta}$, and outputs 1 for a query $q$ only when its noisy answer $\hat{q}(D)$ is larger than $\hat{\theta}$. Therefore,

$$\begin{aligned}
&\frac{\Pr[D_1 \to E]}{\Pr[D_3 \to E]} \\
&= \frac{\int_{-\infty}^{\infty} \Pr[\hat{\theta} = x] \cdot \left(\Pr[\hat{q_a}(D_1) > x] \cdot \Pr[\hat{q_b}(D_1) \le x]\right)^{\frac{k}{2}} dx}{\int_{-\infty}^{\infty} \Pr[\hat{\theta} = x] \cdot \left(\Pr[\hat{q_a}(D_3) > x] \cdot \Pr[\hat{q_b}(D_3) \le x]\right)^{\frac{k}{2}} dx} \\
&= \frac{\int_{-\infty}^{\infty} \Pr[\hat{\theta} = x] \cdot \left(\Pr[Lap(\lambda) > x - 1] \cdot \Pr[Lap(\lambda) \le x - 1]\right)^{\frac{k}{2}} dx}{\int_{-\infty}^{\infty} \Pr[\hat{\theta} = x] \cdot \left(\Pr[Lap(\lambda) > x] \cdot \Pr[Lap(\lambda) \le x - 2]\right)^{\frac{k}{2}} dx}
\end{aligned}$$

Consider any $x \in (-\infty, +\infty)$. If $x > 1$, then

$$\Pr[Lap(\lambda) > x - 1] = \frac{1}{2} e^{\frac{1-x}{\lambda}} = e^{\frac{1}{\lambda}} \cdot \Pr[Lap(\lambda) > x],$$

and $\Pr[Lap(\lambda) \le x - 1] > \Pr[Lap(\lambda) \le x - 2]$. This leads to

$$\begin{aligned}
&\Pr[Lap(\lambda) > x - 1] \cdot \Pr[Lap(\lambda) \le x - 1] \\
&\ge e^{\frac{1}{\lambda}} \cdot \Pr[Lap(\lambda) > x] \cdot \Pr[Lap(\lambda) \le x - 2] \qquad (16)
\end{aligned}$$

Meanwhile, if $x \le 1$, then

$$\Pr[Lap(\lambda) \le x - 1] = \frac{1}{2} e^{\frac{x-1}{\lambda}} = e^{\frac{1}{\lambda}} \cdot \Pr[Lap(\lambda) \le x - 2],$$

and $\Pr[Lap(\lambda) > x - 1] > \Pr[Lap(\lambda) > x]$. In that case, Equation (16) still holds.

Given that Equation (16) holds for all $x \in (-\infty, \infty)$, we have

$$\begin{aligned}
&\frac{\Pr[D_1 \to E]}{\Pr[D_3 \to E]} \\
&= \frac{\int_{-\infty}^{\infty} \Pr[\hat{\theta} = x] \cdot \left(\Pr[Lap(\lambda) > x - 1] \cdot \Pr[Lap(\lambda) \le x - 1]\right)^{\frac{k}{2}} dx}{\int_{-\infty}^{\infty} \Pr[\hat{\theta} = x] \cdot \left(\Pr[Lap(\lambda) > x] \cdot \Pr[Lap(\lambda) \le x - 2]\right)^{\frac{k}{2}} dx} \\
&> \left(e^{\frac{1}{\lambda}}\right)^{\frac{k}{2}} = e^{\frac{k}{2\lambda}}.
\end{aligned}$$

Therefore, $\frac{\Pr[D_1 \to E]}{\Pr[D_3 \to E]} > e^{2\varepsilon}$ when $\lambda \le k/4\varepsilon$, which proves the lemma. □

## B. ADDITIONAL EXPERIMENTS

In this section, we evaluate the computation efficiency of PrivTree, and the impact of $\beta$ (i.e., tree fanout) on the accuracy of PrivTree. Table B shows the processing time of PrivTree on each dataset (averaged over 100 runs), with $\varepsilon$ varying from 0.05 to 1.6. The running time of PrivTree on road and msnbc are larger than that on the other datasets, since road and msnbc are larger in size than the others. In addition, the computation cost of PrivTree increases with $\varepsilon$. To understand this, recall that when PrivTree decides whether or not to split a node $v$, it first subtracts a bias term $depth(v) \cdot \delta$ from the score of $v$, and then injects noise into the biased score, after which it splits $v$ if the noisy score is larger than the threshold $\theta$. As $\delta$ is inversely proportional to $\varepsilon$ (see Corollary 1), the bias term increases when $\varepsilon$ decreases, in which case the noisy score of $v$ is less likely to be larger than $\theta$. Therefore, when $\varepsilon$ is small, PrivTree has lower probabilities to split nodes, which leads to a small running time.

Previously, in Section 6.1, we evaluate the query accuracy of PrivTree on spatial data with its fanout $\beta$ set to $2^d$, where $d$ is the dataset dimensionality. In that case, whenever PrivTree splits

