Building Quadtrees for Spatial Data under Local Differential Privacy

Ece Alptekin and M. Emre Gursoy

Department of Computer Engineering, Koç University, Istanbul, Turkey
{ealptekin21,emregursoy}@ku.edu.tr

Abstract. Spatial decompositions are commonly used in the privacy literature for various purposes such as range query answering, spatial indexing, count-of-counts histograms, data summarization, and visualization. Among spatial decomposition techniques, quadtrees are a popular and well-known method. In this paper, we study the problem of building quadtrees for spatial data under the emerging notion of Local Differential Privacy (LDP). We first propose a baseline solution inspired from a state-of-the-art method from the centralized DP literature and adapt it to LDP. Motivated by the observation that the baseline solution causes large noise accumulation due to its iterative strategy, we then propose a novel solution which utilizes a single data collection step from users, propagates density estimates to all nodes, and finally performs structural corrections to the quadtree. We experimentally evaluate the baseline solution and the proposed solution using four real-world location datasets and three utility metrics. Results show that our proposed solution consistently outperforms the baseline solution, and furthermore, the resulting quadtrees provide high accuracy in practical tasks such as spatial query answering under conventional privacy levels.

Keywords: Privacy · local differential privacy · location-based services · spatial data · spatial decompositions.

1 Introduction

Location-based services (LBSs) have become ubiquitous, thanks to the popularity of smartphones, GPS-equipped mobile devices, online social networks, and connected car applications. This results in growing volumes of spatial data which are available for collection, storage and analysis; however, the privacy of the data must be protected. A common method to deal with big spatial data in the privacy literature has been to adopt *spatial decompositions* which partition (decompose) the geometric space into smaller subspaces through the likes of tree-based and grid-based data structures. Spatial decompositions have been commonly used in the differential privacy literature for various purposes such as range query answering [8, 29, 34], spatial indexing and modeling [26, 39], count-of-counts histograms [22], data summarization and visualization [2, 27], as well as synthetic data generation [15, 16, 19].

Among spatial decomposition techniques, quadtrees are a popular and well-known method [8, 23, 28]. Indeed, the problem of building quadtrees under centralized differential privacy (DP) has been actively studied [8, 24, 39, 26, 34]. However, a major

drawback of centralized DP is that it assumes all data is collected and stored in a centralized location before the algorithm is executed (i.e., before a quadtree is built). On the other hand, the recently emerging notion of *Local Differential Privacy (LDP)* enables each user to locally perturb their data on their own device and send the perturbed output to the data collector [7, 13]. This way, the data collector cannot observe the true data (in our context, the true location) of any user; yet, the data collector can still build a quadtree by estimating aggregate statistics pertaining to the general population. Due to its desirable privacy guarantee, LDP has recently received significant attention from the academia and industry [7, 14, 37]. It has also been deployed in consumer-facing products of tech companies such as Apple, Google and Microsoft [1, 10, 13].

Motivated by the above, in this paper, we study the problem of building quadtrees for spatial data under LDP. We first propose a *baseline solution* inspired from a state-of-the-art method from the centralized DP literature [39] and adapt it to LDP by making modifications. However, we observe that this baseline solution yields low utility in practice. This is because the baseline solution divides the total ε privacy budget into several pieces, each used for LDP estimation in one depth of the quadtree. Since the privacy budget used per estimation is smaller, higher noise is added to satisfy LDP, which causes noisy node densities and erroneous structure in the resulting quadtrees since estimation results also affect whether an internal tree node should split (have children). To address these weaknesses, we then develop a new, *proposed solution* which performs a single LDP estimation. This single-step estimation is performed at the hypothetical leaves of the quadtree and results in low noise, thereby enabling high-quality estimates to be obtained at the leaves. Then, leaves' densities are propagated upwards (in bottom-up fashion) to populate the densities of remaining nodes. Finally, structural corrections and consistency is achieved for the quadtree in top-down manner.

