Differentially Private Spatial Decompositions

Graham Cormode Cecilia Procopiuc Divesh Srivastava
AT&T Labs – Research
{*graham*, *magda*, *divesh*}@research.att.com

Entong Shen Ting Yu
North Carolina State University
{*eshen*, *tyu*}@ncsu.edu

Abstract— **Differential privacy has recently emerged as the de facto standard for private data release. This makes it possible to provide strong theoretical guarantees on the privacy and utility of released data. While it is well-understood how to release data based on counts and simple functions under this guarantee, it remains to provide general purpose techniques that are useful for a wider variety of queries. In this paper, we focus on spatial data, i.e., any multi-dimensional data that can be indexed by a tree structure. Directly applying existing differential privacy methods to this type of data simply generates noise.**

We propose instead the class of "private spatial decompositions": these adapt standard spatial indexing methods such as quadtrees and kd-trees to provide a private description of the data distribution. Equipping such structures with differential privacy requires several steps to ensure that they provide meaningful privacy guarantees. Various basic steps, such as choosing splitting points and describing the distribution of points within a region, must be done privately, and the guarantees of the different building blocks must be composed into an overall guarantee. Consequently, we expose the design space for private spatial decompositions, and analyze some key examples. A major contribution of our work is to provide new techniques for parameter setting and post-processing of the output to improve the accuracy of query answers. Our experimental study demonstrates that it is possible to build such decompositions efficiently, and use them to answer a variety of queries privately and with high accuracy.

I. Introduction

Releasing representative data sets which do not compromise the privacy of data subjects has occupied serious effort in the database community in recent years. The paradigm of differential privacy has recently emerged as the favored definition: it ensures that what can be learned from the released data does not substantially differ whether or not any given individual's data is included. This is intended to reassure data subjects that their participation in the process does not directly lead to information about them being revealed. The question that the data management community must now address is how to provide differential privacy guarantees, while ensuring that the result is not just private, but also useful.

Consider a dataset containing locations of individuals at a particular time, e.g., GPS locations or home addresses. Such data could be used for many applications: transportation planning, facility location, political boundary drawing etc. More generally, any data set where attributes are ordered and have moderate to high cardinality (e.g., numerical attributes such as salary) can be considered spatial data: whenever data can be indexed by a tree structure (such as a B-tree, R-tree, kd-tree etc.), it is implicitly being treated as spatial. In spatial applications, a basic primitive is to know how many individuals fall within a given region (a multi-dimensional range query). Our aim is to release information which allows such queries to be answered accurately, while giving a strong guarantee of privacy on people's locations.

Although this is a natural and important problem, there has been limited prior work that can be applied directly to this setting. The most straightforward method is to lay down a fine grid over the data, and add noise from a suitable distribution to the count of individuals within each cell [1]. For example, suppose we represent a set of 10^7 GPS locations in a grid of 10 meter × 10 meter squares over the USA territory: this yields a total of approximately 10^{11} entries, most of which are either 0 or 1. Because the purpose of the noise is to mask whether there is an individual there, the output is simply a large mass of noisy counts, with little information remaining to accurately answer queries. Any query which touches, e.g., a 1% fraction of the area includes over 10^9 noisy counts, which translates into a huge error.

In this paper, we aim to balance the requirements of practicality and utility, while achieving a desired privacy guarantee. In particular, we design a class of differentially private spatial decompositions (PSDs). These partition the space into smaller regions, and report statistics on the points within each region. Queries can then be answered by intersecting the query region with the decomposition. By performing a spatial decomposition which results in compact regions with sufficiently many points and a more uniform distribution, we expect query answers to be more accurate.

There are many existing (non-private) approaches to spatial decompositions. Some are *data-independent*, such as quadtrees which recursively divide the data space into equal quadrants. Other methods, such as the popular kd-trees, aim to better capture the data distribution and so are *data-dependent.* In this case, there are additional challenges: the description of the regions must also be differentially private. In other words, simply building a tree-structure over the exact data and populating it with noisy counts is not sufficient to meet the differential privacy definition: the choice of splits is based on the true data, and potentially reveals information.

A second issue is how to answer queries given a PSD. In the non-private case, it is fairly straightforward to take a query region and intersect it with the data structure to obtain a count of contained nodes. In the private case, it is more complicated: because we have count estimates for large regions and subregions, there are multiple ways to decompose a query region, and the answers may vary due to differing noise. To achieve optimal query accuracy, we develop two new techniques that may be of independent interest:

- First, we show that using non-uniform noise parameters can significantly improve accuracy, while maintaining the same privacy guarantee. Specifically, we propose setting the parameters in a geometric progression, increasing from root to leaves. To the best of our knowledge, this is the first result that analyzes the impact of non-uniform noise parameters.
- Second, we design a new method to compute optimal (minimum variance) answers to queries by post-processing the noisy counts. Our method works over a large class of non-uniform noise parameters. It generalizes the method of [3], which applied only to uniform noise parameters.

Putting these pieces together, we show how to build private versions of well-known data structures, such as kd-trees, R-trees, and quadtrees, in multiple dimensions. This gives a full framework for privately representing spatial data. Our experimental study ranges over this framework, and allows us to see the impact of different choices on the accuracy of query answering over a mixture of real and synthetic data. We show that our two novel techniques can reduce the absolute error of queries by up to an order of magnitude.

Outline. In Section II we describe related work, while Section III surveys concepts from differential privacy and spatial decompositions. We build our framework as follows: In Section IV, we show how to set non-uniform noise parameters in a hierarchical structure. Section V describes the post-processing method. In Section VI, we describe how to effectively find private median points under differential privacy, with further extensions in Section VII. Our experimental study in Section VIII considers different optimizations for different families of PSDs, and then compares these all together to find the best choices for working privately with spatial data. We also compare to two prior approaches [2], [4] and show that our methods outperform them.

II. Related Work

In this section, we present a brief summary of the most related topics in anonymization and privacy. For more details, there are several recent surveys and tutorials [5], [6], [7], [8].

Initial efforts to ensure privacy of released data were based on syntactic definitions such as k-anonymity [9] and ℓ-diversity [10]. Subsequent efforts tried to provide a more semantic guarantee, ensuring that no matter what knowledge or power an adversary had, their ability to draw conclusions about an individual in the data was limited. This culminated in the definition of differential privacy [1] which ensures that the probability of any property holding on the output is approximately the same, whether or not an individual is present in the source data.

Although often introduced in the context of an interaction between a data owner and a data user, differential privacy naturally adapts to a non-interactive setting where the data owner wishes to release private information derived from their data. Conceptually, this is quite straightforward: the data owner determines what collection of statistics should be released, then computes these under some mechanism which provides an overall privacy guarantee. Initial work showed how to release contingency tables (equivalent to count cubes) in this 'publishing' model [11]. Subsequent work in the database community also adopted the model of releasing tables of counts (histograms) and studied how to ensure these are accurate for different query workloads [12], [13].

There has been much work in producing mechanisms which offer differential privacy, such as the general purpose exponential mechanism [14], and the geometric mechanism [15]. Approaches such as smoothed sensitivity have attempted to adapt the amount of noise needed to the instance of the data [16]. These mechanisms have been applied to various problems, such as releasing contingency tables [11], time series [17], and recommender systems [18]. Recently, efforts have been made to improve the accuracy of (count) queries, by allowing ranges to be expressed via a smaller number of noisy wavelet or range coefficients [12], [13], and via post-processing with overlapping information [3].

However, there has been limited work that applies to spatial data. The idea of differentially private data-partitioning index structures was previously suggested in the context of private record matching [4]. The approach there is based on using an approximate mean as a surrogate for median (on numerical data) to build kd-trees. This becomes a special case in our general framework. In Section VIII-B, we show that there are much better methods to choose medians privately. We also compare to this case as part of our overall experimental study, and also for a record matching application in Section VIII-C.

The only prior work directly addressing spatial data follows the approach suggested above, and imposes a fixed resolution grid over the base data [2]. It then builds a kd-tree based on noisy counts in the grid, splitting nodes which are not considered "uniform", and then populates the final leaves with "fresh" noisy estimated counts. In our experimental study (Section VIII), we observe that this approach is inferior to other points in the framework.

III. Preliminaries

A. Differential Privacy

We now formally introduce the concepts behind differential privacy. Let $\mathcal{D}_1, \mathcal{D}_2$ be two neighboring datasets, i.e., $\mathcal{D}_1$ and $\mathcal{D}_2$ differ in only one tuple t, written as $\|\mathcal{D}_1 - \mathcal{D}_2\| = 1$. In some cases, this is taken to mean that t has different values in the two datasets; in other cases, it is interpreted as meaning that t is present in only one of the two datasets. Either version leads to privacy guarantees which differ by at most a constant factor. For concreteness, in this paper we consistently use the second definition.