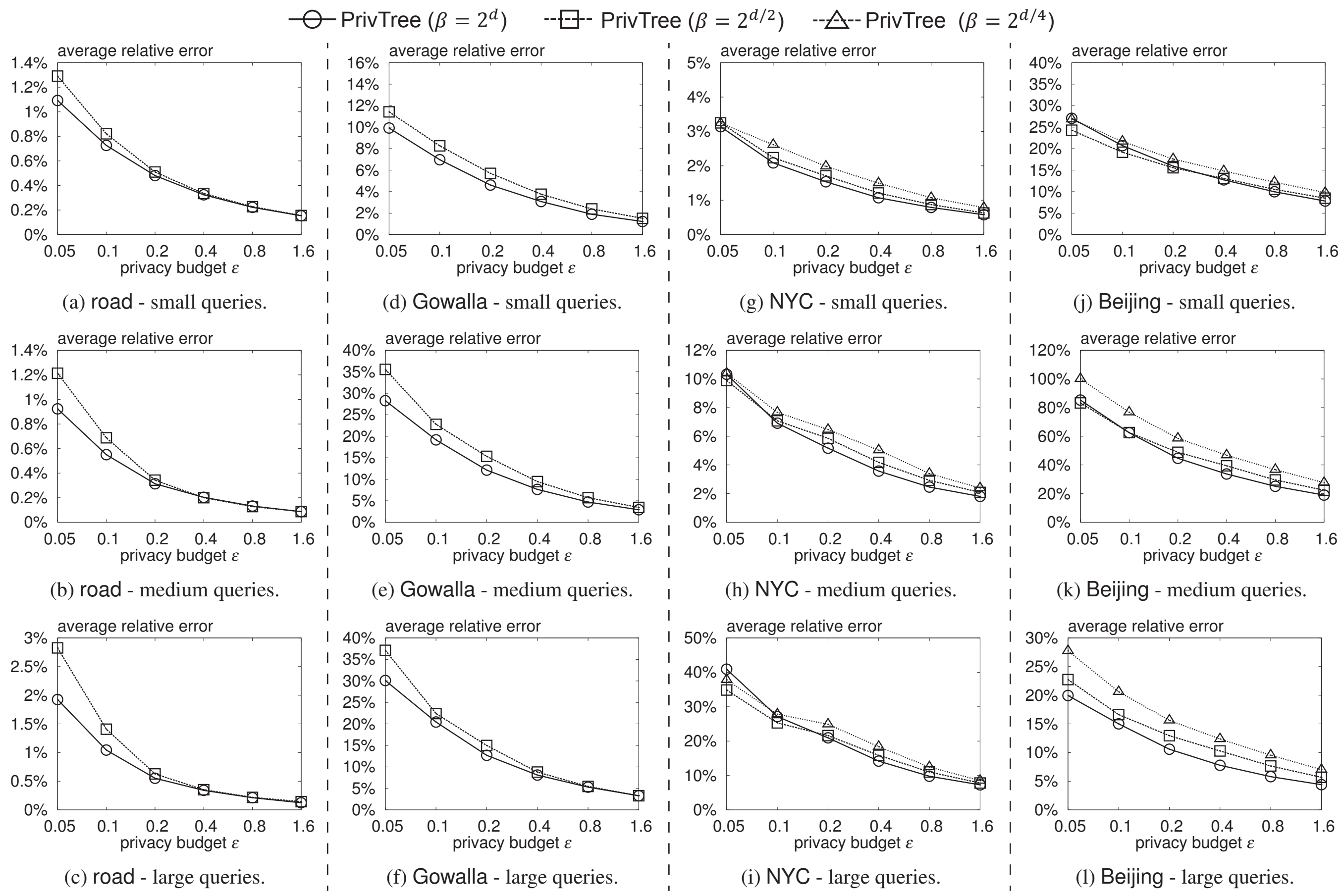


**Figure 8: Impact of fanout on PrivTree.**

**Table 4: Running time of PrivTree (seconds).**

| **Dataset** | $\varepsilon$=0.05 | $\varepsilon$=0.1 | $\varepsilon$=0.2 | $\varepsilon$=0.4 | $\varepsilon$=0.8 | $\varepsilon$=1.6 |
|---|---|---|---|---|---|---|
| road | 0.97 | 1.15 | 1.35 | 1.61 | 1.93 | 2.52 |
| Gowalla | 0.044 | 0.055 | 0.073 | 0.093 | 0.12 | 0.17 |
| NYC | 0.032 | 0.040 | 0.051 | 0.072 | 0.10 | 0.15 |
| Beijing | 0.0085 | 0.012 | 0.013 | 0.019 | 0.030 | 0.047 |
| mooc | 0.22 | 0.26 | 0.30 | 0.35 | 0.41 | 0.46 |
| msnbc | 1.73 | 2.03 | 2.21 | 2.50 | 2.72 | 3.05 |

a node $v$, the sub-domain *dom*$(v)$ of $v$ is divided into $2^d$ parts by bisecting all dimensions of *dom*$(v)$. Figure 2 illustrates the results of the same experiments when $\beta$ varies. In particular, when we set $\beta = 2^i$ with $i < d$, PrivTree would split the dimensions of each node in a round robin fashion, with $i$ dimensions being bisected each time. Observe that, in general, the query error of PrivTree slightly increases when $\beta$ decreases. This is mainly due to the bias term *depth*$(v) \cdot \delta$ that PrivTree subtracts from the score $c(v)$ of each node $v$, when it decides whether $v$ should be split. Specifically, a decreased $\beta$ increases the height of PrivTree's decomposition tree, in which case the nodes towards the leaf level of the tree would be given a larger bias term. In turn, the increased bias term renders it more difficult for PrivTree to correctly decide whether a node should be split, thus degrading the quality of PrivTree's output.

Nevertheless, on a few settings on NYC and Beijing, $\beta = 2^{d/2}$ entails smaller errors than $\beta = 2^d$. The reason is that, when $\beta$ is large, any incorrect decisions made by PrivTree in node splitting would have a more pronounced negative effect, e.g., a quadtree node with a small count would be divided into a larger number of child nodes, each of which would have an even smaller count that is likely to be overwhelmed by the noise subsequently added. The increased number of noise-dominated nodes would then lead to less accurate query answers, which explains why $\beta = 2^{d/2}$ sometimes outperforms $\beta = 2^d$. That said, the overall result in Figure 8 indicates that $\beta = 2^d$ is still a preferable choice for PrivTree.