We experimentally evaluate the baseline solution and the proposed solution using four real-world location datasets and three utility metrics: Average Query Error (AQE) which measures the correctness of query results issued on the quadtrees, Tree Edit Distance (TED) which measures the structural difference between the LDP quadtree and a noise-free, gold-standard quadtree, and Node Density Difference (NDD) which measures the error in the densities of quadtree nodes. Typical privacy values such as $\varepsilon = 0.1, 0.5, 1, 2$ are used throughout the experiments. Results show that our proposed solution consistently outperforms the baseline solution in a variety of settings and quadtree parameters. Furthermore, in many cases, quadtrees built using our proposed solution enable query answering with less than 5% error, demonstrating the practicality and usefulness of our solution.

In short, the contributions of this paper can be summarized as follows:

- We formulate the problem of building quadtrees for spatial data under LDP and propose two solutions: a *baseline solution* inspired from the centralized DP literature, and a *proposed solution* which relies on a single step of LDP data collection to address the iterative noise accumulation in the baseline solution.
- We develop three utility metrics to measure the accuracy of quadtrees built under LDP: Average Query Error (AQE), Tree Edit Distance (TED) and Node Density Difference (NDD).
- We experimentally evaluate the baseline and proposed solutions using four-real world location datasets and the three aforementioned metrics, under varying ε pri-

vacy budgets as well as varying quadtree parameters (max height h^* and split threshold θ). Results show that the proposed solution is superior to the baseline solution, and furthermore, quadtrees built using the proposed solution can be used in practice with low utility loss.

2 Preliminaries

2.1 Data Model and Notation

Consider a two-dimensional geolocation space Ω and let $\mathcal{U} = \{u_1, u_2, u_3, ...\}$ be the collection of users, where $n = |\mathcal{U}|$ denotes the number of users. For each user $u_i \in \mathcal{U}$, the user's true location l_i is a tuple consisting of latitude and longitude coordinates. Each location l_i falls within the boundaries of Ω. For example, if users are from the city of London, then Ω corresponds to the geographic boundaries of London and each location l_i corresponds to some GPS coordinates within London. We assume that Ω by itself is not sensitive, as it can be learned from public resources.

2.2 Local Differential Privacy

Local Differential Privacy (LDP) has recently emerged as a popular privacy standard and it has been deployed in consumer-facing products of companies such as Apple, Google and Microsoft [1, 7, 10, 13, 37]. In LDP, each user locally perturbs their true data on their own device using a randomized algorithm Ψ and sends the perturbed output to the server (i.e., the data collector). After the server collects perturbed data from many users, it performs estimation to recover aggregate statistics pertaining to the general user population. The main reasons why LDP is a good fit for the problem considered in this paper are twofold. First, since perturbation is performed on the user's side, the server does not observe the true location of any user, which protects the privacy of the user's location from a potentially untrusted server. Second, while the server cannot infer any particular user's true location, it can recover aggregate statistics pertaining the general user population, which enables the server to estimate location densities and build spatial decomposition structures pertaining to the overall population.

In our context, the sensitive data that needs to be protected using LDP is each user's location. Accordingly, we formalize LDP as follows.

Definition 1 (ε-LDP). *A randomized algorithm Ψ satisfies ε-local differential privacy (ε-LDP), where $\varepsilon > 0$, if and only if for any two inputs l_i, l_i^*, it holds that:*

$$\forall y \in Range(\Psi): \quad \frac{Pr[\Psi(l_i) = y]}{Pr[\Psi(l_i^*) = y]} \leq e^{\varepsilon} \tag{1}$$

where $Range(\Psi)$ stands for the set of all possible outputs of the algorithm Ψ.

Given the perturbed output y, ε-LDP ensures that the server (or any third party who observes y) will not be able to distinguish between user's actual location l_i and fake location l_i^* beyond the probability odds ratio controlled by e^{ε}. The strength of privacy

is controlled by the parameter ε, commonly known as the *privacy budget*. Lower ε yields stronger privacy.

Similar to centralized DP, LDP enjoys the *sequential composition* property [11, 33, 38] which can be formalized as follows.