Definition 1: Let $\mathcal{D}_1, \mathcal{D}_2$ be two neighboring datasets. Let $\mathcal{A}$ denote a randomized algorithm over datasets, and S be an

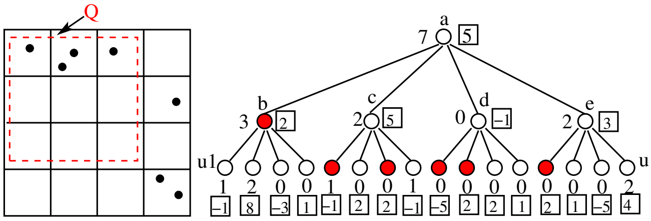


Fig. 1. Example of private quadtree: noisy counts (inside boxes) are released; actual counts, although depicted, are not released. Query Q (dotted red rectangle) could be answered by adding noisy counts of marked nodes.

arbitrary set of possible outputs of $\mathcal{A}$. Algorithm $\mathcal{A}$ is said to be ε-differentially private if

$$\Pr[\mathcal{A}(\mathcal{D}_1) \in S] \leq e^{\varepsilon} \Pr[\mathcal{A}(\mathcal{D}_2) \in S].$$

Intuitively, differential privacy guarantees that no individual tuple can significantly affect the released information: the output distribution generated by $\mathcal{A}$ is nearly the same, whether or not that tuple is present in the dataset. The most common technique for designing differentially-private algorithms was proposed in [19]. It is a noise adding mechanism, as follows.

Definition 2: **(Laplace mechanism)** Let $f(\mathcal{D})$ denote a numeric function over dataset $\mathcal{D}$. An ε-differentially private mechanism for releasing f is to publish $\mathcal{L}(\mathcal{D}) = f(\mathcal{D}) + X$, where X is a random variable drawn from the Laplace distribution $\text{Lap}(\sigma(f)/\varepsilon)$.

The value $\sigma(f)$, called the *sensitivity of* f, is the maximum change in f when any single tuple of $\mathcal{D}$ changes. Formally:

$$\sigma(f) = \max_{\mathcal{D}_1,\mathcal{D}_2:\|\mathcal{D}_1-\mathcal{D}_2\|=1} |f(\mathcal{D}_1) - f(\mathcal{D}_2)|.$$

For example, if $f = \text{count}$, then $\sigma(f) = 1$: for any two neighboring datasets $\mathcal{D}_1$ and $\mathcal{D}_2$, the cardinalities of $\mathcal{D}_1$ and $\mathcal{D}_2$ differ by 1.

B. Spatial Decompositions

Spatial indexes are routinely used in database management, with a long history stretching over decades [20], [21]. Their main purpose is to allow queries to determine quickly whether or not a particular point is present in the data set. Clearly, we cannot extend this functionality to a privacy-preserving setting. However, spatial indexes also allow efficient computation for a rich set of aggregate queries, particularly *range queries*. We focus on these queries when studying the utility of PSDs.

Formally, a *spatial decomposition* is a hierarchical (tree) decomposition of a geometric space into smaller areas, with data points partitioned among the leaves. Indexes are usually computed down to a level where the leaves either contain a small number of points, or have a small enough area, or a combination of the two. Unless otherwise specified, we assume that the tree is complete, i.e., all leaf-to-root paths have the same length, and all internal nodes have the same fanout.

The following classification of spatial decompositions is widely used. We will describe our private spatial decompositions following the same taxonomy.

Data-independent decompositions. The best known example in this category is the *quadtree* in two dimensions, and its generalizations to higher dimensions (octree, etc.) Their defining feature is that their structure is pre-defined, and depends only on the domain of the attributes. E.g., in the quadtree, nodes are recursively divided into four equal regions via horizontal and vertical lines through the midpoint of each range [21].

Data-dependent decompositions. Here, the hierarchical partition of nodes is based on the input. Examples include:
– *Kd-trees:* Nodes in a kd-tree are recursively split via lines passing through the median data value along some coordinate axis. The splitting axis changes at each level, in a cyclic order.
– *Hilbert R-trees:* The R-tree is a spatial decomposition formed by nested (hyper)rectangles which may overlap. A Hilbert R-tree is computed as follows: Map all points to a Hilbert space-filling curve of sufficiently large order (see [22] for details). Build a binary tree on the Hilbert values, then map it back into the original space. The bounding boxes of data points corresponding to nodes in the tree form the Hilbert R-tree.

Hybrid trees. Although less studied in the classical setting, a hybrid approach can be particularly useful in a privacy-aware model. In particular, we consider using data-dependent splits for the first ℓ levels of the tree (for some ℓ decided in advance), then switch to data-independent splitting (e.g., quadtree) for the remainder. We observe in Section VIII-B that queries over private hybrid trees are generally more accurate than with many other indexes.

C. Building Private Spatial Decompositions

We are now ready to describe how to build PSDs. The simplest PSD is formed by instantiating a data-independent tree such as a full quadtree, but computing counts for each node via the Laplace mechanism. See Figure 1 for an example.

Recall that the concept of sensitivity (Definition 2) captures the maximum change in the output caused by the presence or absence of a single tuple. For *data-independent* trees, releasing the structure of the index (i.e., the node rectangles) does not endanger the privacy of any individual. The only data-dependent features are the node counts: each node stores the number of data points that lie in the spatial cell associated with it. Note that adding or deleting a single tuple changes the counts of all the nodes on the path from the root to the leaf containing that tuple. Therefore, to obtain a tree that satisfies ε-differential privacy as a whole, we need to combine the privacy guarantees of individual node counts. We use the following well-known composition lemma:

Lemma 1: [18] Let $\mathcal{A}_1, \ldots, \mathcal{A}_t$ be t algorithms such that $\mathcal{A}_i$ satisfies ε_i-differential privacy, $1 \leq i \leq t$. Then their sequential composition $\langle \mathcal{A}_1, \ldots, \mathcal{A}_t \rangle$ satisfies ε-differential privacy, for $\varepsilon = \sum_{i=1}^{t} \varepsilon_i$.

In any partition tree, we only need to consider sequential compositions of private counts along a path: if nodes u and v are not on the same root-to-leaf path, their node counts are independent of each other, so knowing the output of $\mathcal{A}_u$ does not affect the privacy guarantee of $\mathcal{A}_v$.

This outline provides a simple PSD. However, we have developed several optimization techniques which can significantly improve the utility of PSDs, which is the technical focus of this paper. Section IV shows how to choose the values ε_i, while Section V shows how to process the noisy counts to maximize query accuracy. These results apply to all spatial decompositions. For data-dependent and hybrid trees, to guarantee differential privacy, the structure of the tree itself must also be perturbed by noise. We study this issue in Section VI for kd-trees. We also consider building private Hilbert R-trees, which are binary trees (i.e., one-dimensional kd-trees) in the Hilbert space. When mapping back to the original space, we use the bounding boxes of Hilbert values corresponding to tree nodes to define the node rectangles (thus preserving privacy). For brevity, we do not explicitly discuss Hilbert R-trees further, although we include them in our experimental evaluation in Section VIII.

IV. Allocating Noise Parameters

In this section, we focus on how to choose noise parameters ε_i such that the composition rule is satisfied for all tree paths (see Section III-C). Let h denote the height of the tree; leaves have level 0 and the root has level h. We assume that all nodes at level i have the same Laplace parameter ε_i (other choices are discussed at the end of this section). Hence, given a total privacy "budget" of ε, we need to specify ε_i, $0 \le i \le h$, such that $\sum_{i=0}^{h} \varepsilon_i = \varepsilon$. We refer to a choice of ε_i's as a *budget strategy*. The goal is to minimize the resulting query errors.

Error measure. For any query Q, let $\tilde{Q}$ denote the answer to Q computed over the private tree. Then $\tilde{Q}$ is a random variable which is an unbiased estimator of the true answer (since noise has mean 0). Its variance $\mathrm{Var}(\tilde{Q})$ is a strong indicator of query accuracy. As in prior work [13], [12], we define the error measure to be $\mathrm{Err}(Q) = \mathrm{Var}(\tilde{Q})$. The error of a query workload $Q_1, \dots, Q_s$ is $\sum_{i=1}^{s} \mathrm{Err}(Q_i)/s$.

A. Query Processing

Unlike a tree over the original (unperturbed) counts, a PSD may return many different results to a query Q.

Query processing example. Figure 1 shows a possible processing for query Q, which sums the noisy counts in nodes b, $u5$, $u7$, $u9$, $u10$ and $u13$. The answer is 2. However, if we replace b's count by the sum of its children's counts, and the sum of $u5$ and $u7$ by the difference between c's count and the sum of $u6$, $u8$, the answer becomes 8. This is because the noise is independent, and there are multiple ways of representing Q as a union or difference of node rectangles. Adding/subtracting corresponding noisy counts yields different results. ■

Consequently, to analyze $\mathrm{Err}(Q)$, we must first describe a standard method for computing $\tilde{Q}$. Let Y be the set of noisy counts, and let U be the set of nodes used to answer Q. Then $\mathrm{Err}(Q) = \sum_{u \in U} \mathrm{Var}(Y_u)$, i.e., the total variance is the sum of the node variances. Thus the error grows, to a first approximation, with the number of noisy counts included. We adopt the canonical range query processing method [21], which minimizes the number of added counts.

The method is as follows: Starting from the root, visit all nodes u whose corresponding rectangle intersects Q. If u is fully contained in Q, add the noisy count Y_u to the answer $\tilde{Q}$; otherwise, recurse on the children of u, until the leaves are reached. If a leaf a intersects Q but is not contained in Q, use a *uniformity assumption* to estimate what fraction of Y_a should be added to $\tilde{Q}$.

