



# HTF: Homogeneous Tree Framework for Differentially-Private Release of Location Data

Sina Shaham<sup>1</sup>, Gabriel Ghinita<sup>2</sup>, Ritesh Ahuja<sup>1</sup>, John Krumm<sup>3</sup>, Cyrus Shahabi<sup>1</sup>

{sshaham,rahuja,shahabi}@usc.edu,gabriel.ghinita@umb.edu,jckrumm@microsoft.com

<sup>1</sup>University of Southern California, <sup>2</sup>University of Massachusetts Boston, <sup>3</sup>Microsoft Research

## ABSTRACT

Mobile apps that use location data are pervasive, spanning domains such as transportation, urban planning and healthcare. Important use cases for location data rely on statistical queries, e.g., identifying hotspots where users work and travel. Such queries can be answered efficiently by building histograms. However, precise histograms can expose sensitive details about individual users. Differential privacy (DP) is a mature and widely-adopted protection model, but most approaches for DP-compliant histograms work in a data-independent fashion, leading to poor accuracy. The few proposed data-dependent techniques attempt to adjust histogram partitions based on dataset characteristics, but they do not perform well due to the addition of noise required to achieve DP. We identify *density homogeneity* as a main factor driving the accuracy of DP-compliant histograms, and we build a data structure that splits the space such that data density is homogeneous within each resulting partition. We show through extensive experiments on large-scale real-world data that the proposed approach achieves superior accuracy compared to existing approaches.

## CCS CONCEPTS

• Security and privacy → Privacy protections.

## KEYWORDS

Location Protection, Differential Privacy

### ACM Reference Format:

Sina Shaham<sup>1</sup>, Gabriel Ghinita<sup>2</sup>, Ritesh Ahuja<sup>1</sup>, John Krumm<sup>3</sup>, Cyrus Shahabi<sup>1</sup>. 2021. HTF: Homogeneous Tree Framework for Differentially-Private Release of Location Data. In *29th International Conference on Advances in Geographic Information Systems (SIGSPATIAL '21)*, November 2–5, 2021, Beijing, China. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3474717.3483943>

## 1 INTRODUCTION

Statistical analysis of location data, typically collected by mobile apps, helps researchers and practitioners understand patterns within the data, which in turn can be used in various domains such as transportation, urban planning and public health. At the same time,

significant privacy concerns arise when locations are directly accessed. Sensitive details about individuals, such as political or religious affiliations, alternative lifestyle habits, etc., can be derived from users' whereabouts. Therefore, it is essential to account for user privacy and protect location data.

*Differential privacy (DP)* [9] is a well-established protection model for statistical data processing. DP allows answering aggregate queries (e.g., count, sum) while hiding the presence of any specific individual within the data. In other words, the query results do not permit an adversary to infer with significant probability whether a certain individual's record is present in the dataset or not. DP achieves protection by injecting random noise in the query results according to well-established rules. It is a powerful semantic model adopted by both government entities (e.g., Census Bureau) as well as major industry players.

In the location domain, existing DP-based approaches build a spatial index structure, and perturb index node counts using random noise. Subsequently, queries are answered based on the noisy node counts. Building DP-compliant index structures has several benefits: first, querying indexes is a natural approach for most existing spatial processing techniques; second, using an index helps quantify and limit the amount of disclosure, which becomes infeasible if one allows arbitrary queries on top of the exact data; third, query efficiency is improved. Due to large amounts of background knowledge data available to adversaries (e.g., public maps, satellite imagery), information leakage may occur both from query answers, as well as from the properties of the indexing structure itself. To deal with the structure leakage, initial approaches used *data-independent* index structures, such as quad-trees, binary space partitioning trees, or uniform grids (UG). No structural leakage occurred, and the protection techniques focused on improving the signal-to-noise ratio in query answers. However, such techniques perform poorly when the data distribution is skewed.

More recently, data-dependent approaches emerged, such as adaptive grids (AG), or kd-tree based approaches [15, 20]. AG overcomes the rigidity of UG by providing a two-level grid, where the first level has fixed granularity, and the second uses a granularity derived from the coarse results obtained from the first level. While it achieves improvement, it is still a rather blunt tool to account for high variability in dataset density, which is quite typical of real-life datasets. Other more sophisticated approaches tried to build kd-trees or R-trees in a DP-compliant way, but to do so they used DP mechanisms such as the exponential mechanism (EM) (discussed in Section 2) which are difficult to tune and may introduce significant errors in the data. In fact, the work in [20] shows that data-dependent structures based on EM fail to outperform AG.

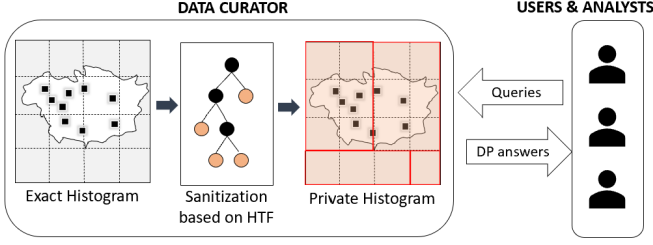
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

*SIGSPATIAL '21*, November 2–5, 2021, Beijing, China

© 2021 Association for Computing Machinery.

ACM ISBN 978-1-4503-8664-7/21/11...\$15.00

<https://doi.org/10.1145/3474717.3483943>



**Figure 1: System model for private location histograms.**

Our proposed *Homogeneous Tree Framework* (HTF) is addressing the problem of DP-compliant location protection using a data-dependent approach that focuses on building index structures with *homogeneous intra-node density*. Our key observation is that density homogeneity is the main factor influencing the signal-to-noise ratio for DP-compliant spatial queries (we discuss this aspect in detail in Section 3). Rather than using complex mechanisms like EM which have high sensitivity, we derive theoretical results that can directly link index structure construction with intra-node data density based on the lower-sensitivity Laplace mechanism (introduced in Section 2). This novel approach allows us to build effective index structures capable of delivering low query error without excessive consumption of privacy budget. HTF is custom-tailored for capturing areas of homogeneous density in the dataset, which leads to significant accuracy gains. Our specific contributions are:

- We identify data homogeneity as the main factor influencing query accuracy in DP-compliant spatial data structures;
- We propose a custom technique for homogeneity-driven DP-compliant space partitioning based on the Laplace mechanism, and we perform an in-depth analysis of its sensitivity;
- We derive effective DP budget allocation strategies to balance the noise added during the building of the structure with that used for releasing query answers;
- We propose a set of heuristics to automatically tune data structure parameters based on data properties, with the objective of minimizing overall error in query answering;
- We perform an extensive empirical evaluation showing that HTF outperforms existing state-of-the-art on real and synthetic datasets under a broad range of privacy requirements.

The rest of the paper is organized as follows: Section 2 presents background information and introduces the problem definition. Section 3 provides the overview of the proposed framework, followed by technical details in Section 4. We evaluate our approach empirically against the state-of-the-art in Section 5. We survey related work in Section 6 and conclude in Section 7.

## 2 BACKGROUND AND DEFINITIONS

Private publication of location histograms follows the two-party system model shown in Fig. 1. The data owner/curator first builds an exact histogram with the distribution of locations on the map. Non-trusted users/analysts are interested in learning the population distribution over different areas, and perform statistical queries. The goal of the curator is to publish the location histogram without the privacy of any individual being compromised. To this end, the exact histogram undergoes a sanitizing process according to DP to generate a *private histogram*. In our proposed method, a tree-based

**Table 1: Summary of notations.**

Symbol	Description
$\epsilon_{\text{tot}}$	Total privacy budget
$\epsilon_{\text{height}}, \epsilon_{\text{data}}$	Height estimation, data perturbation budget
$\epsilon_{\text{prt}}, \epsilon'_{\text{prt}}, \epsilon''_{\text{prt}}$	Partitioning budget: total, per level, per round
$o_k$	Objective function output for index $k$
$c_{ij}$	Number of data points in row $i$ and column $j$
$h$	Tree height

algorithm is applied for protection, and the tree's nodes, representing a private histogram of the map, are released to the public. Analysts/researchers ask unlimited *count queries* that are answered from the private histogram. Furthermore, they may download the whole private histogram, and the protection method remains strong enough to protect the identity of individuals in the database.