Definition 2 (Sequential Composition). *Consider algorithms $\Psi_1, \Psi_2, ..., \Psi_m$ such that each Ψ_j satisfies ε_j-LDP. Then, the sequential execution of $\Psi_1(l_i), \Psi_2(l_i), ..., \Psi_m(l_i)$ satisfies $(\sum_{j=1}^{m} \varepsilon_j)$-LDP.*

The popularity of LDP has led to the development of several LDP protocols [3, 13, 31, 32]. New systems and applications often use these LDP protocols as building blocks. In particular, we will use the *Optimized Unary Encoding (OUE)* protocol as a building block in this paper, since it was shown to provide higher accuracy than many other protocols [32]. As in other protocols, OUE consists of two main components: (i) user-side encoding and perturbation to satisfy LDP on users' devices, and (ii) server-side estimation after collecting perturbed data from the user population.

User-Side Perturbation in OUE: OUE assumes that the user's data is encoded as a unary bitvector (vector of bits), i.e., only one position in the user's vector contains a 1 bit and all remaining positions contain 0 bits. Let B_i denote user u_i's unary bitvector. The perturbation algorithm Ψ of OUE takes B_i as input and outputs perturbed bitvector B'_i as:

$$\Pr\Big[B'_i[j] = \Psi(B_i[j]) = 1\Big] = \begin{cases} \frac{1}{2} & \text{if } B_i[j] = 1 \\ \frac{1}{e^{\varepsilon}+1} & \text{if } B_i[j] = 0 \end{cases} \tag{2}$$

In other words, each position $j \in [1, |B_i|]$ is considered independently from others, and the bit that exists in $B_i[j]$ is either kept or flipped according to the above probabilities. After this perturbation is complete, u_i sends the perturbed bitvector B'_i to the server.

Server-Side Estimation in OUE: The server receives perturbed bitvectors from many users, collectively denoted by $\{B'_1, B'_2, ..., B'_n\}$. Then, the server computes the reported counts of each position $j \in [1, |B_i|]$, which we denote by $\widehat{C}(j)$:

$$\widehat{C}(j) = \sum_{i=1}^{n} B'_i[j] \tag{3}$$

Finally, the server computes the estimate for position j, i.e., how many users have 1 bit in the j'th position of their original bitvectors. This estimate, denoted by $\bar{C}(j)$, is computed as:

$$\bar{C}(j) = \frac{2 \cdot \big((e^{\varepsilon}+1) \cdot \widehat{C}(j) - |\mathcal{U}|\big)}{e^{\varepsilon}-1} \tag{4}$$

2.3 Spatial Decompositions and Quadtrees

A spatial decomposition decomposes a geometric space into smaller subspaces. Tree-based (hierarchical) decompositions are quite common, in which data points are divided among the leaf nodes [8, 19, 22, 23, 39]. Tree-based decompositions are usually computed down to a level in which either the leaves contain a small number of points or each leaf covers a small enough area.

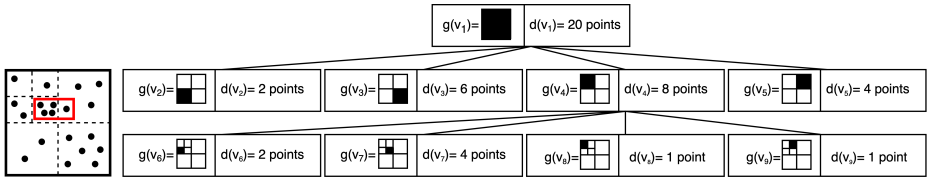

Fig. 1: Sample users with their locations represented as points (on the left) and the corresponding quadtree (on the right).

Among tree-based spatial decomposition techniques, quadtrees are a popular and well-known method [8, 23, 28]. A quadtree recursively divides the geographical space into four equi-sized quadrants (top left, top right, bottom left, bottom right) at each step. A sample quadtree is illustrated in Figure 1. The root corresponds to Ω as a whole. In the next level of the tree, Ω is divided into four quadrants, each of which corresponds to a child of the root node. The division of a node into four quadrants will keep occurring recursively, until: (i) the maximum height limit h^* is reached, or (ii) the current node contains fewer number of points than a pre-defined threshold θ.