Let $n(Q)$ be the number of nodes that contribute their counts to $\tilde{Q}$. For each $0 \le i \le h$, let n_i be the number of nodes at level i that are maximally contained in Q (i.e. contributed their counts to $\tilde{Q}$ according to the method above), so $n(Q) = \sum_{i=0}^{h} n_i$. The following result bounds each n_i and will guide us in choosing noise parameters ε_i.

Lemma 2: Let Q be an arbitrary range query, and $\mathcal{T}$ be a spatial decomposition of height h in two dimensions. Then

(i) If $\mathcal{T}$ is a quadtree, $n_i \le 8 \cdot 2^{h-i}$ and

$$n(Q) \le 8(2^{h+1} - 1) = O(4^{h/2}).$$

(ii) If $\mathcal{T}$ is a kd-tree, $n_i \le 8 \cdot 2^{\lfloor (h-i+1)/2 \rfloor}$ and

$$n(Q) \le 8(2^{\lfloor (h+1)/2 \rfloor + 1} - 1) = O(2^{h/2}).$$

Proof: Let f denote the fanout of $\mathcal{T}$: $f = 4$ for the quadtree and $f = 2$ for the kd-tree. Consider the left vertical extent of Q. If $\mathcal{T}$ is a quadtree, then for each rectangle at level i intersected by this vertical line, there are 2 rectangles at level $i-1$ intersected by it. If $\mathcal{T}$ is a kd-tree, then for each rectangle that the vertical extent intersects at level i (assume wlog that level i corresponds to a vertical split), it intersects 1 at level $i-1$ and 2 at level $i-2$. In either case, the number of rectangles intersected by the left extent of Q grows by at most a factor f every two levels.

We repeat this argument for the other three extents of Q. Additionally, the number of nodes that are maximally contained in Q can be bounded by the number of nodes that are intersected by Q, and the recurrence for n_i follows. ■

We remark that a similar result extends to d dimensional decompositions, where the behavior is $n(Q) = O(f^{h(1-1/d)})$ for a tree of height h and fanout f.

B. Budget Strategies

The variance of the Laplace mechanism with parameter ε_i is $\mathrm{Var}(\mathrm{Lap}(\varepsilon_i)) = 2/\varepsilon_i^2$. Since the noise is independently generated in each node, we deduce that

$$\mathrm{Err}(Q) = \textstyle\sum_{i=0}^{h} 2n_i/\varepsilon_i^2. \tag{1}$$

For simplicity, we analyze strategies only for quadtrees; hence, $n_i \le 8 \cdot 2^{h-i}$ by Lemma 2(i). The analysis for kd-trees is similar.

Uniform Budget: A natural strategy is to set $\varepsilon_i = \varepsilon/(h+1)$. Prior work that finds counts in trees (e.g. [3]) has used this model. Then $\mathrm{Err}(Q) = \frac{2(h+1)^2}{\varepsilon^2} \sum_{i=0}^{h} n_i \le \frac{16}{\varepsilon^2}(h+1)^2(2^{h+1}-1)$.

Geometric Budget: We can significantly improve the query accuracy by considering a non-uniform budgeting strategy.

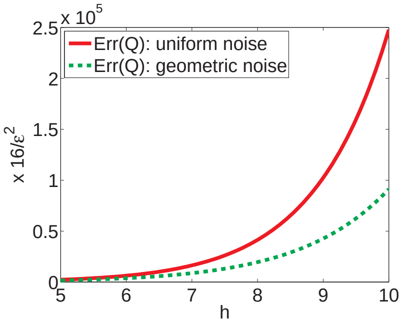


Fig. 2. Worst case behavior of Err(Q) for uniform and geometric noise.

Substituting the bound for n_i into (1) we obtain the following optimization problem to minimize the upper bound.

$$\begin{array}{ll} \text{Minimize} & \sum_{i=0}^{h} 2^{h-i}/\varepsilon_i^2 \\ \text{Subject to:} & \sum_{i=0}^{h} \varepsilon_i = \varepsilon. \end{array}$$

Lemma 3: An upper bound for Err(Q) is

$$\frac{16(2^{(h+1)/3}-1)^3}{\varepsilon^2(\sqrt[3]{2}-1)^3} \leq \frac{2^{h+7}}{\varepsilon^2}$$

which is attained for $\varepsilon_i = 2^{(h-i)/3}\varepsilon\frac{\sqrt[3]{2}-1}{2^{(h+1)/3}-1}$.

Proof: By the Cauchy-Schwarz inequality, we have

$$(\sum_{i=0}^{h}\varepsilon_i)(\sum_{i=0}^{h}2^{h-i}/\varepsilon_i^2) \geq (\sum_{i=0}^{h}\sqrt{\varepsilon_i 2^{h-i}/\varepsilon_i^2})^2,$$

with equality attained if and only if $\varepsilon_i = C2^{h-i}/\varepsilon_i^2, \forall i$, where C is a constant. Hence, $\varepsilon_i = \sqrt[3]{C}2^{(h-i)/3}$. Plugging this into $\sum_{i=0}^{h}\varepsilon_i = \varepsilon$, we obtain $\sqrt[3]{C} = \frac{\varepsilon(\sqrt[3]{2}-1)}{2^{(h+1)/3}-1}$. Then Err($Q$) is at most

$$16\sum_{i=0}^{h}2^{h-i}/(C^{2/3}2^{2(h-i)/3}) = 16C^{-2/3}\sum_{i=0}^{h}2^{(h-i)/3} = 16\varepsilon/C,$$

and the result follows. ■

Lemma 3 shows that query accuracy is improved with a geometric budgeting scheme: starting from the root, the budget increases geometrically (by a factor of $2^{1/3}$), so the leaf counts are reported with highest accuracy.

Studying the bound in the lemma, it seems that we should reduce h to reduce the noise. But this variance only bounds the error from noisy counts. We also have to account for the error due to queries which partly intersect some leaves, i.e., errors arising from the uniformity assumption. In the worst case, this error is proportional to the number of points in each leaf intersected by the query. We intersect $n_h \approx 2^h$ leaves. On the average for an input of n points, we could have $O(n/4^h)$ points per leaf (assuming balanced leaf counts). Hence, the uniformity assumption error behaves as $O(2^h n/4^h) = O(n/2^h)$. Then the overall error grows as $O(n/2^h + 2^{h/3})$, suggesting we benefit more overall as h increases.

Comparing strategies. Let Q be a query that includes the maximum number of counts at each level, i.e., $n_i = 8 \cdot 2^{h-i}$.[1] Figure 2 shows that worst case error for a uniform budget, $\text{Err}_{\text{unif}}(h) = (h+1)^2(2^{h+1}-1)$ grows much faster than the geometric budget error, $\text{Err}_{\text{geom}}(h) = \frac{(2^{(h+1)/3}-1)^3}{(\sqrt[3]{2}-1)^3}$. This comparison assumes that Q touches a maximal number of nodes at each level. While in practice queries may touch fewer nodes, our experiments show that uniform noise is still significantly less accurate for a large number of different queries.

Other budget strategies. There is a large class of strategies for dividing ε along a path. For example, we could build a quadtree down to depth h and set $\varepsilon_h = \varepsilon$, i.e., allocate the entire budget to leaves (this approach is used in [4]). In this case, queries are computed over the grid defined by the leaf regions and the hierarchical structure of the tree is irrelevant. Or we could conserve the budget by setting $\varepsilon_i = 0$ for some levels i, and releasing no counts for those levels. Queries then use counts from descendant nodes instead. Conceptually, this is equivalent to increasing the fanout of nodes in the tree.

Finally, we note that we do not have to use the same ε_i for all nodes on level i. For a generic query workload, this approach makes sense, as each node on level i is equally likely to be touched by the workload. However, if the workload is known a priori, one should analyze it to determine how frequently each node in the tree contributes to the answers. Then ε_u could be larger for the nodes u that contribute more frequently, subject to the sum along each path remaining ε.

V. Optimizing Query Accuracy

In the previous section we discussed how using geometric noise in a hierarchical decomposition can significantly improve the accuracy of queries. We now show that it is possible to further improve query accuracy via a post-processing of the noisy counts. The goal is to compute a new set of counts for which query errors are minimized. Note that this does not affect the privacy guarantee, as our method takes as input the output of the differentially private mechanism.

Post-processing example. Consider a simple tree with one root a and four children, b, c, d and e. Let Y_v denote the noisy count of node $v \in \{a,b,c,d,e\}$, and assume first that we used uniform noise of $\varepsilon/2$ on each count. There are two natural estimates of the true count of the root: Y_a itself, and the sum of counts of leaves, $Y_b + Y_c + Y_d + Y_e$. A first attempt is to estimate $\beta_a = Y_a/2 + (Y_b + Y_c + Y_d + Y_e)/2$, i.e., the average of the two estimates. In this case, $\text{Var}(\beta_a) = \text{Var}(Y_a)/4 + 4\,\text{Var}(Y_b)/4 = (5/4)\,\text{Var}(Y_a)$, *worse* than directly using Y_a. But it turns out we can do better: setting $\beta_a = 4Y_a/5 + (Y_b + Y_c + Y_d + Y_e)/5$ yields $\text{Var}(\beta_a) = (4/5)\,\text{Var}(Y_a)$.