### 2.1 Differential Privacy

Consider two databases  $\mathcal{D}$  and  $\mathcal{D}'$  that differ in a single record  $t$ , i.e.,  $\mathcal{D}' = \mathcal{D} \cup \{t\}$  or  $\mathcal{D}' = \mathcal{D} \setminus \{t\}$ .  $\mathcal{D}$  and  $\mathcal{D}'$  are commonly referred to as *neighboring* or *sibling*.

**Definition 1** ( $\epsilon$ -Differential Privacy[8]). A randomized mechanism  $\mathcal{A}$  provides  $\epsilon$ -DP if for any pair of neighbor datasets  $\mathcal{D}$  and  $\mathcal{D}'$ , and any  $S \in \text{Range}(\mathcal{A})$ ,

$$\frac{\Pr(\mathcal{A}(\mathcal{D}) = S)}{\Pr(\mathcal{A}(\mathcal{D}') = S)} \leq e^\epsilon \quad (1)$$

Parameter  $\epsilon$  is referred to as privacy budget.  $\epsilon$ -DP requires that the output  $S$  obtained by executing mechanism  $\mathcal{A}$  does not significantly change by adding or removing one record in the database. Thus, an adversary is not able to infer with significant probability whether an individual's record was included or not in the database. An important property of DP is *composability* [10]: running in succession multiple mechanisms that satisfy DP with privacy budgets  $\epsilon_1, \epsilon_2, \dots, \epsilon_n$ , results in  $\epsilon$ -differential privacy where  $\epsilon = \sum_{i=1}^n \epsilon_i$ .

There are two common methods to achieve differential privacy: the *Laplace mechanism* and the *exponential mechanism* (EM). Both approaches are closely related to the concept of *sensitivity*, which captures the maximal difference achieved in the output by adding or removing a single record from the database.

**Definition 2** ( $L_1$ -Sensitivity[9]). Given sibling datasets  $\mathcal{D}, \mathcal{D}'$  the  $L_1$ -sensitivity of a set  $f = \{f_1, \dots, f_m\}$  of real-valued functions is:

$$\Delta f = \max_{\mathcal{D}, \mathcal{D}'} \sum_{i=1}^m |f_i(\mathcal{D}) - f_i(\mathcal{D}')|$$

**2.1.1 Laplace Mechanism.** The Laplace mechanism is a widely used technique to achieve  $\epsilon$ -DP. It adds to the output of a query function  $f$  noise drawn from Laplace distribution  $\text{Lap}(b)$  with scale  $b$ , where  $b$  depends on two factors: sensitivity and privacy budget.

$$\text{Lap}(x|b) = \frac{1}{2b} e^{-|x|/b} \text{ where } b = \frac{\Delta f}{\epsilon} \quad (2)$$

To simplify notation, we denote Laplace noise by  $\text{Lap}(\frac{\Delta f}{\epsilon})$ , as it only depends on the sensitivity and budget.

**2.1.2 Exponential Mechanism.** The exponential mechanism (EM) is another approach to achieve  $\epsilon$ -DP when the output of a computation is not numerical. With EM, the output is drawn from a probability distribution chosen based on a utility function. Consider the generalized problem where for an input  $d$ , output  $s$  is chosen from the space denoted by  $S$ , i.e.  $s \in S$ . The utility function  $u$  takes as input two parameters  $d$  and  $s$ , and returns a real value  $r$  measuring the quality of  $s$  as a solution for the input  $x$ . EM aims to determine in a differentially private way  $\max_{s \in S} \{u(x, s)\}$ . In general a single record may have a significant impact on the utility, hence the required noise may grow large, leading to poor accuracy.

## 2.2 Problem Formulation

Consider a two-dimensional location dataset  $D$  discretized to an arbitrarily-fine  $N \times M$  grid. Each point is represented by its corresponding *rectangular cell* in the grid. We study the problem of releasing DP-compliant *histograms* to answer count queries as accurately as possible. Cell counts are modeled via an  $N \times M$  *frequency matrix*, in which the entry in the  $i^{\text{th}}$  row and  $j^{\text{th}}$  column represents the number of records located inside cell  $(i, j)$  of the grid.

A DP histogram is generated based on a non-overlapping partitioning of the frequency matrix by applying methods to preserve  $\epsilon$ -DP. The DP histogram consists of the boundary of *partitions* and their noisy counts, where each partition consists of a group of cells.

Let us denote the total count of a partition with  $q$  cells by  $c$  and its noisy count by  $\bar{c}$ . There are two sources of error in answering a query. The first is referred to as *noise error*, which is due to Laplace noise added to the partition count. The second source of noise is referred to as *uniformity error* and arises when a query has partial overlap with a partition. An assumption of uniformity is made within the partition, and the answer *per cell* is calculated as  $\bar{c}/q$ .

For example, consider the  $3 \times 3$  grid shown in Fig. 2a, where each count represents the number of data points in the corresponding cell. The cells are grouped in four partitions  $C_1$ ,  $C_2$ ,  $C_3$ , and  $C_4$ , each entailing 0, 12, 4 and 2 data points, respectively. Independent noise with the same magnitude is added to each partition's count denoted by  $n_1$ ,  $n_2$ ,  $n_3$ , and  $n_4$ , and released to the public as a DP histogram. The result of the query shown by the dashed rectangle can be calculated as  $(12 + n_2)/4 + (2 + n_4)/2$ .

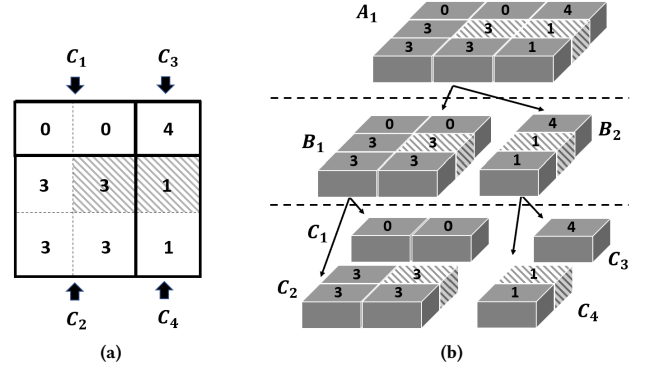
**PROBLEM 1.** *Generate a DP histogram of dataset  $D$ , such that the expected value of relative error (MRE) is minimized, where for a query  $q$  with the true count  $c$  and noisy count  $\bar{c}$  RE is calculated as*

$$MRE(q) = \frac{|c - \bar{c}|}{c} \times 100 \quad (3)$$

In the past, several approaches have been developed for Problem 1. Still, current solutions have poor accuracy, which limits their practicality. Some methods tend to perform better when applied to specific datasets (e.g., uniform) and quite poorly when applied to others. Limitations of existing work have been thoroughly evaluated in [13], and we review them in Section 6.

## 3 HOMOGENEOUS-TREE FRAMEWORK

Our proposed approach relies on two key observations to reduce the noise error and uniformity error. To address *noise error*, one needs to carefully calibrate the sensitivity of each operation performed,



**Figure 2: Example of HTF partitioning. The dashed rectangles show the query.**

in order to reduce the magnitude of required noise. We achieve this objective by carefully controlling the depth of the indexing structure. To control the impact of *uniformity error*, we guide our structure-construction algorithm such that each resulting partition (i.e., internal node or leaf node) has a homogeneous data distribution within its boundaries.

Homogeneity ensures that uniformity error is minimized, since a query that does not perfectly align with the boundaries of an internal/leaf node is answered by scaling the count within that node in proportion with the overlap between the query and the node. None of the existing works on DP-compliant data structures has directly addressed homogeneity. Furthermore, conventional spatial indexing structures (designed for non-private data access) are typically designed to optimize criteria other than homogeneity (e.g., reduce node area or perimeter, control the data count balance across nodes). As a result, existing approaches that use such structures underperform when DP-compliant noise is present.

We propose a *Homogeneous-Tree Framework (HTF)* which builds a customized spatial index structure specifically designed for DP-compliant releases. We address directly aspects such as selection of structure height, a homogeneity-driven node split strategy, and careful tuning of privacy budget for each structure level. Our proposed data structure shares similarities with kd-trees, due to the specific requirements of DP: namely, (1) nodes should not overlap, since that would result in increased sensitivity, and (2) the leaf set should cover the entire data domain, such that an adversary cannot exclude specific areas by inspecting node boundaries. However, as shown in previous work, using kd-trees directly for DP releases leads to poor accuracy [7, 13].