We establish some notation and terminology with the help of Figure 1. Let Q denote a quadtree and let $\mathcal{V} = \{v_1, v_2, v_3, ...\}$ denote the set of nodes (vertices) in Q. The depth of a node v is denoted by $h(v)$, e.g., in Figure 1, $h(v_1) = 1$, $h(v_2) = h(v_3) = 2$, $h(v_7) = 3$. Nodes with the same $h(v)$ are found at the same depth of Q. Each node corresponds to a region of the geographical space as shown in Figure 1, e.g., v_1 corresponds to Ω, v_2 corresponds to bottom left quadrant of Ω, and so forth. We denote the geographical space corresponding to vertex v by $g(v)$. Furthermore, the number of data points located in the corresponding region of v is called the *density* of v and it is denoted by $d(v)$. In Figure 1, $d(v_1) = 20$, $d(v_2) = 2$, $d(v_3) = 6$, etc. By definition, the root node v_1 has density $d(v_1) = n$ since all users are assumed to be located within Ω.

We denote the children of a node v by $c(v)$, e.g., according to Figure 1, $c(v_4) = \{v_6, v_7, v_8, v_9\}$. If a node has children, it will always have exactly 4 children. If a node does not have children, then it is called a *leaf* node. The parent of a node v is denoted by $p(v)$, e.g., $p(v_6) = p(v_8) = v_4$.

In general, let v_i denote a node with set of children $c(v_i) = \{v_{i1}, v_{i2}, v_{i3}, v_{i4}\}$. By definition of Q and from the example in Figure 1, it can be verified that:

$$g(v_i) = g(v_{i1}) \cup g(v_{i2}) \cup g(v_{i3}) \cup g(v_{i4}) \tag{5}$$

and the pairwise intersection $g(v_{ij}) \cap g(v_{ik})$ is empty for all $j \neq k$. Furthermore, the sum of densities of all children equals the density of the parent:

$$d(v_i) = d(v_{i1}) + d(v_{i2}) + d(v_{i3}) + d(v_{i4}) \tag{6}$$

One of the key applications of quadtrees is in answering spatial count (density) queries [8, 23, 39]. Consider a query q. Initially, the answer of q is set to 0, i.e., $ans_q = 0$. Then, starting from the root, the quadtree is traversed in-top down fashion. For each node v that is visited:

1. If $g(v)$ is disjoint from q, then v is ignored.
2. If $g(v)$ is fully contained in q, then ans_q is increased by $d(v)$.
3. If $g(v)$ partially intersects q and v is not a leaf node, then every child of v with a region not disjoint from q must be visited.
4. If $g(v)$ partially intersects q and v is a leaf node, then ans_q is incremented by:

$$ans_q = ans_q + d(v) \times \frac{||g(v) \cap q||}{||g(v)||}$$

Here, $|| \cdot ||$ denotes the size of a geographical region.

We exemplify this process with the help of Figure 1. Let q be a query denoted using the red rectangle. Starting from v_1, we first find that $g(v)$ intersects with v_1, therefore v_1's children must be visited. Since v_2 and v_3 have no intersection with q, they are ignored. v_5 has intersection with q and it is a leaf node, therefore ans_q is incremented by $d(v_5) \times \frac{||g(v_5) \cap q||}{||g(v_5)||}$. v_4 has intersection with q but it is not a leaf node, therefore v_4's children must be visited next. Among v_4's children, v_6, v_8 and v_9 are ignored. Since v_7 intersects with q, ans_q is incremented by $d(v_7) \times \frac{||g(v_7) \cap q||}{||g(v_7)||}$.

2.4 Problem Formulation

Let $\mathcal{U}$ be the collection of users where l_i denotes the location of user u_i. Let Ω, θ and h^* parameters be known by the server. The server would like to construct a quadtree according to these parameters. If the server was able to access the real location l_i of each user, then a noise-free quadtree Q can be built. However, each user's location must be protected by ε-LDP, which causes users' responses to the server to be perturbed (noisy) to achieve privacy protection. Given the noisy responses, the server can build a *noisy* quadtree, denoted by Q'. For Q' to be useful in practice, it is desired that Q' has high similarity to Q. In other words, it is desired to build a quadtree Q' while satisfying ε-LDP, such that Q' has high similarity to the hypothetical, noise-free Q that would have been built if privacy protection did not exist.