For any non-uniform budgeting scheme, if the budget for a is ε_1 and the budget of its children is ε_0, then $\beta_a = \frac{4\varepsilon_1^2}{4\varepsilon_1^2+\varepsilon_0^2}Y_a + \frac{\varepsilon_0^2}{4\varepsilon_1^2+\varepsilon_0^2}(Y_b + Y_c + Y_d + Y_e)$ improves accuracy: one can derive $\text{Var}(\beta_a) = \frac{8}{4\varepsilon_1^2+\varepsilon_0^2} < \frac{2}{\varepsilon_1^2} = \text{Var}(Y_a)$. ■

As this example illustrates, there are many possible ways of obtaining new counts as linear combinations of the published noisy counts. The choices increase exponentially with the tree

[1] More rigorously, $n_i = \min\{8 \cdot 2^{h-i}, 4^{h-i}\}$, but this does not alter the analysis significantly.

size, as we can combine counts of ancestors, descendants, siblings etc. in various ways. Luckily, we do not need to explore them in order to compute the best solution. From linear statistical inference, we know that the best set of new counts is obtained via the ordinary least-squares estimate (OLS) [23]. In general, computing the OLS for n unknowns requires solving a linear system with $n \times n$ matrices. Rather than explicitly inverting such (large) matrices, we present a linear time algorithm that achieves the same result, by taking advantage of the inherent symmetries of the matrices defined for the tree structure. Prior work considered the simpler case of uniform noise parameters [3]. We give a solution to the more general case when all nodes at level i have the same Laplace parameter ε_i (this encompasses both uniform and geometric budgeting). The generalization requires significant technical effort to prove, but yields a simple algorithm.

Notation: As before, let f denote the fanout of the spatial index and let h denote its height. We use $h(v)$ to denote the height of node v: $h(v) = 0$ if v is a leaf, and $h(root) = h$. We assume that the tree is complete, i.e., all paths have length h and all internal nodes have fanout f. Let $u \prec v$ denote that u is a leaf in the subtree of v. Let $\mathrm{anc}(u)$ be the set of all ancestors of u, *including node u.* We use $\mathrm{par}(u)$ and $\mathrm{child}(u)$ to denote the parent, resp. the set of children, of node u.

For linear inference over a tree structure, the definition of an OLS is as follows.

Definition 3: Let Y denote the vector of original noisy counts, i.e., Y_v is the noisy count of node v. Let ε_v denote the noise parameter for node v. We denote by β the vector of counts after post-processing. Then β is the *ordinary least squares estimator (OLS)* if it is *consistent*, i.e., $\beta_v = \sum_{u\in\mathrm{child}(v)} \beta_u$ for all nodes v, and it minimizes $\sum_v \varepsilon_v^2 (Y_v - \beta_v)^2$.

The OLS β has two powerful properties: It is *unbiased* for any query Q (recall that Y is also unbiased, since noise has mean 0). Most importantly, among all unbiased linear estimators derived from Y, it achieves *minimum error for all range queries*. In particular, it achieves smaller error than Y.

The computation of β is based on the following result, proven in the Appendix.

Lemma 4: For any node v in the spatial index, the following recurrence holds (with Y and β as in Definition 3):

$$\left(\sum_{j=0}^{h(v)} f^j \varepsilon_j^2\right)\beta_v + f^{h(v)} \sum_{w\in\mathrm{anc}(v)\setminus\{v\}} \beta_w \varepsilon_{h(w)}^2 = \sum_{u\prec v}\sum_{w\in\mathrm{anc}(u)} \varepsilon_{h(w)}^2 Y_w \qquad (2)$$

Lemma 4 provides an efficient way to compute β. First, precompute the following array E of $h+1$ entries: $E_l = \sum_{j=0}^{l} f^j \varepsilon_j^2$. Since $E_l = E_{l-1} + f^l \varepsilon_l^2$, this takes time $O(h)$. For any node v, define $Z_v = \sum_{u\prec v}\sum_{w\in\mathrm{anc}(u)} \varepsilon_{h(w)}^2 Y_w$. We compute Z_v for all nodes v in two linear traversals of the tree, as follows:

Phase I: Top-down traversal

We compute Z_v for all leaves v. Note that in that case, $Z_v = \sum_{w\in\mathrm{anc}(v)} \varepsilon_{h(w)}^2 Y_w$. Let $\alpha_{root} = \varepsilon_h^2 Y_{root}$. In a top-down traversal of the tree, compute for each node u: $\alpha_u = \alpha_{\mathrm{par}(u)} + \varepsilon_{h(u)}^2 Y_u$. When we reach a leaf v, we set $Z_v = \alpha_v$.

Phase II: Bottom-up traversal

We compute Z_v for all internal nodes v as $Z_v = \sum_{u\in\mathrm{child}(v)} Z_u$; this requires a single bottom-up traversal.

Phase III: Top-down traversal

We now compute β_v for all nodes v. While doing so, we also compute an auxiliary value F_v, defined as $F_v = \sum_{w\in\mathrm{anc}(v)\setminus\{v\}} \beta_w \varepsilon_{h(w)}^2$. Note that Equation (2) for $v = root$ is:

$$\left(\sum_{j=0}^{h} f^j \varepsilon_j^2\right)\beta_{root} (= E_h \beta_{root}) = Z_{root}$$

So we compute $\beta_{root} = Z_{root}/E_h$. In addition, let $F_{root} = 0$. For any node $v \neq root$, assume we have already computed β_w and F_w for all $w \in \mathrm{anc}(v) \setminus \{v\}$. Then we compute $F_v = F_{\mathrm{par}(v)} + \beta_{\mathrm{par}(v)} \varepsilon_{h(v)+1}^2$. From Equation (2), we find

$$\beta_v = \frac{Z_v - f^{h(v)} \sum_{w\in\mathrm{anc}(v)\setminus\{v\}} \beta_w \varepsilon_{h(w)}^2}{E_{h(v)}} = \frac{Z_v - f^{h(v)} F_v}{E_{h(v)}}.$$

From Lemma 4 and the above description we conclude:

Theorem 5: The algorithm consisting of Phases (I)–(III) described above computes the OLS estimator in time linear in the size of the tree.

In Section VIII-B we conduct an experimental evaluation which supports our theoretical result, and shows significant improvement in query accuracy using the OLS.

VI. Data-Dependent and Hybrid Trees

As noted in Section III-B, publishing data-dependent or hybrid trees with privacy guarantees must overcome an additional challenge: the tree structure itself could reveal private information. In most cases, the current node is split via a line through the current median value along some axis. For privacy reasons, we cannot reveal the exact median. In Section VI-A we discuss methods for computing a private median for a set of points.

Extending the computation of private medians to a hierarchical structure requires some subtlety. Note that, unlike private counts, it is no longer the case that only root-to-leaf compositions matter. If a tuple is deleted from the dataset, it affects the (true) medians not only on its respective path, but also for nodes in different parts of the tree. To show the privacy of our approach, we appeal to the fact that the composition of differentially private outputs is well-understood in an interactive model, where a user asks a series of queries. We can imagine the following interaction: At the root level, our private algorithm $\mathcal{A}$ outputs a noisy median value m_1. A user subsequently asks for the median of the points lying on one side of m_1, and for the median of the points lying on the other side of m_1 (hence, m_1 becomes part of the user query). Algorithm $\mathcal{A}$ returns two noisy values, m_2 and m_3, each computed with respect to the user-specified subset. Hence, the computation can continue recursively, and it is again sufficient to consider sequential compositions only along root-to-leaf paths. Now observe that it is straightforward for the data owner to play both roles, and output the final result (without interaction) as the PSD.

A. Private Medians

We now outline several methods for computing a private median for a given set of points. These approaches are either implicitly or explicitly developed in prior work, but we are not aware of any previous comparison. Here, we bring together the different approaches, and give some new analysis on their behavior. We compare them empirically in Section VIII.

Let $C = \{x_1, \ldots, x_n\}$ be a (multi)set of n values in non-decreasing order in some domain range $[lo, hi]$ of size $hi - lo = M$, and let x_m be its median value. We wish to compute a private median for C. We could apply the Laplace mechanism (Definition 2), i.e., return $\mathcal{L}(C) = x_m + X$, where X is Laplace noise. However, the sensitivity of the median is of the same order of magnitude as the range M (see [16], [4]), and the noise value X usually dwarfs x_m. A consequence of this is that the noisy median is frequently outside the range $[lo, hi]$. When used in a kd-tree construction, such a noisy median does not help to divide the data. Instead, we study the following four methods.

Smooth Sensitivity [16] tailors the noise to be more specific to the set C. However, smooth sensitivity has slightly weaker privacy guarantees than the Laplace mechanism: it only satisfies so-called (ε, δ)-differential privacy; see [16] for details.

Definition 4: (from Claim 3.3 of [16]) Let $0 < \varepsilon, \delta < 1$, and let $\xi = \frac{\varepsilon}{4(1+\ln(2/\delta))}$. The *smooth sensitivity* of the median is defined as

$$\sigma_s(\text{median}) = \max_{0 \le k \le n} (e^{-k\xi} \max_{0 \le t \le k+1} (x_{m+t} - x_{m+t-k-1})).$$

(where we define $x_i := lo$ if $i < 0$ and $x_i := hi$ if $i > n$).