Similar to kd-trees, HTF divides a node into two sub-regions across a split dimension, which is alternated through successive levels of the tree. The root node covers the whole dataspace. Figure 2b provides an example of a non-private simplified version of the proposed HTF construction applied on a  $3 \times 3$  grid (frequency matrix). HTF consists of three steps:

(A) *Space partitioning* aims to find an enhanced partitioning of the map such that the accuracy of the private histogram is maximized. HTF performs heuristic partitioning based on a *homogeneity metric* we define. At every split, we choose the coordinate that results in the highest value of the homogeneity metric. For example, in the running example (Fig. 2b) node  $B_1$  is split into  $C_1$  and  $C_2$ ,

which are homogeneous partitions. However, the metric evaluation is not straightforward in the private case, as metric computation for each candidate split point consumes privacy budget. We use the Laplace mechanism to determine an advantageous split point without consuming large amounts of budget. As part of HTF, a search mechanism is used to select plausible candidates for evaluation and find a near-optimal split position. The total privacy budget allocated for the private partitioning is denoted by  $\epsilon_{\text{prt}}$ .

(B) *Data sanitization* starts by traversing the tree generated in the partitioning step. At each node, a certain amount of budget is used to perturb the node count using the Laplace mechanism. Based on the sanitized count, HTF evaluates the *stop condition* (i.e., whether to follow the downstream path from that node or release it as is), which is an important aspect in building private data structures. The private evaluation of stop conditions enables HTF to avoid over- or under-partitioning of the space, and preserve good accuracy. Revisiting the example in Fig. 2b, suppose that we do not want to further partition the space when the number of data points in a node is less than 7. Once HTF reaches node  $B_2$ , the actual node count (6) is noise-perturbed. The value of the sanitized count may be less than 7 after sanitization, leading to pruning at  $B_2$  and stopping further partitioning. Finally, the tree's leaf set (i.e., sanitized count of each leaf node) is released to the public. The total budget used for data sanitization is denoted by  $\epsilon_{\text{data}}$ .

(C) *Height estimation* is another important HTF step. Tree height is an important factor in improving accuracy, as it influences the budget allocated at each index level. HTF dedicates a relatively small amount of budget ( $\epsilon_{\text{height}}$ ) to determine an appropriate height.

The total budget consumption of HTF ( $\epsilon_{\text{tot}}$ ) is the sum of budgets used in each of the three steps:

$$\epsilon_{\text{tot}} = \epsilon_{\text{prt}} + \epsilon_{\text{data}} + \epsilon_{\text{height}} \quad (4)$$

The DP composition rules in the case of HTF apply as follows:

- *Sequential decomposition*: The sum of budgets used for node splits along every tree path adds up to the total budget available for partitioning.
- *Parallel decomposition*: The budget allocated for partitioning nodes in the same level is independent, since the nodes at the same level have non-overlapping extents.

## 4 TECHNICAL APPROACH

Section 4.1 introduces the split objective function used in HTF, and provides its sensitivity analysis. Section 4.2 focuses on HTF index structure construction. Section 4.3 presents the data perturbation algorithm used to protect leaf node counts.

### 4.1 Homogeneity-based Partitioning

Previous approaches that used kd-tree variations for DP-compliant indexes preserved the original split heuristics of the kd-tree: namely, node splits were performed on either median or average values of the enclosed data points. To preserve DP, the split positions were computed using the exponential mechanism (Section 2) which computes a merit function for each candidate split. However, such an approach results in poor query accuracy [13].

We propose *homogeneity* as the key factor for guiding splits in the HTF index structure. This decision is based on the observation

that if all data points are uniformly distributed within a node, then the uniformity error that results when intersecting that node with the query range is minimized. At each index node split, we aim to obtain two new nodes with a high degree of intra-node density homogeneity. Of course, since the decision is data-dependent, the split point must be computed in a DP-compliant fashion.

For a given node of the tree, suppose that the corresponding partitioning covers  $U \times V$  cells of the  $N \times M$  grid (i.e., frequency matrix), in which the count of data points located in its  $i^{\text{th}}$  row and  $j^{\text{th}}$  column is denoted by  $c_{ij}$ . Without loss of generality, we discuss the partitioning method w.r.t. the horizontal axis (i.e., rows). The aim is to find an index  $k$  which groups rows 1 to  $k$  into one node and rows  $k+1$  to  $U$  into another, such that homogeneity is maximized within each of the *resulting* nodes (we also refer to resulting nodes as *clusters*). We emphasize that the input grid abstraction is used in order to obtain a finite set of candidate split points. This is different than alternate approaches that use grids to obtain DP-compliant releases. Furthermore, the frequency matrix can be arbitrarily fined-grained, so discretization does not impose a significant constraint.

The proposed split objective function is formally defined as:

$$o_k = \sum_{i=1}^k \sum_{j=1}^V |c_{ij} - \mu_1| + \sum_{i=k+1}^U \sum_{j=1}^V |c_{ij} - \mu_2|, \quad (5)$$

where

$$\mu_1 = \frac{\sum_{i=1}^k \sum_{j=1}^V c_{ij}}{k \times V}, \quad \mu_2 = \frac{\sum_{i=k+1}^U \sum_{j=1}^V c_{ij}}{(U-k) \times V}. \quad (6)$$

The optimal index  $k^*$  minimizes the objective function.

$$k^* = \arg \min_k o_k \quad (7)$$

Consider the example in Figure 2b and the partitioning conducted for node  $B_1$ . There exist three possible ways to split rows of the frequency matrix: (i) separate the top row of cells resulting in clusters  $\{[0,0]\}$  and  $\{[3,3],[3,3]\}$  yielding the objective value of zero in Eq. (5); (ii) separate the bottom row of cells resulting in two clusters  $\{[0,0],[3,3]\}$  and  $\{[3,3]\}$  yielding the objective value of 6, or (iii) no division is performed, yielding the objective value of 8. Therefore, the proposed algorithm will select the first option ( $k^*=2$ ), generating two nodes  $C_1$  and  $C_2$ .

Note that the value of  $k^*$  is not private, since individual location data were used in the process of calculating the optimal index. Hence, a DP mechanism is required to preserve privacy. Thus, we need to assess the sensitivity of  $k^*$ , which represents the maximum change in the split coordinate that can occur when adding or removing a single data point. The sensitivity calculation is not trivial, since a single data point can cause the optimal split to shift to a new position far from the non-private value. Another challenge is that the exponential mechanism, commonly used in literature to select candidates from a set based on a cost function, tends to have high sensitivity, resulting in low accuracy.

**4.1.1 Baseline Split Point Selection.** We propose a DP-compliant homogeneity-driven split point selection technique based on the Laplace mechanism. As before, consider  $U \times V$  frequency matrix of a given node and a horizontal dimension split. Denote by  $o_k$  the objective function for split coordinate  $k$  among the  $U$  candidates.

There are  $U$  possible outputs  $\mathcal{O} = (o_1, o_2, \dots, o_U)$ , one for each split candidate. In a non-private setting, the index corresponding to the minimum  $o_i$  value would be chosen as the optimal division. To preserve DP, given that the partitioning budget per computation is  $\epsilon''_{\text{prt}}$ , we add independent Laplace noise to each  $o_i$ , and then select the optimal value among all noisy outputs.

$$\bar{\mathcal{O}} = (\bar{o}_1, \bar{o}_2, \dots, \bar{o}_U) = \mathcal{O} + \mathbf{Lap}(2/\epsilon''_{\text{prt}}), \quad (8)$$

where  $\mathbf{Lap}(2/\epsilon''_{\text{prt}})$  denotes a tuple of  $U$  independent samples of Laplace noise. Note that since the grid is fixed, enumerating split candidates as cell coordinates is data-independent, hence does not incur disclosure risk. The Laplace noise added to each  $o_i$  is calibrated according to a sensitivity of 2, as proved in Theorem 1:

**THEOREM 1 (SENSITIVITY OF PARTITIONING).** *The sensitivity of cost function  $o_k$  for any given horizontal or vertical index  $k$  is 2.*

**PROOF.** In the calculation of objective function  $o$  for a given index  $k$ , adding or removing an individual data point affects only one cell and the corresponding cluster. The objective function for split point  $k$  can be written as

$$o_k = \sum_{i=1}^k \sum_{j=1}^V |c_{ij} - \mu_1| + \sum_{i=k+1}^U \sum_{j=1}^V |c_{ij} - \mu_2|, \quad (9)$$