In short, the problem that we study in this paper is to design and develop a solution in which: (i) the privacy of each user's location l_i is protected via ε-LDP, and (ii) the server is able to build a quadtree Q' such that Q' has high similarity to quadtree Q which would have been built if privacy protection had not been applied.

3 Building Quadtrees under LDP

In this section, we describe two solutions for the problem formulated in Section 2.4. Our first solution is inspired by a differentially private quadtree building algorithm from the centralized DP literature, which we modify in order to adapt to LDP. We explain the centralized algorithm and the key modifications we need to perform to adapt it to LDP in Section 3.1. Then, our baseline solution is given in 3.2. However, we observe that although the iterative noise addition approach used in the baseline solution satisfies LDP, it is not a desirable solution in practice since it causes large noise accumulation and therefore low utility. This motivates the proposal of our new approach in Section 3.3, which yields higher utility than the baseline solution.

3.1 Quadtrees under Centralized DP

Consider the quadtree building algorithm presented in [39], which satisfies centralized DP. The algorithm starts by initializing quadtree Q with a root node v_1 such that $g(v_1) = \Omega$ and marks v_1 as unvisited. Then, the algorithm proceeds iteratively, and at each iteration, it checks if there is any unvisited node $v \in Q$. If there is an unvisited node, the algorithm computes the noisy density of v. For this purpose, the Laplace mechanism of DP can be used [11]. The noisy density is determined as: $\hat{d}(v) = d(v) + Lap(\lambda)$, where $Lap(\lambda)$ is a Laplace random variable with mean 0 and scale λ. Afterwards, node v is checked regarding whether it satisfies the splitting conditions, i.e., its noisy density is higher than split threshold θ and depth of resulting children will not exceed max depth limit h^*. Indeed, if $\hat{d}(v) \geq \theta$ and $h(v) + 1 \leq h^*$, then v is split into four children, the children are inserted to Q, and they are marked as *unvisited*. Otherwise, v is not split and becomes a leaf node. At this point, v has been processed; it is marked as *visited* and the algorithm proceeds to the next iteration (next unvisited node in Q). When all nodes in Q eventually become visited, the algorithm terminates and returns Q.

It can be shown that this algorithm satisfies ε-DP in the centralized setting when $\lambda \geq h^*/\varepsilon$. To verify this, let a new user with an arbitrary location be inserted (removal of an arbitrary user is very similar). This insertion will impact the noisy densities of nodes residing in exactly one root-to-leaf path in Q; since by properties of quadtrees, nodes with the same depth do not have any intersection in their geographic coverages $g(\cdot)$. Then, since the number of nodes residing in one root-to-leaf path in Q is at most h^*, adding Laplace noise calibrated to h^*/ε is sufficient to achieve ε-DP.

Our baseline solution for building quadtrees with LDP adapts the aforementioned algorithm from the centralized DP literature to LDP. Two key modifications are needed in this adaptation. First, the above algorithm visits nodes one-by-one in arbitrary order, as long as they are unvisited. In contrast, our baseline solution visits nodes in breadth-first order, i.e., depth = 1 in the first iteration, depth = 2 in the second iteration, and so forth. Notice that this has no adverse impact on utility or privacy (the aforementioned algorithm can be trivially modified to act in breadth-first order), but it offers us an important convenience in LDP: It enables us to estimate node densities with LDP iteratively such that in the first iteration all nodes with depth = 1 are estimated, in the second iteration all nodes with depth = 2 are estimated, and so forth. The second modification we perform is that instead of adding Laplace noise, we execute an LDP protocol (OUE) to perform node density estimation. While Laplace noise addition is a de facto mechanism to achieve centralized DP, it is not directly applicable to LDP, therefore the usage of an LDP protocol becomes necessary.