The smooth sensitivity mechanism for median is defined as $\text{SS}(C) = x_m + \frac{2\sigma_s}{\varepsilon} \cdot X$ where X is a random variable drawn from the Laplace distribution with parameter 1 and $\sigma_s = \sigma_s(\text{median})$.

Exponential Mechanism is a general method proposed in [14] as an alternative to the Laplace mechanism: instead of adding random noise to the true value, the output value is drawn from a probability distribution over all possible outputs, so that the differential privacy condition is satisfied. Applied to median, we obtain the following algorithm: [2]

Definition 5: For any $x \in [lo, hi]$, let $\text{rank}(x)$ denote the rank of x in C. The exponential mechanism EM returns x with $\Pr[\text{EM}(C) = x] \propto e^{-\frac{\varepsilon}{2}|\text{rank}(x) - \text{rank}(x_m)|}$.

Since all values x between two consecutive values in C have the same rank, they are equally likely to be chosen. Thus, EM can be implemented efficiently by observing that it chooses an output from the interval $I_k = [x_k, x_{k+1})$ with probability proportional to $|I_k| e^{-\frac{\varepsilon}{2}|k-m|}$. Conditional on I_k being chosen in the first step, algorithm EM then returns a uniform random value in I_k.

Cell-based Method is a heuristic proposed in [2]. It imposes a fixed resolution grid over C then computes the median based on the noisy counts in the grid cells. When applied to a hierarchical decomposition, a fixed grid is computed over the entire data, then medians are computed from the subset of grid cells in each node. Cell counts have sensitivity 1. The accuracy of this method depends on the coarseness of the grid relative to the data distribution.

Noisy Mean is a heuristic from [4], which replaces median with mean. A private mean can be computed privately by computing a noisy sum (with sensitivity M) and a noisy count (with sensitivity 1), and outputting their ratio. If the count is reasonably large, this is a fair approximation to the mean, though there is no guarantee that this is close to the median.

Provided the data is not too skewed, smooth sensitivity and exponential mechanism have constant probability of choosing a good split; i.e., the noisy median has a constant fraction of the data points on each side. We say that the data is not too skewed if it obeys the "80/20 rule": the central 80% portion of the data covers at least 20% of the entire data range M (hence, it's not too concentrated). Formally, $(x_{4n/5} - x_{n/5}) \ge M/5$. For smooth sensitivity, we also require n to be large enough that ξn is at least a small constant (where ξ is as in Definition 4). The proof of the following lemma is provided in the appendix.

Lemma 6: Let C be such that $x_{4n/5} - x_{n/5} \ge M/5$.
(i) $\Pr[\text{SS}(C) \in [x_{n/5}, x_{4n/5}] | n\xi \ge 4.03] > 0.5(1 - e^{-\varepsilon/4})$;
(ii) $\Pr[\text{EM}(C) \in [x_{n/5}, x_{4n/5}]] \ge \frac{1}{6}$.

B. Balancing Privacy Budgets

As discussed before, the privacy guarantee of a data-dependent tree is obtained by composing the individual privacy guarantees for medians and counts along each root-to-leaf path, as in Lemma 1.

Let h be the height of the tree and let $\mathcal{A}_i^m$, $1 \le i \le h$, be the noisy median algorithms corresponding to the internal nodes on a path, such that $\mathcal{A}_i^m$ is ε_i^m-differentially private. Let $\mathcal{A}_i^c$, $0 \le i \le h$, be the noisy count algorithms corresponding to the same nodes, plus the leaf, such that $\mathcal{A}_i^c$ is ε_i^c-differentially private. Then the PSD released is ε-differentially private, where

$$\varepsilon = \sum_{i=1}^{h} \varepsilon_i^m + \sum_{i=0}^{h} \varepsilon_i^c$$

(assuming that sums are equal along each path. Else, ε is the maximum sum along any path.)

Median vs. count noise. In our experiments, we consider separate allocations of the ε budget for the median computation, and for the count computation. An important consideration is that larger values of ε_i yield smaller noise (in a probabilistic sense). However, the noise magnitude has different consequences on the overall accuracy of the tree, for median vs. count. Large count noise gives more uncertainty on the count of each region in the tree which Q touches. By contrast, large median noise may result in a skewed split in some internal node u. Hence, the children of u are unbalanced in terms of number of points in their leaves, as well as the size of regions they represent. However, queries that touch the descendants of u may still perform well, provided the respective noisy counts are accurate enough. Still, we cannot neglect median finding

[2] This computation is implicit in McSherry's PINQ system [24].

entirely. If the median does not divide the current point set so that there are at least a constant fraction of points on each side, then we are essentially wasting a level of the tree. This becomes more of a challenge deeper in the tree, as the point set sizes shrink.

Let $\varepsilon_{\text{median}} = \sum_{i=1}^{h} \varepsilon_i^m$ and $\varepsilon_{\text{count}} = \sum_{i=0}^{h} \varepsilon_i^c$. We study different strategies for choosing these values such that $\varepsilon_{\text{median}} + \varepsilon_{\text{count}} = \varepsilon$, by analyzing their impact on various query loads.

Budgeting median noise. Once $\varepsilon_{\text{median}}$ is fixed, we must distribute it among internal nodes. A simple strategy is uniform budgeting, i.e., $\varepsilon_i^m = \varepsilon_{\text{median}}/h$. The hybrid tree approach, which switches to quadtree splitting after ℓ levels implies setting $\varepsilon_i^m = \varepsilon_{\text{median}}/\ell$ for $h-\ell < i \le h$ and $\varepsilon_i^m = 0$ for $0 \le i \le h-\ell$.

Flattening the kd-tree. From Lemma 2 and the computation of query errors from Section IV, it seems that a (2D) kd-tree built over the same number of leaves as a quadtree has significantly worse accuracy. The reason is that the kd-tree has twice the height of the quadtree, so the noise budget is divided into twice as many pieces. To better compare quadtrees and kd-trees, we can ensure that both have the same fanout of 4. We propose "flattening" the kd-tree: connect the root to its four grandchildren, and recurse down the tree, skipping every other level. (This is akin to setting $\varepsilon_i = 0$ for every other level). The bound on $n(Q)$ over a flattened kd-tree is now the same as for a quadtree of the same height, following from Lemma 2. From now on, we assume all trees have fanout $f = 4$.

VII. Further Enhancements

In this section we discuss how to apply two popular strategies—sampling and pruning—in the context of differential privacy. Sampling can be used to significantly improve the running time for computing data-dependent trees. Pruning can improve the query accuracy for both data-independent and data-dependent trees.

Sampling to reduce computation time. When data is large, computing differentially private functions can be very time consuming, even when the functions take near-linear time. A natural technique is to compute the function on only a small sample of the full data to speed up the computation. Intuitively, sampling is very compatible with differential privacy: the influence of any individual is reduced, since they may not be included in the sample. The next result extends Kasiviswanathan *et al.* [25][3]:

Theorem 7: Given an algorithm $\mathcal{A}$ which provides ε-differential privacy, and $0 < p < 1$, including each element of the input into a sample S with probability p and outputting $\mathcal{A}(S)$ is $2pe^{\varepsilon}$-differentially private.

Treating $2e^{\varepsilon}$ as a constant (it is between 2 and 5.5 for $0 < \varepsilon < 1$), it is sufficient to sample at a rate of $\approx \varepsilon'/10$ to achieve ε'-differential privacy. For large enough data, sampling at a rate of, say, 1% and applying Laplace noise with parameter 0.9 achieves 0.1-differential privacy—but it processes a data set two orders of magnitude smaller. We use this result to compute noisy medians for data-dependent and hybrid trees, via the SS and EM methods from Section VI-A. We conclude that sampling makes both methods an order of magnitude faster. It only marginally deteriorates the accuracy of the exponential mechanism, and it improves the accuracy of the smooth sensitivity-based noise. See Section VIII-B.

Sampling is not useful for noisy counts: computing a sample (independently per node) and computing the exact count both require linear time. However, sampling introduces more inaccuracy in the noisy count. For the same reason, the cell-based method for noisy medians (Section VI-A) is implemented without sampling.

Pruning. Due to the uneven nature of the splitting process, it is possible that both data dependent and data independent trees contain some nodes with few or no points. Dividing such sets can be counter-productive: the noise added to the counts of their descendants may add up for queries which intersect their region. Instead, it is preferable to cut off the tree at this point. This requires some care: the choice to stop cannot be based on the "true" count of the points, as this does not meet the differential privacy requirement. Instead, we can choose to stop if the noisy count estimated for a node in the tree is less than threshold m. In our setting, we choose to apply this pruning after the postprocessing procedure of Section V, which operates on a complete tree. We observed that this improves empirically over retaining all nodes.

VIII. Experimental Study

We begin by studying design choices for each class of methods: the choice of medians for data-dependent PSDs, different strategies for budgeting and post-processing for data-independent PSDs. We then study the effectiveness of query answering for the different approaches. Finally, we apply our findings to a particular application [4].