The modified objective function value following addition of a single record to an arbitrary cell  $c_{xy}$  can be represented as

$$o'_k = \sum_{i=1}^k \sum_{j=1}^V |c'_{ij} - \mu'_1| + \sum_{i=k+1}^U \sum_{j=1}^V |c'_{ij} - \mu'_2|. \quad (10)$$

Without loss of generality, assume that the additional record is located in the first cluster which results in  $\mu'_1 = \mu_1 + 1/kV$ ,  $\mu'_2 = \mu_2$ , and  $c'_{ij}$  being equal to  $c_{ij}$  for all possible  $i$  and  $j$  except for  $c_{xy}$  where we have  $c'_{xy} = c_{xy} + 1$ . Therefore, the sensitivity of the objective function has value 2 as follows:

$$\Delta o_k = o_k - o'_k \leq \frac{2(kV - 1)}{kV} \leq 2 \quad (11)$$

Eq. (11) is derived using the *reverse triangle inequality*:

$$\left| |c_{ij} - \mu_1 - \frac{1}{kV}| - |c_{ij} - \mu_1| \right| \leq \frac{1}{kV} \quad (12)$$

$$\forall \{ij\} | i \in \{1, \dots, k\} \wedge j \in \{1, \dots, V\}, ij \neq xy$$

and

$$\left| |c_{xy} + 1 - \mu_1 - \frac{1}{kV}| - |c_{xy} - \mu_1| \right| \leq \frac{kV - 1}{kV} \quad (13)$$

Similarly, the sensitivity upper bound corresponding to an individual record's removal can be shown to be 2.  $\square$

We refer to the above approach as the *baseline* approach. One challenge with the baseline is that the calculation of noise is performed separately for each candidate split point, and since the computation depends on all data points within the parent node, the budget consumption adds up according to *sequential composition*. This means that the calculation of each individual split candidate in  $\bar{o}_i$  may receive only  $1/U$  of the budget available for that level.

For large values of  $U$ , the privacy budget per computation becomes too small, decreasing accuracy. This leads to an interesting

trade-off between the number of split point candidates evaluated and the accuracy of the entire release. On one hand, increasing the number of candidates leads to a higher likelihood of including the optimal split coordinate in the set  $\bar{\mathcal{O}}$ ; on the other hand, there will be more noise added to each candidate's objective function output, leading to the selection of a sub-optimal candidate. Next, we propose an optimization which finds a good compromise between number of candidates and privacy budget per candidate.

**4.1.2 Optimized Split Point Selection.** We propose an optimization that aims to minimize the number of split point candidate evaluations required, and searches for a local minimum rather than the global one. Algorithm 1 outlines the approach for a single split step along the  $y$ -axis (i.e., row split). Inputs to Algorithm 1 include (i) the frequency matrix  $F_{U \times V}$  of the parent node, (ii) the total budget allocated for the partitioning per level of the tree  $\epsilon'_{\text{prt}}$ , and (iii) variable  $T$  which bounds the maximum number of objective function computations – a key factor indicating the extent of search, and thus of the budget per operation. The proposed approach is essentially a search tree, determining the candidate split to minimize the objective function's output. The search starts from a wide range of candidates and narrows down within each interval until reaching a local minimum, similar to a binary search.

Let  $\{l, \dots, r\}$  represent the index range where the search is conducted, initially set to the first and last possible index of the input frequency matrix. At every iteration of the main 'for' loop, the search interval is divided into four equal length sub-intervals, including three *inner* points and two *boundary* point. The inner points are referred to as *split indices*. The objective function is calculated for each of these candidates, and perturbed using Laplace noise to satisfy DP. The split corresponding to the minimum value is chosen as the center of the next search interval, and its immediate 'before' and 'after' split positions are assigned as the updated search boundaries  $l$  and  $r$ ). Hence, in every iteration, two new computations of the objective function are performed, except the first run which has a single computation. Therefore, the total number of private evaluations sums to  $(2T + 1)$  each perturbed with the privacy budget of  $\epsilon''_{\text{prt}} = \epsilon'_{\text{prt}} / (2T + 1)$ .

## 4.2 HTF Index Structure Construction

Our proposed HTF index structure is built in accordance to the split point selection algorithm introduced in Section 4.1. The HTF construction pseudocode is presented in Algorithm 2. Each node stores the rectangular spatial extent of the node (*node.region*), its children (*node.left* and *node.right*), real data count (*node.count*), noisy count (*node.ncount*), and the node's height in the tree.

The root of the tree represents the entire data domain ( $N \times M$  frequency matrix) and its height is denoted by  $h$ . Deciding the height of the tree is a challenging task: a large height will result in a smaller amount of privacy budget per level, whereas a small one does not provide sufficient granularity at the leaf level, decreasing query precision. We estimate an advantageous height value using a small amount of budget ( $\epsilon_{\text{height}}$ ) to perturb the total number of data records based on the Laplace mechanism:

$$|\bar{D}| = |D| + \text{Lap}(1/\epsilon_{\text{height}}). \quad (14)$$



**Algorithm 1** Near-optimal Split Point Estimator

---

```

1: function GETSPLITPOINT(  $F_{U \times V}$ ,  $\epsilon'_{\text{prt}}$ ,  $axis$ ,  $T$  )
2:    $\epsilon''_{\text{prt}} \leftarrow \epsilon'_{\text{prt}} / (2T + 1)$ 
3:    $l \leftarrow 1$ ,  $r \leftarrow (axis == 0) ? V : U$  #  $axis = 0$  means  $x$ -split
4:    $k \leftarrow \lfloor (r - l) / 2 \rfloor$ 
5:   Compute  $o_k$  at  $axis$  according to Eq. (5)
6:    $\bar{o}_k \leftarrow o_k + \text{Lap}(2/\epsilon''_{\text{prt}})$ 
7:   while  $l \leq r$  and  $T > 0$  do
8:      $k_1 \leftarrow \lfloor (k - l) / 2 \rfloor$ 
9:      $k_2 \leftarrow \lfloor (r - k) / 2 \rfloor$ 
10:    Compute  $o_{k_1}$  and  $o_{k_2}$  at  $axis$  according to Eq. (5)
11:     $(\bar{o}_{k_1}, \bar{o}_{k_2}) \leftarrow (o_{k_1}, o_{k_2}) + \text{Lap}(2/\epsilon''_{\text{prt}})$ 
12:     $MinOutput \leftarrow \min(\bar{o}_{k_1}, \bar{o}_k, \bar{o}_{k_2})$ 
13:    if  $MinOutput = \bar{o}_k$  then
14:       $l \leftarrow k_1$ ,  $r \leftarrow k_2$ 
15:    else if  $MinOutput = \bar{o}_{k_1}$  then
16:       $k \leftarrow k_1$ ,  $r \leftarrow k$ 
17:    else
18:       $l \leftarrow k$ ,  $k \leftarrow k_2$ 
19:     $T \leftarrow T - 1$ 
20:  return  $k$ 

```

---

Next, we set the height to:

$$h = \log_2 \left( \frac{|D|\epsilon_{\text{tot}}}{10} \right) \quad (15)$$

The formula is motivated by the work in [20]. The authors show that when data are uniformly distributed in space, using a grid with a lower granularity of  $\sqrt{\frac{|D|\epsilon_{\text{tot}}}{c_0}} \times \sqrt{\frac{|D|\epsilon_{\text{tot}}}{c_0}}$  improves the mean relative error, where the value of constant  $c_0$  is set to 10 experimentally. We emphasize that the approach does not indicate that the number of leaves on the tree is  $\frac{|D|\epsilon_{\text{tot}}}{c_0}$ , but the information contained in this number is merely used as an estimator of the tree's height. This estimation is formally characterized in [13] and referred to as *scale-epsilon exchangeability property*. The intuition is that the error due to decreasing the amount of budget used for the estimation is offset by having a larger number of data points in the entire dataset.

The last input to the algorithm is the budget allocated per level of the partitioning tree. We use *uniform budget allocation* to allocate the budget between levels denoted as  $\epsilon'_{\text{prt}} = \epsilon_{\text{prt}}/h$ .

Starting from the root node, the proposed algorithm recursively creates two child nodes and decreases height by one. This is done by splitting the underlying area of the node into two hyperplanes based on Algorithm 1. The division is done on the  $y$  dimension if the current height is an even number and in the  $x$  dimension otherwise. The algorithm continues until reaching the minimum height of zero, or to a point where no further splitting is possible.