3.2 Baseline Solution for Building Quadtrees under LDP

Our baseline solution for building a quadtree under LDP is shown in Algorithm 1. Given the total privacy budget ε, on line 1, the algorithm computes $\hat{\varepsilon} = \varepsilon/(h^* - 1)$. Here, considering that the algorithm will traverse the quadtree depth by depth and h^* is the max depth parameter, $\hat{\varepsilon}$ is the privacy budget that the algorithm will spend at each depth. Since the density of the root node is always equal to $|\mathcal{U}|$ and the server can trivially know the size of the user population, no privacy budget is spent at depth

Algorithm 1: Baseline solution

Input : $\mathcal{U}$, Ω, θ, h^*, ε
Output : Quadtree Q

1 $\hat{\varepsilon} \leftarrow \varepsilon/(h^* - 1)$
2 Initalize quadtree Q with root node v_1
3 Set $g(v_1) = \Omega$
4 $i \leftarrow 1$ `// current depth`
5 **while** $i \leq h^*$ **do**
`// find densities at current depth`
6 $nodes \leftarrow$ list of nodes in Q with depth = i
7 **if** $|nodes| == 1$ **then**
8 Set $d(nodes[1]) = |\mathcal{U}|$ `// i = 1, root node only`
9 **else**
10 $estimates \leftarrow$ GET_ESTIMATES$(nodes, \mathcal{U}, \hat{\varepsilon})$
11 **for** $j = 1$ *to* $|nodes|$ **do**
12 Set $d(nodes[j]) = estimates[j]$
`// for each node at current depth, determine if it should split or not`
13 **for** $j = 1$ *to* $|nodes|$ **do**
14 **if** $d(nodes[j]) \geq \theta$ *and* $i + 1 \leq h^*$ **then**
15 Split $nodes[j]$ into its four children and add the children to Q
16 **else**
17 Do not split $nodes[j]$
18 $i \leftarrow i + 1$
19 **return** Q

= 1 (only the root node exists at depth = 1). Thus, dividing ε into $h^* - 1$ pieces is sufficient. On lines 2-3, the algorithm initializes the root node. Then, the main loop of the algorithm (lines 5-18) iterates depth-by-depth, and at each iteration, it estimates the densities of nodes at the current depth. The case where depth = 1 (only the root node exists) is handled between lines 7-8. At every other depth value, nodes' densities at that depth are computed with the help of the GET_ESTIMATES function (lines 10-12), which is explained below. GET_ESTIMATES satisfies $\hat{\varepsilon}$-LDP, and note that Algorithm 1 invokes it at most $h^* - 1$ times; thus, the overall algorithm satisfies ε-LDP by sequential composition. Finally, lines 13-17 are devoted to checking which nodes should split and which ones should not split. If and only if a node satisfies the splitting conditions, i.e., its noisy density is $\geq \theta$ and the height of its children will not exceed h^*, then the node will split.

The GET_ESTIMATES function, which is used by our baseline solution as well as our proposed solution, is described in Algorithm 2. Its inputs are the list of nodes $\mathcal{N}$, list of users $\mathcal{U}$, and privacy budget $\hat{\varepsilon}$. It returns a list called $estimates$, such that $estimates[j]$ is the estimated noisy density of node $\mathcal{N}[j]$. The execution of the GET-_ESTIMATES function can be broken down into three steps. First, the server sends $\mathcal{N}$ to each user. Second, on the user side, each user u_i constructs a bitvector B_i, where $|B_i| = |\mathcal{N}|$. For each position j, $B_i[j]$ is determined according to whether u_i's real

Algorithm 2: Get_Estimates function

Input : $\mathcal{N}, \mathcal{U}, \hat{\varepsilon}$
Output: $estimates$

`// Server-side`
1 Server sends $\mathcal{N}$ to each user in $\mathcal{U}$
`// User-side`
2 **foreach** $u_i \in \mathcal{U}$ **do**
3 u_i constructs his/her bitvector B_i with length $|\mathcal{N}|$ such that:

$$\forall j \in [1, |\mathcal{N}|]: \quad B_i[j] = \begin{cases} 1 & \text{if } l_i \in g(\mathcal{N}[j]) \\ 0 & \text{otherwise} \end{cases}$$