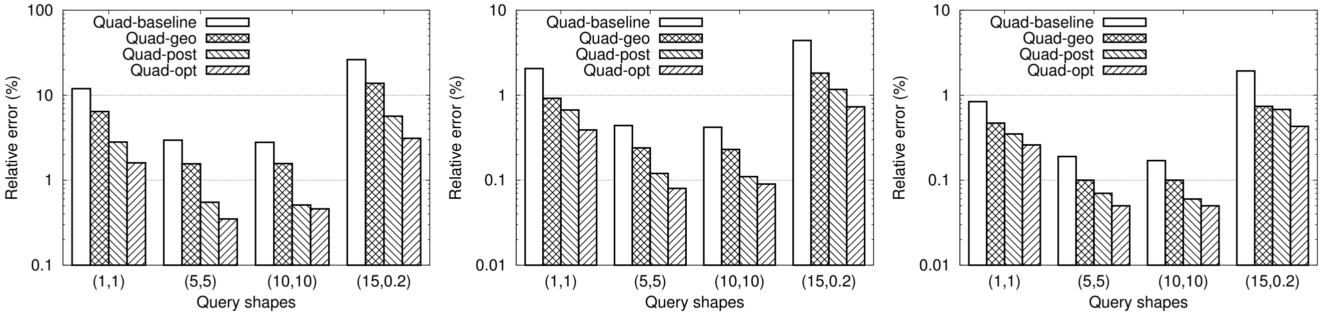
A. Experimental Environment

To evaluate the accuracy of query answering of various PSDs, we experimented on a mixture of real and synthetic data. Real data was drawn from the 2006 TIGER/Line dataset[4]. Here, we present results using GPS coordinates of road intersections in the states of Washington and New Mexico. This data represents a rather skewed distribution corresponding roughly to human activity, so we treat it as the locations of individuals which should be kept private. The dataset has 1.63 million coordinates in the range $[-124.82, -103.00] \times [31.33, 49.00]$. We conducted experiments on other data sets as well, including synthetic 2D data with various distributions and TIGER/Line data of different regions, and obtained similar results.

We show results for rectangular queries where query sizes are expressed in terms of the original data. For example, since 1 degree is approximately 70 miles, (15, 0.2) indicates a

[3] Via the exposition at http://adamdsmith.wordpress.com/2009/09/02/sample-secrecy/

[4] http://www.census.gov/geo/www/tiger/



(a) $\varepsilon = 0.1$ (b) $\varepsilon = 0.5$ (c) $\varepsilon = 1.0$

Fig. 3. Query accuracy of quadtree optimizations

"skinny" query of 1050 × 14 miles. We consider several query shapes; for each shape we generate 600 queries that have a non-zero answer, and record the median relative error.

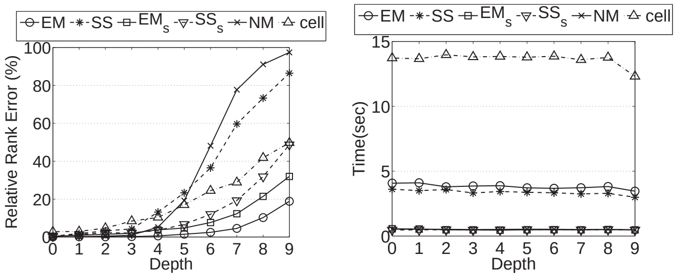
All experiments were conducted on a 2.80GHz CPU with 8GB RAM, so the data fits easily in memory. We implemented our PSDs in Python 2.6 with scientific package Numpy.

B. Experimental Results

Budget and post-processing. In Sections IV and V we proposed two techniques to improve query accuracy for PSDs: geometric noise allocation, and the post-processing method. We compare quadtrees with geometric budgeting (*quad-geo*), post-processing (*quad-post*) and both combined (*quad-opt*) to the baseline approach of uniform budget with no post-processing (*quad-baseline*). Figure 3 shows the relative errors when all trees are grown to the same height, $h = 10$. Clearly, each optimization significantly improves the query accuracy, and in combination they perform even better: the relative error is reduced by up to an order of magnitude, especially when the privacy budget is limited ($\varepsilon = 0.1$). We observe the same improvement on other PSDs, so all subsequent results are presented with both optimizations.

Quality of private medians. In Section VI-A we described several approaches to private medians: smooth sensitivity (SS), exponential mechanism (EM), noisy mean (NM) and the cell-based approach (cell). To show their relative behavior, we compare them on a synthetic one-dimensional dataset with 2^{20} data points distributed uniformly within a domain of $[0, 2^{26}]$ (the same relative performance was seen on other distributions). We build a binary tree structure with splits found by each mechanism, and measure the average (normalized) rank error of the private medians for each level. In the worst case the noisy median may fall out of the data range $[x_1, x_n]$ and have 100% relative error.

Figure 4 shows the accuracy and time for median finding with privacy budget $\varepsilon = 0.01$ at each level (counting from the root down). For SS, which is (ε, δ)-differentially private, we set $\delta = 10^{-4}$. We also show the result of combining sampling with probability $p = 1\%$ with SS and EM, obtaining methods $\mathrm{SS_s}$ and $\mathrm{EM_s}$ respectively.



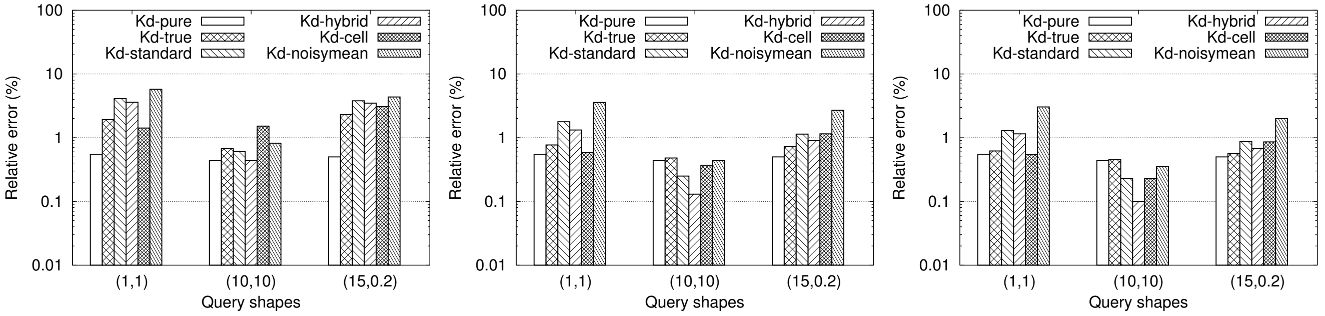
(a) Private median quality (b) Time efficiency

Fig. 4. Private medians accuracy and efficiency

We argued that efficiency could be improved by combining either method with sampling. The exponential mechanism (EM) is always the most accurate, providing almost true medians for large data sizes (2^{20} points at depth 0) and medians with moderate error for smaller data sizes (around 2^{11} points at depth 9) without sampling. For smooth sensitivity, the sampling ($\mathrm{SS_s}$) approach improves the accuracy, since the increase in privacy budget outweighs the effect of fewer data points. The reverse is true for $\mathrm{EM_s}$: although the privacy budget becomes about 50 times larger, the ranks are about 100 times smaller, so the probability associated with elements far from the median is correspondingly larger. Nevertheless, as Figure 4(b) shows, there is a commensurate performance improvement: the sampling versions are about an order of magnitude faster across all data sizes. Although fast, the noisy mean approximation (NM) gives poor quality medians for smaller data sizes. The cell-based approach (cell length 2^{10}) has larger error for large data sizes while being much slower than others. Based on these results, we use the exponential mechanism as the default method for noisy medians in all other experiments, and advocate the use of sampling when the data under consideration is large.

Other parameter settings. We summarize our findings on parameter settings. Detailed plots are omitted for brevity.

Hybrid trees. A hybrid tree switches from data dependent splitting to data independent splitting at a "switch level" ℓ. We found that switching about half-way down the tree (height

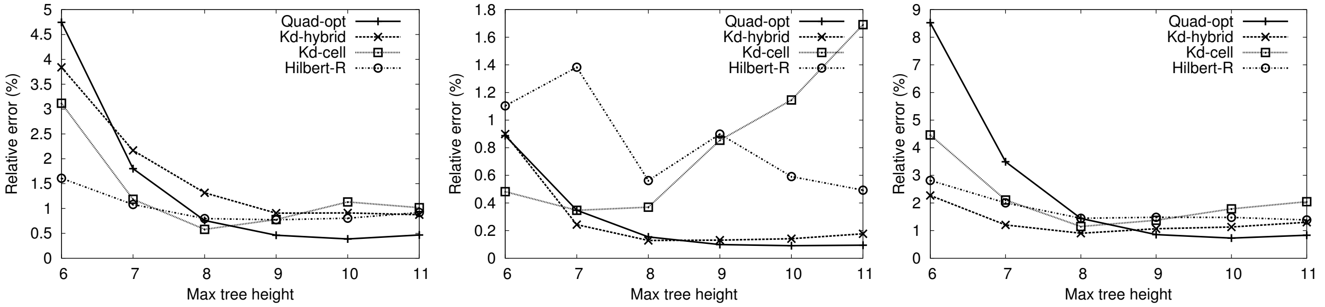


(a) $\varepsilon = 0.1$

(b) $\varepsilon = 0.5$

(c) $\varepsilon = 1.0$

Fig. 5. Query accuracy for kd-trees



(a) Query (1,1)

(b) Query (10,10)

(c) Query (15,0.2)

Fig. 6. Query accuracy comparison

3 or 4) gives the best result over this data set.