### 4.3 Leaf Node Count Perturbation

Once the HTF structure is completed, the final step of our algorithm is to release DP-compliant counts for index nodes, so that answers

**Algorithm 2** DP space partitioning

---

```

1: function PRIVATEPARTITIONING( $node$ ,  $\epsilon'_{\text{prt}}$ )
2:   if  $node.height=0$  then
3:     return  $node$ 
4:    $axis \leftarrow node.height \bmod 2$ 
5:    $node.count \leftarrow sum(node.region)$ 
6:    $OptIdx \leftarrow \text{GetSplitPoint}(node.FreqMatrix, \epsilon'_{\text{prt}}, axis, T)$ 
7:    $leftC, rightC \leftarrow \text{split } node \text{ on } OptIdx$ 
8:    $node.leftChild \leftarrow \text{PrivatePartitioning}(leftChild, \epsilon'_{\text{prt}})$ 
9:    $node.rightChild \leftarrow \text{PrivatePartitioning}(rightChild, \epsilon'_{\text{prt}})$ 

```

---

to queries can be reconstructed from the noisy counts. The total partitioning budget adds up to  $\epsilon_{\text{height}} + \epsilon_{\text{prt}}$ , where  $\epsilon_{\text{height}}$  was used to estimate the tree height and  $\epsilon_{\text{prt}}$  budget to generate the private partitioning tree. The data perturbation step uses the remaining  $\epsilon_{\text{data}}$  amount of budget and releases node counts according to the Laplace mechanism.

One can choose various strategies to release index node counts. At one extreme, one can simply release a noisy count for each index node; in this case, the budget must be shared across nodes on the same path (sequential composition), and can be re-used across different paths (parallel composition). This approach has the advantage of simplicity, and may do well when queries exhibit large variance in size – it is well-understood that when perturbing large counts, the relative error is much lower, since the Laplace noise magnitude only depends on sensitivity, and not the actual count.

However, in practice, queries tend to be more localized, and one may want to allocate more budget to the lower levels of the structure, where the actual counts are smaller, thus decreasing relative error. In fact, as another extreme, one can concentrate the entire  $\epsilon_{\text{data}}$  on the leaf level. However, doing so can also decrease accuracy, since some leaf nodes have very small real counts.

Our approach takes a middle ground, where the available  $\epsilon_{\text{data}}$  is spent to (i) determine which nodes to publish and (ii) ensure sufficient budget remains for the noisy counts. Specifically, we publish only leaf nodes, but these are not the same leaves returned by the structure construction algorithm. Instead, we perform an additional pruning step which uses the noisy counts of internal nodes to determine a *stop condition*, i.e., the level at which a node count is likely to be small enough that a further recursion along that path is not helpful to obtain good accuracy. Effectively, we perform pruning of the tree using a small fraction of the data budget, and then split the remaining budget among the non-pruned nodes along a path. This helps decrease the *effective height* of the tree across each path, and hence the resulting budget per level increases.

Next, we present in detail our approach that contributes two main ideas: (i) how to determine smart stop (or pruning) conditions based on noisy internal node counts, and (ii) how to allocate perturbation budget across shortened paths.

The proposed technique is summarized in Algorithm 3: it takes as inputs the root node of the tree generated in the data partitioning step; the remaining budget allocated for the perturbation of data ( $\epsilon_{\text{data}}$ ); a tracker of accumulated budget ( $\epsilon_{\text{accu}}$ ); a stop condition predicate denoted by *cond*; and the nominal tree height  $h$  as computed in Section 4.2. Similar to prior work [7], we use a *geometric*

*progression* budget allocation strategy, but we enhance it to avoid wasting budget on unnecessarily long paths. The intuition behind this strategy is to assign more budget to the nodes located in the lower levels of the tree, since their actual counts are lower, and hence larger added noise impacts the relative error disproportionately high. Conversely, at the higher levels of the tree, where actual counts are much higher, the effect of the noise is negligible.

Eq. (16) formulates this goal as a convex optimization problem.

$$\min_{\epsilon_0 \dots \epsilon_h} \sum_{i=0}^h 2^{h-i} / \epsilon_i^2 \quad (16)$$

$$\text{where} \quad (17)$$

$$\sum_{i=0}^h \epsilon_i = \epsilon, \quad \epsilon_i > 0 \quad \forall i = 0 \dots h \quad (18)$$

Writing Karush-Kuhn-Tucker (KKT) [4] conditions, the optimal allocation of budget can be calculated as:

$$L(\epsilon_1, \dots, \epsilon_h, \lambda) = \sum_{i=0}^h 2^{h-i} / \epsilon_i^2 + \lambda \left( \sum_{i=0}^h \epsilon_i - \epsilon \right) \quad (19)$$

$$\Rightarrow \frac{\partial L}{\partial \epsilon_i} = -\frac{2^{h-i+1}}{\epsilon_i^3} + \lambda = 0 \quad (20)$$

$$\Rightarrow \epsilon_i = \frac{2^{h-i+1}}{\lambda^{1/3}}, \quad (21)$$

and substituting  $\epsilon_i$ 's in the constraint of problem the optimal budget in the  $i$ -th level is derived as

$$\epsilon_i = \frac{2^{(h-i)/3} \epsilon (2^{1/3} - 1)}{(2^{(h+1)/3} - 1)}. \quad (22)$$

The algorithm starts the traversal from the partitioning tree's root and recursively visits the descendent nodes. Once a new node is visited, the first step is to use the node's height to determine the allocated budget ( $\epsilon'_{\text{data}}$ ) based on geometric progression. Recall that the nodes on the same level follow parallel decomposition of the budget as their underlying areas in space do not overlap. Additionally, the algorithm keeps track of the amount of budget used so far on the tree, optimizing the budget in later stages. Next, the computed value of  $\epsilon'_{\text{data}}$  is utilized to perturb the *node.count* by adding Laplace noise, resulting in noisy count *node.ncount*.

The stop condition we use takes into account the noisy count in the current internal node (i.e., count threshold); and the spatial extent of the internal node threshold (i.e., extent threshold). If none of the thresholds is met for the current node, the algorithm recursively visits the node's children; otherwise, the algorithm prunes the tree considering that the current node should be a leaf node. In the latter case, the algorithm subtracts the accumulated budget used so far on that path from the root, and uses the entire remaining budget available to perturb the count. This significantly improves the utility, as geometric allocation tends to save most of the budget for the lower levels of the tree. Revisiting the example in Figure 2b, suppose that the stop condition is to prune when the underlying area consists of less than four cells. During the data perturbation process, the node  $B_2$  is turned into a leaf node due to its low number of cells. At this point, the node's children are

---

### Algorithm 3 DP data perturbation

---

```

1: function PERTURBER(node,  $\epsilon_{\text{data}}$ ,  $\epsilon_{\text{accu}}$ , cond, h)
2:    $i \leftarrow \text{node.height}$ 
3:    $\epsilon'_{\text{data}} \leftarrow \frac{2^{(h-i)/3} \epsilon_{\text{data}} (2^{1/3} - 1)}{(2^{(h+1)/3} - 1)}$ 
4:    $\epsilon_{\text{accu}} = \epsilon_{\text{accu}} + \epsilon'_{\text{data}}$ 
5:    $\text{node.ncount} = \text{node.count} + \text{Lap}(1/\epsilon'_{\text{data}})$ 
6:   if  $\text{node.ncount} \leq \text{cond}$  then
7:      $\epsilon_{\text{remain}} = \epsilon_{\text{data}} - \epsilon_{\text{accu}}$ 
8:      $\text{node.ncount} = \text{node.count} + \text{Lap}(1/\epsilon_{\text{remain}})$ 
9:      $\text{node.leftChild} = \text{node.rightChild} = \mathbf{null}$ 
10:  else
11:    Perturber(node.leftChild,  $\epsilon_{\text{data}}$ ,  $\epsilon_{\text{accu}}$ , cond, h)
12:    Perturber(node.rightChild,  $\epsilon_{\text{data}}$ ,  $\epsilon_{\text{accu}}$ , cond, h)

```

---

removed, and its noisy count is determined based on the remaining budget available on the lower levels of the tree.