4 u_i perturbs B_i to satisfy $\hat{\varepsilon}$-LDP according to Equation 2
5 u_i sends resulting perturbed bitvector B'_i to server
`// Server-side`
6 $estimates \leftarrow [\,]$ `// initialize empty`
7 **for** $j = 1$ *to* $|\mathcal{N}|$ **do**
8 Server computes reported count of position j as: $\widehat{C}(j) = \sum_{i=1}^{|\mathcal{U}|} B'_i[j]$
9 Server computes estimate $\bar{C}(j)$ as: $\bar{C}(j) = \frac{2 \cdot \left((e^{\hat{\varepsilon}}+1) \cdot \widehat{C}(j) - |\mathcal{U}|\right)}{e^{\hat{\varepsilon}}-1}$
10 Server appends $\bar{C}(j)$ to $estimates$
11 **return** $estimates$

location l_i falls within the geographic boundaries of node $\mathcal{N}[j]$. That is, if l_i falls within $g(\mathcal{N}[j])$ then the j'th position of B_i is set to 1, otherwise it is 0. After constructing the B_i, it must be perturbed probabilistically in order to satisfy $\hat{\varepsilon}$-LDP. To do so, the user-side perturbation process of the OUE protocol is executed, as shown in Equation 2. The output of this process is the perturbed bitvector B'_i, which is sent to the server. Finally, the third step begins after the server receives perturbed bitvectors from all users. The server initializes $estimates$ as an empty list. Afterwards, for j between 1 and $|\mathcal{N}|$, the server first computes $\widehat{C}(j)$ and then uses $\widehat{C}(j)$ to compute $\bar{C}(j)$ according to the equations on lines 8-9. $\bar{C}(j)$ is appended to $estimates$ in each iteration, and eventually, $estimates$ is returned by Algorithm 2.

3.3 Proposed Solution for Building Quadtrees under LDP

The main weakness of the baseline solution is that it divides the total ε privacy budget into $h^* - 1$ pieces, each to be used in one depth of estimation. Since the privacy budget used per estimation is smaller, higher noise gets added to satisfy privacy. This causes resulting node density estimations to contain large amounts of noise. Excessively noisy densities are also used in the decision to split or not split a node, further causing structural inaccuracies in the resulting quadtree, since a node that should not be split according to its real density ends up being split due its noisy density (or vice versa). Consequently, although ε-LDP is achieved, the quadtrees built using our baseline solution can have low similarity in terms of structure and node densities compared to a noise-free quadtree.

Algorithm 3: Proposed solution

Input : $\mathcal{U}$, Ω, θ, h^*, ε
Output: Quadtree Q

```
// Step 1: Construct initial tree
```
Initalize quadtree Q with root node v_1
Set $g(v_1) = \Omega$
for $i = 1$ *to* $h^* - 1$ **do**
    foreach *node* v *in* Q *with* $h(v) = i$ **do**
        Split v into its four children and add the children to Q
```
// Step 2: Density estimation for leaf nodes with LDP
```
$leaves \leftarrow$ list of nodes in Q with depth = h^*
$estimates \leftarrow$ GET_ESTIMATES$(leaves, \mathcal{U}, \varepsilon)$
for $j = 1$ *to* $|leaves|$ **do**
    Set $d(leaves[j]) = estimates[j]$
```
// Step 3: Bottom-up propagation of densities
```
for $i = h^* - 1$ *to* 1 **do**
    foreach *node* v *in* Q *with* $h(v) = i$ **do**
        Initialize $d(v) \leftarrow 0$
        for $v_{child} \in c(v)$ **do**
            $d(v) \leftarrow d(v) + d(v_{child})$
```
// Step 4: Top-down correction
```
for $i = 1$ *to* $h^* - 1$ **do**
    foreach *node* v *in* Q *with* $h(v) = i$ **do**
        if $d(v) < \theta$ **then**
            Remove all children $c(v)$ and all subtrees rooted at those children from Q
return Q

In order to address this problem, the key insight of our proposed solution is that instead of dividing ε into $h^* - 1$ pieces, it uses the whole ε in a single step. This single-step estimation is performed at the leaves of the quadtree (depth = h^*) and contains low noise, therefore the leaves contain high quality density estimates. Then, leaves' densities are propagated upwards (in bottom-up fashion) towards the root, to populate the densities of remaining nodes. Finally, in top-down fashion, corrections and refinements are made in the structure of the quadtree.

The pseudocode of our proposed solution is shown in Algorithm 3. Its inputs and outputs are same as the baseline solution. It can be observed from Algorithm 3 that the proposed solution consists of four main steps, which are explained below.