Median vs. count noise. As we increase $\varepsilon_{\text{median}}$ and decrease $\varepsilon_{\text{count}}$, we obtain more accurate divisions of the data, but less accurate counts. Although it may vary for different queries and PSDs, in most cases the best results were seen when budget was biased towards the node counts, allocated roughly as $\varepsilon_{\text{count}} = 0.7\varepsilon$ and $\varepsilon_{\text{median}} = 0.3\varepsilon$, so we adopt this division.

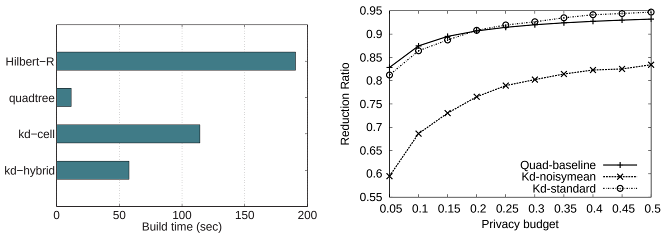
Hilbert curve resolution. Recall that we can treat Hilbert R-trees as one-dimensional kd-trees, and so obtain PSDs. First we set the *order* of the Hilbert curve. Given the large domain and skewed data distribution, a Hilbert curve of order 23 is needed to differentiate each individual point in the dataset. However, since the leaves of our PSDs should contain several points, we found similar accuracy for all resolutions in the range 16 to 24, and use curves of order 18 in our experiments.

Comparison of kd-trees. We compare the accuracy of queries among the kd-tree with EM medians (*kd-standard*), the hybrid kd-tree (*kd-hybrid*), the cell-based kd-tree proposed in [2] (*kd-cell*) with cell length 0.01 and the noisy mean based kd-tree from [4] (*kd-noisymean*). To see “the cost of privacy”, we also include some results on non-private structures: the exact kd-tree (*kd-pure*) and the kd-tree which uses exact medians but noisy counts (*kd-true*). In all experiments, the kd-trees have the same (pre-determined) maximum number of levels $h = 8$ (with fanout 4). We also use a pruning condition based on frequency: we remove descendants of any node whose noisy count is below a threshold $m = 32$.

Figure 5 shows the result of comparing kd-tree variants across a range of privacy budgets ($\varepsilon = 0.1, 0.5, 1.0$). We first observe that even a pure kd-tree without noise of any kind introduces some error because of the uniformity assumption for leaves that intersect, but are not contained in the query. Comparing *kd-true* and *kd-pure*, we see that adding noise to counts does not dramatically degrade query accuracy: it remains below 1% relative error. Rather, it is the noise in the choice of medians which seems to have the greater impact on query accuracy, as is evident from the results for the PSDs. Since the splitting points do not evenly divide the data, particularly at greater depths in the tree, there are some leaves with a much larger number of points than in the exact trees. This explains the weak performance of *kd-noisymean*, whose choice of splitting points was seen to be poor in Figure 4(a).

The *kd-cell* approach has good performance on queries which are small and square. However, as the query area grows, the method looses its edge, and is dominated by *kd-hybrid*. Overall, the accuracy of query answering from all cases is good, with small relative errors of less than 10% even for high privacy ($\varepsilon = 0.1$). Of the kd-tree variations, the kd-hybrid seems the most reliably accurate.

Comparison of PSDs. Figure 6 shows the query accuracy for the best performing instances of the representative methods: optimized quadtree, cell-based kd-tree, hybrid kd-tree and Hilbert R-tree on three queries of different shapes. In this set of experiments we varied the depth of the trees ($h = 6$



(a) Construction time (b) Results for private record matching

Fig. 7. Time efficiency and results on private record matching

to 11) while keeping the privacy budget fixed at $\varepsilon = 0.5$. At depth 10 the optimized quadtree is able to provide the best accuracy of all. It seems that the ability to devote all of the privacy budget to noisy counts for quadtrees outweighs the ability of kd-trees to more evenly split the data via private medians. However, the hybrid kd-tree at height $h = 8$ obtains about the same accuracy as the quadtree at height $h = 10$ for the larger queries. The cell-based kd-tree, which first imposes a uniform grid over the domain, provides good accuracy on small square queries since the node shape matches the query shape. However, *kd-cell* performs worst on the larger queries, which require probing more nodes. We also note that the private Hilbert R-tree has comparably good performance on some queries, but much higher errors on others.

Scalability. Figure 7(a) illustrates the time taken to build each spatial decomposition using our prototype code. For the data sets considered, PSD construction time was not a bottleneck, since we typically consider this to be a one-time cost. In general, the structures which only divide the domain take much less time than the data dependent approach, while hybrid kd-tree lies in-between, at around a minute. The cell-based kd-tree also takes a long time, due to first materializing noisy counts of many cells, then building a tree on top of them. The Hilbert R-tree takes even more time, due to the higher cost of encoding and decoding coordinates in Hilbert space. The count post-processing step adds only a few seconds to the overall cost.

C. Application to Private Record Matching.

The *kd-noisymean* tree was originally introduced to solve a private record matching problem—see the full details in [4]. Here, a differentially private data structure is used to focus in on areas of potential overlap between datasets before an expensive secure multiparty computation (SMC) procedure is invoked. The metric of interest is the *reduction ratio*, which measures how much SMC work is saved relative to the baseline of no elimination, so bigger is better. We obtained the code from the authors of [4] and ran the same experiments on the same data. Figure 7(b) shows reduction ratio for *quad-baseline*, *kd-noisymean* and *kd-standard* (in this application, all count budget is allocated to leaves and thus post-processing does not apply). We observe that the new kd-tree approach we have proposed can improve appreciably over the two methods tried in [4]. Note that improving reduction ratio from 0.93 to 0.95 represents 28% less SMC work.

IX. Concluding Remarks

We have presented a comprehensive framework for differentially private spatial decompositions, and shown how to produce private versions of many popular methods, such as quadtrees, kd-trees and Hilbert R-trees. We have proposed novel techniques for setting hierarchical noise parameters in a non-uniform manner that minimizes query error. Further, we have developed a post-processing technique that re-computes node counts based on the initial noisy counts to optimize query accuracy. This new technique applies to a large class of other settings in privacy.

In the process of computing private data-dependent decompositions, we have provided the first survey of techniques for computing private medians and derived theoretical results that give insight into the expected behavior of the two most accurate methods. We have shown how to combine all these results (plus other techniques such as sampling) into a single whole that achieves the desired privacy guarantee. In our ongoing work, we study other settings: sparse categorical data [26], and higher dimensional data.

Our experimental study is consistent with our theoretical insights, and shows that each of our techniques leads to significant improvements in query accuracy, running time or both. For most PSDs, we achieve relative query errors in single-digit percentages, while guaranteeing strong differential privacy. We conclude that PSDs represent an efficient and accurate way to release spatial data privately.

Acknowledgments. We thank Adam Smith and Michael Hay for some helpful discussions.

References

[1] C. Dwork, “Differential privacy,” in *ICALP*, 2006, pp. 1–12.

[2] Y. Xiao, L. Xiong, and C. Yuan, “Differentially private data release through multidimensional partitioning,” in *SDM Workshop at VLDB*, 2010.

[3] M. Hay, V. Rastogi, G. Miklau, and D. Suciu, “Boosting the accuracy of differentially-private histograms through consistency,” in *VLDB*, 2010.

[4] A. Inan, M. Kantarcioglu, G. Ghinita, and E. Bertino, “Private record matching using differential privacy,” in *EDBT*, 2010.

[5] B. Chen, D. Kifer, K. LeFevre, and A. Machanavajjhala, “Privacy-preserving data publishing,” *Foundations and Trends in Databases*, vol. 2, pp. 1–167, 2009.

[6] G. Cormode and D. Srivastava, “Anonymized data: Generation, models, usage,” in *SIGMOD*, 2009.

[7] C. Dwork, “A firm foundation for private data analysis,” *Commun. ACM*, vol. 54, no. 1, pp. 86–95, 2011.

[8] J. Gehrke, D. Kifer, and A. Machanavajjhala, “Privacy in data publishing,” in *ICDE*, 2010.

[9] P. Samarati and L. Sweeney, “Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression,” in *IEEE Security and Privacy*, 1998.

[10] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam, “ℓ-diversity: Privacy beyond k-anonymity,” in *ICDE*, 2006.

[11] 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*, 2007.

[12] C. Li, M. Hay, V. Rastogi, G. Miklau, and A. McGregor, “Optimizing linear counting queries under differential privacy,” in *PODS*, 2010.

[13] X. Xiao, G. Wang, and J. Gehrke, “Differential privacy via wavelet transforms,” in *ICDE*, 2010.