## 5 EXPERIMENTAL EVALUATION

### 5.1 Experimental Setup

We evaluate HTF on both real and synthetic datasets:

**Los Angeles Dataset.** This is a subset of the Veraset dataset [22]<sup>1</sup>, including location measurements of cell phones within LA city. In particular, we consider a large geographical region covering a  $70 \times 70 \text{ km}^2$  area centered at latitude 34.05223 and longitude -118.24368. The selected data generates a frequency matrix of 3.5 million data points during a time period between Jan 1-7 2020.

**Synthetic dataset.** We generate locations according to a Gaussian distribution as follows: a cluster center, denoted by  $(x_c, y_c)$ , is selected uniformly at random. Next, coordinates for each data point  $x$  and  $y$  are drawn from a Gaussian distribution with the mean of  $x_c$  and  $y_c$ , respectively. We model three sparsity levels by using three standard deviation ( $\sigma$ ) settings for Gaussian variables: low ( $\sigma = 20$ ), medium ( $\sigma = 50$ ), and high ( $\sigma = 100$ ) sparsity.

We discretize the space to a  $1024 \times 1024$  frequency matrix. We use as performance metric the mean relative error (MRE) for range queries. Similar to prior work [13, 20, 23, 26], we consider a *smoothing factor* of 20 for the relative error, to deal with cases when the true count for a query is zero (i.e., relative error is not defined). Each experimental run consists of 2,000 random rectangular queries with center selected uniformly at random. We vary the size of queries to a region covering {2%, 6%, 10%} of the dataspace.

### 5.2 HTF vs Data Dependent Algorithms

Data-dependent algorithms aim to exploit users' distribution to provide an enhanced partitioning of the map. The state-of-the-art data-dependent approach for DP-compliant location publication is the kd-tree technique from [7]. The kd-tree algorithm generates the partitioning tree by splitting on median values, which are determined using the exponential mechanism. We have also included the smoothing post-processing step from [14, 20] which resolves

<sup>1</sup>Veraset is a data-as-a-service company that provides anonymized population movement data collected through location measurement signals of cell phones across USA.

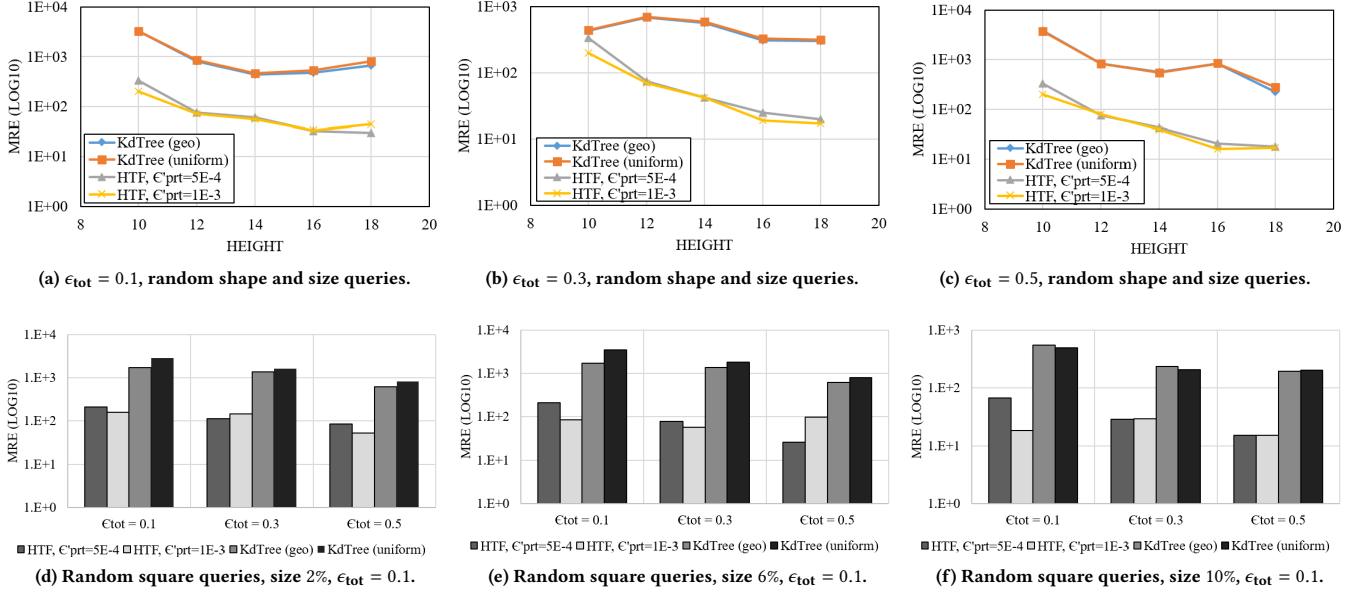


Figure 3: Comparison with Data Dependent Algorithms, Los Angeles Dataset.

inconsistencies within the structure (e.g., using the fact that counts in a hierarchy should sum to the count of an ancestor).

Fig. 3 presents the comparison of the HTF algorithm with kd-tree approaches, namely: (i) geometric budget allocation in addition to smoothing and post-processing labelled as *KdTree (geo)*; (ii) uniform budget allocation including smoothing and post-processing labelled as *KdTree (uniform)*; (iii) HTF algorithm with the partitioning budget per level set to  $\epsilon'_{\text{prt}} = 5E-4$ ; and (iv) HTF algorithm with  $\epsilon'_{\text{prt}} = 1E-3$ . Recall that  $\epsilon'_{\text{prt}}$  denotes the budget per level of partitioning, and therefore, given the tree's height as  $h$ , the remaining budget for perturbation is derived as  $\epsilon_{\text{data}} = \epsilon_{\text{tot}} - \epsilon'_{\text{prt}} \times h - \epsilon_{\text{height}}$ . The value of  $\epsilon_{\text{height}}$  in the experiments is set to  $1E-4$ , the HTF's  $T$  value is set to 3, and stop condition thresholds are set to no less than 5 cells or 100 data points. Moreover, for the kd-tree algorithm, 15% of the total budget is allocated to implement the partitioning.

In Figs. 3a, 3b, and 3c, the MRE performance is compared for different values of  $\epsilon_{\text{tot}}$  over a workload of uniformly located queries with random shape and size. HTF clearly outperforms kd-tree for all height settings. Looking at the MRE performance, the kd-tree algorithm follows a parabolic shape commonly occurring in tree-based algorithms, meaning that the MRE performance reaches its best values at a particular height, and further partitioning of the space increases the error. This is caused by excessive partitioning in low-density areas. HTF, on the other hand, is applying stop conditions to avoid the adverse effects of over-partitioning, and is able to estimate the optimal height beforehand. Figs. 3d, 3e, and 3f, show the results when varying query size (for square shape queries). HTF outperforms the kd-tree algorithm significantly.

### 5.3 HTF vs Grid-based Algorithms

Grid-based approaches are mostly data-independent, and they partition the space using regular grids. The uniform grid (UG) approach uses a single-layer fixed size grid, whereas its successor adaptive

grid (AG) method considers two layers of regular grids: the first layer is similar to UG, whereas the second uses a small amount of data-dependent information (i.e., noisy query results on the first layer) to tune the second layer grid granularity.

Fig. 4 presents the comparison of HTF with AG and UG. For HTF, we consider several stop condition thresholds. HTF consistently outperforms grid-based approaches, especially when the total privacy budget is lower (i.e., more stringent privacy requirements). The impact of the stop count condition on HTF depends on the underlying distribution of data points. For the Los Angeles dataset, MRE is relatively larger for small values of  $\epsilon_{\text{tot}}$ . The performance improves and reaches its near-optimal values around stop count 50, and ultimately worsens when the stop count becomes larger. This matches our expectation as, on one hand, small values of stop count result in over-partitioning, and on the other hand, when the stop count is too large, the partitioning tree cannot reach the ideal heights, resulting in high MRE values.

Figures 4d, 4e, and 4f, show the obtained results for varying query sizes (2, 6, and 10% of the data domain). HTF outperforms both AG and UG. Note that all three algorithms are adaptive and change their partitioning according to the number of data points. Therefore, in low privacy regimes, the structure of algorithms may cause fluctuations in the accuracy. However, as the privacy budget grows, the algorithms reach their maximum partitioning limit, and increasing the budget always results in lower MRE.

### 5.4 HTF vs Data Independent Algorithms

The most prominent DP-compliant data-independent technique [7] uses QuadTrees. The technique recursively partitions the space into four equal size quadrants. Two commonly used budget allocation techniques used in [7] are geometric budget allocation (geo) and uniform budget allocation. For a fair comparison, we have also included the smoothing post-processing step from [20, 24].



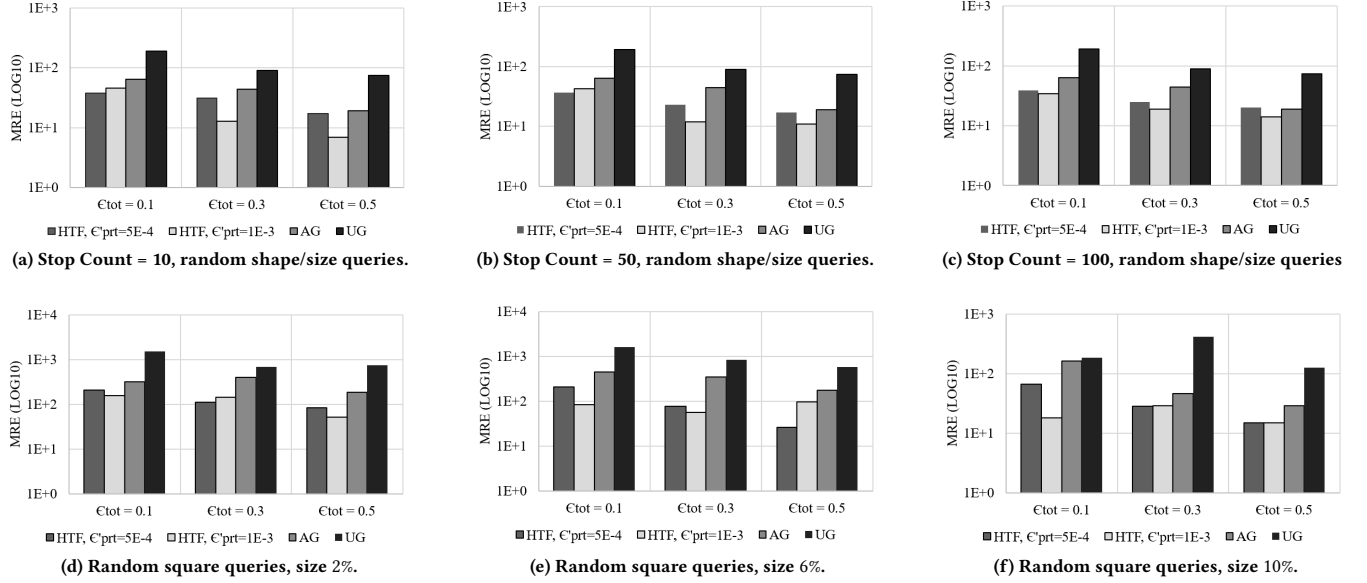


Figure 4: Comparison to Grid-based Algorithms, Los Angeles Dataset.

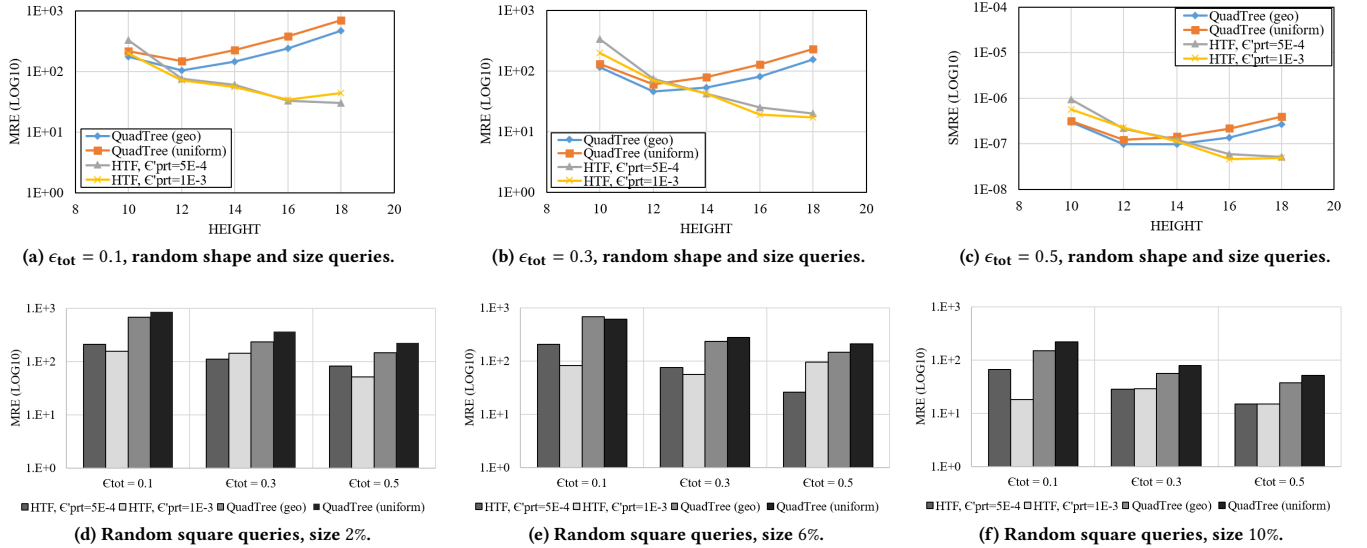


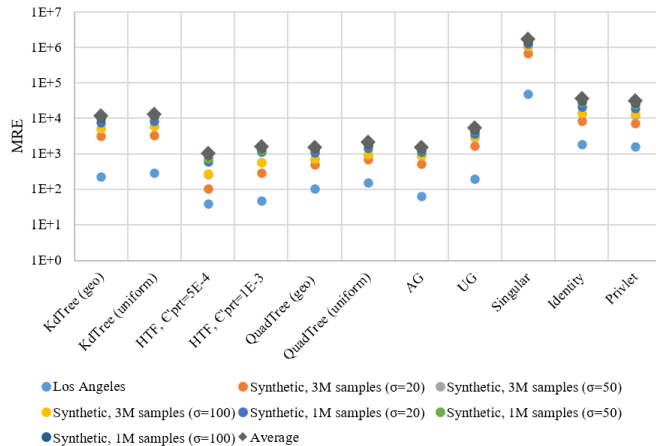
Figure 5: Comparison to Data Independent Algorithms, Los Angeles Dataset.

Fig. 5 presents the comparison results. Figures 5a, 5b, 5c are generated using random shape and size queries, and several different height settings. Note that the fanout of QuadTrees is double the fanout of HTF, so the height represented by  $2k$  in the figures corresponds to the implementation of QuadTree with the height of  $k$ . The error of the QuadTree approach is large for small heights, then improves to its optimal value, and rises again significantly as height further increases, due to over-partitioning. Similar to kd-trees, no systematic method has been developed for QuadTree to determine optimal height, whereas the HTF height selection heuristic yields levels 15, 16, and 17 for the allocated privacy budget of 0.1, 0.3, and 0.5, respectively. HTF outperforms QuadTree for all settings of  $\epsilon_{tot}$ . Figures 5d, 5e, and 5f show the accuracy of HTF and QuadTree

for square-shaped randomly placed queries of varying size. HTF outperforms Quadtree in all cases.

## 5.5 Additional Benchmarks

To further validate HTF performance, we run experiments on the Los Angeles dataset as well as six synthetic datasets generated using Gaussian distribution. Fig. 6 presents the comparison of all algorithms with randomly generated query workload and a privacy budget of  $\epsilon_{tot} = 0.1$ . Three additional algorithms are used as comparison benchmarks, (i) *Singular* algorithm—preserving differential privacy by adding independent Laplace noise to each entry of the frequency matrix, (ii) *Uniform* algorithm in which Laplace noise is added to the total count of the grid with the assumption that data



**Figure 6: Mixed workloads,  $\epsilon_{\text{tot}} = 0.1$ , All Datasets.**

are uniformly distributed within the grid, and (iii) the *Privet* [23] algorithm based on wavelet transformations.

Fig. 6 shows that HTF consistently outperforms existing approaches. For the denser datasets ( $\sigma = 20$ ) the gain compared to approaches designed for uniform data (e.g., UG, AG, Quadtrees) is lower. As data sparsity grows, the difference in accuracy between HTF and the benchmarks increases. HTF performs best on a relative basis for lower  $\epsilon'_{\text{prt}}$ , i.e., more stringent privacy requirements.