[14] F. McSherry and K. Talwar, “Mechanism design via differential privacy,” in *FOCS*, 2007.
[15] A. Ghosh, T. Roughgarden, and M. Sundararajan, “Universally utility-maximizing privacy mechanisms,” in *STOC*, 2009.
[16] K. Nissim, S. Raskhodnikova, and A. Smith, “Smooth sensitivity and sampling in private data analysis,” in *STOC*, 2007.
[17] V. Rastogi and S. Nath, “Differentially private aggregation of distributed time-series with transformation and encryption,” in *SIGMOD*, 2010.
[18] F. McSherry and I. Mironov, “Differentially private recommender systems: Building privacy into the netflix prize contenders,” in *KDD*, 2009.
[19] C. Dwork, F. Mcsherry, K. Nissim, and A. Smith, “Calibrating Noise to Sensitivity in Private Data Analysis,” in *Theory of Cryptography Conference*, 2006.
[20] H. Samet, *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006.
[21] M. deBerg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*. Springer, 2008.
[22] I. Kamel and C. Faloutsos, “Hilbert R-tree: An improved R-tree using fractals,” in *VLDB*, 1994.
[23] C. R. Rao, *Linear statistical inference and its applications*. Wiley, 1965.
[24] F. McSherry, “Privacy integrated queries: an extensible platform for privacy-preserving data analysis,” in *SIGMOD*, 2009.
[25] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith, “What can we learn privately?” in *FOCS*, 2008.
[26] G. Cormode, M. Procopiuc, D. Srivastava, and T. Tran, “Differentially private publication of sparse data,” in *ICDT*, 2012.

Appendix

Proof of Lemma 4.

We use the notation defined in Section V. In addition, we define for each node u the set $\mathrm{desc}(u)$ to be the descendants of u, *including node u*. We also use the shorthand $\mathrm{anc}(u,v) = \mathrm{anc}(u) \cap \mathrm{anc}(v)$. Let C_u denote the true count of node u. We want to show that if β is the OLS for C, then it satisfies the recurrence in the statement of the lemma. For any v

$$Y_v = \sum_{u \prec v} C_v + X_v \tag{3}$$

where X_v is the random variable drawn from the Laplace distribution with parameter $\varepsilon_{h(v)}$. Therefore $\mathrm{Var}(Y_v) = 2/\varepsilon^2_{h(v)}$. The set of equations (3) for all v can be written in matrix form as follows. Let $n = f^h$ be the number of leaves, and let m be the number of nodes in the tree. The binary matrix $H_{m \times n}$ representing the tree hierarchy is defined as $H_{u,v} = 1$ if $v \prec u$ and 0 otherwise. Assume wlog that the nodes u are in breadth-first order, and the leaves are ordered from left to right. Then (3) can be written as $Y = H \cdot C + X$, and the covariance matrix of Y is $\mathrm{cov}(Y) = 2\,\mathrm{diag}(1/\varepsilon^2_{h(u)})$.

We apply the standard transformations

$$Z = (\mathrm{cov}(Y)/2)^{-\frac{1}{2}} Y = \mathrm{diag}(\varepsilon_{h(u)}) Y \text{ and } U = (\mathrm{cov}(Y)/2)^{-\frac{1}{2}} H$$

to obtain the equation $Z = U \cdot C + \mathrm{diag}(\varepsilon_{h(u)}) \cdot X$. Now $\mathrm{cov}(Z) = 2I$, where I the unit matrix, and the new equation fits the classical model.

Since the OLS is consistent, i.e., $\beta_v := \sum_{u \prec v} \beta_u$, it suffices to estimate the leaf counts, and the other estimates follow. A vector β is the OLS for the true count C if it minimizes $(Z - U\beta)^T (Z - U\beta)$ (equivalent to Definition 3). After differentiating, we have that β satisfies

$$U^T U \beta = U^T Z \tag{4}$$

By simple calculations, $(U^T U)_{u,w} = \sum_{v \in \mathrm{anc}(u,w)} \varepsilon^2_{h(v)}$ and $(U^T Z)_u = \sum_{v \in \mathrm{anc}(u)} \varepsilon_{h(v)} Z_v = \sum_{v \in \mathrm{anc}(u)} \varepsilon^2_{h(v)} Y_v$. For any node v, we sum the corresponding rows on the left side of (4) to obtain $\sum_{u \prec v} (U^T U)_u \beta =$

$$\begin{aligned}
&\sum_{z \in [n]} \sum_{u \prec v} \sum_{w \in \mathrm{anc}(u,z) \setminus \mathrm{anc}(v)} \varepsilon^2_{h(w)} \beta_z + \sum_{z \in [n]} \sum_{u \prec v} \sum_{w \in \mathrm{anc}(u,z) \cap \mathrm{anc}(v)} \varepsilon^2_{h(w)} \beta_z \\
&= \sum_{z \prec v} \sum_{u \prec v} \sum_{w \in \mathrm{anc}(u,z) \setminus \mathrm{anc}(v)} \varepsilon^2_{h(w)} \beta_z + \sum_{w \in \mathrm{anc}(v)} \sum_{u \prec v} \sum_{z \prec w} \varepsilon^2_{h(w)} \beta_z \\
&= \sum_{u \prec v} \sum_{w \in \mathrm{anc}(u) \setminus \mathrm{anc}(v)} \sum_{z \prec w} \varepsilon^2_{h(w)} \beta_z + \sum_{w \in \mathrm{anc}(v)} \varepsilon^2_{h(w)} \sum_{u \prec v} \left(\sum_{z \prec w} \beta_z \right) \\
&= \sum_{u \prec v} \sum_{w \in \mathrm{anc}(u) \setminus \mathrm{anc}(v)} \varepsilon^2_{h(w)} \beta_w + \sum_{w \in \mathrm{anc}(v)} f^{h(v)} \varepsilon^2_{h(w)} \beta_w \\
&= \sum_{j=0}^{h(v)-1} \varepsilon^2_j \sum_{u \prec v} \sum_{w \in \mathrm{anc}(u): h(w) = j} \beta_w + f^{h(v)} \sum_{w \in \mathrm{anc}(v)} \varepsilon^2_{h(w)} \beta_w \\
&= \sum_{j=0}^{h(v)-1} \varepsilon^2_j \sum_{w \in \mathrm{desc}(v): h(w) = j} f^j \beta_w + f^{h(v)} \sum_{w \in \mathrm{anc}(v)} \varepsilon^2_{h(w)} \beta_w \\
&= \sum_{j=0}^{h(v)-1} \varepsilon^2_j f^j \beta_v + f^{h(v)} \varepsilon^2_{h(v)} \beta_v + f^{h(v)} \sum_{w \in \mathrm{anc}(v) \setminus \{v\}} \varepsilon^2_{h(w)} \beta_w \\
&= \sum_{j=0}^{h(v)} \varepsilon^2_j f^j \beta_v + f^{h(v)} \sum_{w \in \mathrm{anc}(v) \setminus \{v\}} \varepsilon^2_{h(w)} \beta_w
\end{aligned}$$

We now sum the corresponding rows on the right hand side of (4), i.e., we sum $(U^T Z)_u$ over $u \prec v$. Equating the left and right hand side, we obtain (2). ■

Proof of Lemma 6. (i)

$$\begin{aligned}
\text{Observe that } \sigma_s \quad &\le \max\Big(\max_{0 \le k < 2n/5} e^{-k\xi} (x_{m+k+1} - x_{m-k-1}), \\
&\qquad \max_{2n/5 \le k \le n} e^{-k\xi} (x_{m+k+1} - x_{m-k-1}) \Big) \\
&\le \max\big((x_{4n/5} - x_{n/5}), e^{-2\xi n/5} M \big)
\end{aligned}$$

Using our assumptions, $e^{-2\xi n/5} M < M/5 \le (x_{4n/5} - x_{n/5})$, so $\sigma_s \le (x_{4n/5} - x_{n/5})$. Consider the case when the median is closer to $x_{n/5}$ than to $x_{4n/5}$. Then the output of SS is within the desired range if $0 \le (2\sigma_s/\varepsilon) X \le \sigma_s/2$, i.e., $X \le \varepsilon/4$. Symmetrically, if the median is closer to $x_{4n/5}$, the noisy median is in the desired range if $0 \ge (2\sigma_s/\varepsilon) X \ge -\sigma_s/2$, i.e., $X \ge -\varepsilon/4$. Since X is drawn from a symmetric distribution, we conclude that

$$\Pr[\mathrm{SS}(C) \in [x_{n/5}, x_{4n/5}] \,|\, \xi n \ge 4.03] > \Pr[0 \le X \le \varepsilon/4] = \tfrac{1 - e^{-\frac{\varepsilon}{4}}}{2}.$$

(ii) Let α be the proportionality constant implicit in Definition 5. Let E be the event that $\mathrm{EM}(C) \in [x_{n/5}, x_{4n/5}]$. Then

$$\begin{aligned}
\Pr[E] \quad &\ge \alpha e^{-0.15 \varepsilon n} (x_{4n/5} - x_{n/5}) \\
\text{and } 1 - \Pr[E] = \Pr[\neg E] \quad &\le \alpha e^{-0.15 \varepsilon n} M.
\end{aligned}$$

Using our assumption, we have $1 - \Pr[E] \le 5 \Pr[E]$. Hence $\Pr[E] \ge 1/6$. ■