## 6 RELATED WORK

The comprehensive benchmark analysis in [13] showed that the dimensionality and scale of the data directly determine the accuracy of an algorithm. For one-dimensional data, both data-dependent and data-independent methods perform well. The hierarchical method in [14] uses a strategy consisting of hierarchically structured range queries arranged as a tree. Similar methods (e.g., [26]) differ in their approach to determining the tree’s branching factor and allocating appropriate budget to each of its level. Data-dependent techniques on the other hand exploit correlation in real-world datasets in order to boost the accuracy of histograms. They first compress the data without loss: for example, EPPA [1] applies the Discrete Fourier Transform, whereas DAWA [16] uses dynamic programming to compute the least cost partitioning. The compressed data is then sanitized, for example, directly with LPM [1] or with a greedy algorithm that tunes the privacy budget to a sample query set given in advance [16]. *Privet* [23] compresses data through a wavelet transformation such that the noise incurred by a range query scales proportionately to the logarithm of its length.

In the 2D scenario, the main focus is on spatial datasets that exhibit sparse and skewed data distributions, where only data-dependent approaches tend to be competitive. General-purpose mechanisms such as the matrix mechanism of Li and Miklau [17, 18] and its workload-aware counterpart DAWA [16] operate over a discrete 1D domain, and may be extended to spatial data by applying a Hilbert transform to the 2D data representation [13]. However, approaches specialized for answering spatial range queries, such as UG [20], AG [20], QuadTree [7] and kd-tree [25] outperform general-purpose mechanisms [13]. Xiao et al. [25] present the earliest attempt at a DP-compliant spatial decomposition algorithm based on the kd tree. It first imposes a uniform grid over the data

domain, and then constructs a private kd tree over the cells in the grid. While the simplicity of the approach is appealing, the split criterion is solely based on the median, leading to high-sensitivity and split partitions with low intra-node homogeneity.

Recent work focuses on high-dimensional data, where the key idea is to reduce the impact of the higher dimensionality. The most accurate algorithm in this class is High-Dimensional Matrix Mechanism (HDMM) [19] which represents queries and data as vectors and uses sophisticated optimization and inference techniques to answer them. DPCube [24] searches for dense sub-cubes to release privately. Some of the privacy budget is used to obtain noisy counts over a regular partitioning, which is then refined to a standard kd-tree. Fresh noisy counts for the partitions are obtained with the remaining budget, and a final inference step resolves inconsistencies between the two sets of counts.

In contrast to DP-compliant aggregate statistics published by a data curator, significant work has also been devoted to preventing the data curators themselves (such as a location based service provider) from inferring a mobile user’s location in the online setting [2, 11, 21]. Spatial  $k$ -anonymity (SKA) [11, 12] generalizes the specific position of the querying user to a region that encloses at least  $k$  users. Geo-indistinguishability [2] extends the DP definition to the Euclidean space and obfuscates user check-ins to protect the exact location coordinates. Synthesizing privacy-preserving location trajectories is explored in [3]. Finally, reporting high-order statistics of mobility data in the context of DP has been pursued in [5, 6, 26], where the focus is on releasing trajectories using noisy counts of prefixes or  $n$ -grams in a trajectory.

## 7 CONCLUSIONS AND FUTURE WORK

We proposed a novel approach to privacy-preserving release of location histograms with differential privacy guarantees. Our technique capitalizes on the key observation that density homogeneity within partitions of the constructed index structure reduces the error introduced by DP noise. We devised low-sensitivity strategies for finding split coordinates in a DP-compliant way, and implemented effective privacy budget allocation strategies across different stages of the data sanitization process. In future work, we plan to extend our approach to trajectory data, which are more challenging due to their high-dimensionality. We also plan to combine our data sanitization techniques with machine learning approaches, in order to further boost the accuracy of DP-compliant location queries.

## ACKNOWLEDGEMENT

This research has been funded in part by NSF grants IIS-1910950, IIS-1909806 and CNS-2027794, the USC Integrated Media Systems Center (IMSC), and an unrestricted cash gift from Microsoft Research. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of any of the sponsors such as the NSF.

## REFERENCES

- [1] G. Acs, C. Castelluccia, and R. Chen. Differentially private histogram publishing through lossy compression. In *2012 IEEE 12th International Conference on Data Mining*, pages 1–10. IEEE, 2012.

- [2] M. E. Andrés, N. E. Bordenabe, K. Chatzikokolakis, and C. Palamidessi. Geoindistinguishability: Differential privacy for location-based systems. In *ACM CCS*, 2013.
- [3] V. Bindschaedler and R. Shokri. Synthesizing plausible privacy-preserving location traces. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 546–563. IEEE, 2016.
- [4] S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
- [5] R. Chen, G. Acs, and C. Castelluccia. Differentially private sequential data publication via variable-length n-grams. In *ACM CCS*, pages 638–649, 2012.
- [6] R. Chen, B. C. Fung, B. C. Desai, and N. M. Sossou. Differentially private transit data publication: A case study on the montreal transportation system. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 213–221, 2012.
- [7] G. Cormode, C. Procopiuc, D. Srivastava, E. Shen, and T. Yu. Differentially private spatial decompositions. In *2012 IEEE 28th International Conference on Data Engineering*, pages 20–31. IEEE, 2012.
- [8] C. Dwork. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pages 1–19. Springer, 2008.
- [9] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.
- [10] C. Dwork, A. Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.
- [11] G. Ghinita, K. Zhao, D. Papadias, and P. Kalnis. A reciprocal framework for spatial k-anonymity. *Information Systems*, 35(3):299–314, 2010.
- [12] M. Gruteser and D. Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 31–42. ACM, 2003.
- [13] M. Hay, A. Machanavajjhala, G. Miklau, Y. Chen, and D. Zhang. Principled evaluation of differentially private algorithms using dpbench. In *Proceedings of the 2016 International Conference on Management of Data*, pages 139–154, 2016.
- [14] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *Proc. VLDB Endow.*, 3(1–2):1021–1032, Sept. 2010.
- [15] A. Inan, M. Kantarcioglu, G. Ghinita, and E. Bertino. Private record matching using differential privacy. In *Proceedings of the 13th International Conference on Extending Database Technology*, pages 123–134, 2010.
- [16] C. Li, M. Hay, G. Miklau, and Y. Wang. A data-and workload-aware algorithm for range queries under differential privacy. *Proceedings of the VLDB Endowment*, 7(5):341–352, 2014.
- [17] C. Li and G. Miklau. An adaptive mechanism for accurate query answering under differential privacy. *Proc. VLDB Endow.*, 5(6):514–525, 2012.
- [18] C. Li and G. Miklau. Optimal error of query sets under the differentially-private matrix mechanism. In *Proceedings of the 16th International Conference on Database Theory*, pages 272–283, 2013.
- [19] R. McKenna, G. Miklau, M. Hay, and A. Machanavajjhala. Optimizing error of high-dimensional statistical queries under differential privacy. *Proc. VLDB Endow.*, 11(10):1206–1219, 2018.
- [20] W. Qardaji, W. Yang, and N. Li. Differentially private grids for geospatial data. In *2013 IEEE 29th international conference on data engineering (ICDE)*, pages 757–768. IEEE, 2013.
- [21] D. Quercia, I. Leontiadis, L. McNamara, C. Mascolo, and J. Crowcroft. Spotme if you can: Randomized responses for location obfuscation on mobile phones. In *2011 31st International Conference on Distributed Computing Systems*, pages 363–372. IEEE, 2011.
- [22] Veraset. Veraset Movement data for the USA, The largest, deepest and broadest available movement dataset (anonymized GPS signals). <https://datarade.ai/data-products/veraset-movement-data-for-the-usa-the-largest-deepest-and-broadest-available-movement-dataset-veraset>, 2021. [Online; accessed 19-May-2021].
- [23] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. *IEEE Transactions on knowledge and data engineering*, 23(8):1200–1214, 2010.
- [24] Y. Xiao, L. Xiong, L. Fan, S. Goryczka, and H. Li. Dpcube: Differentially private histogram release through multidimensional partitioning. *7(3):195–222*, 2014.
- [25] Y. Xiao, L. Xiong, and C. Yuan. Differentially private data release through multidimensional partitioning. In *Workshop on Secure Data Management*, pages 150–168. Springer, 2010.
- [26] J. Zhang, X. Xiao, and X. Xie. Privtree: A differentially private algorithm for hierarchical decompositions. In *Proceedings of the 2016 International Conference on Management of Data*, pages 155–170, 2016.