Step 1: Construct initial, balanced quadtree with height = h^*. We construct an initial quadtree Q that is fully-grown until height h^*. That is, we let each node in the quadtree split into its four children until h^* is reached. The resulting Q is fully balanced. Note that the θ threshold is not taken into account in constructing this initial Q.

Step 2: Density estimation for leaf nodes with LDP. In the second step, we retrieve all leaf nodes in Q and obtain density estimates for each of them using the full privacy budget ε. This is achieved by a single invocation of the GET_ESTIMATES function

with all leaves and full privacy budget ε. After this step is complete, for each leaf node $v \in Q$, its density $d(v)$ is determined.

Step 3: Bottom-up propagation of densities. While densities of the leaves have been determined in step 2, densities of all remaining nodes (non-leaf nodes) are unknown. In this step, densities of non-leaf nodes are determined in bottom-up fashion. First, all nodes with depth $h^* - 1$ are handled, then, all nodes with depth $h^* - 2$ are handled, ... until the root node. For each non-leaf node, recall from Equation 6 that its density is equal to the sum of the densities of its four children.

Step 4: Top-down correction. The initial quadtree Q constructed in step 1 is fully-balanced rather than taking into account the density threshold θ. As a result, it is possible that a node v which should not have children (because it fails the $d(v) \geq \theta$ condition) actually has children in Q. The goal of step 4 is to iterate through Q in top-down fashion and fix such situations. To do so, the algorithm starts at depth = 1 and moves down iteratively (depth = 2, depth = 3, ..., depth = $h^* - 1$). For each node at the current depth, if node v does not satisfy the $d(v) \geq \theta$ condition, its children along with the subtrees rooted at those children (i.e., if the child has any descendants) are removed from Q.

4 Experimental Evaluation

4.1 Experiment Setup

We implemented the baseline and proposed solutions in Python. We experimentally compare them under varying ε, θ, and h^* parameters using multiple evaluation metrics. We use four real-world location datasets in our experiments: Kaggle, Brightkite, Gowalla, and Foursquare.

Kaggle: The Kaggle dataset contains trips of 442 taxis driving in the city of Porto. The dataset was originally made public for the taxi service prediction competition in ECML-PKDD; we downloaded it from Kaggle [12]. While the dataset contains full taxi trips (multiple location reading per trip), we pre-processed it by keeping only the starting location of each trip, and treated the trip starting locations as the current locations of users in the user population. At the end of this processing, we ended up with 1,048,575 users and their latitude, longitude locations.

Brightkite: The Brightkite dataset contains users' location check-ins from a social network service provider called Brightkite [6]. From the full dataset, we extracted check-ins made in the United States, between longitudes -124.26 and -71.87 and latitudes 25.45 and 47.44.

Gowalla was also a location-based social network site where users contributed their data by sharing their locations [6]. Similar to Brightkite, we extracted check-ins made in the United States using the same latitude and longitude boundaries.

Foursquare: The Foursquare dataset contains location check-ins of users in Tokyo, between the time period from 12 April 2012 to 16 February 2013 [35]. We used this dataset without any pre-processing. It contains a total of 573,703 location check-ins. The minimum latitude is 35.51, maximum latitude is 35.87, minimum longitude is 139.47, and maximum longitude is 139.91.

4.2 Utility Metrics

Let Q denote the gold standard, noise-free quadtree that would be built without privacy protection. Let Q' denote the noisy quadtree built under ε-LDP. We use three utility metrics to measure the difference between Q and Q'. Higher the values of these metrics, higher the difference between Q and Q', and therefore higher the amount of utility loss.

Average Query Error (AQE): We generate $N = 100$ random queries and compute their answers on the actual quadtree Q and the noisy quadtree Q', denoted by ans_q and ans'_q respectively. Then, AQE measures the average error between ans_q and ans'_q across all queries as follows:

$$AQE = \frac{\sum_{i=1}^{N} \frac{|ans_{q_i} - ans'_{q_i}|}{max\{ans_{q_i}, b\}}}{N}$$

where q_i denotes the i'th query and b denotes a sanity bound that mitigates the effect of queries with extremely high selectivities (extremely low answers) [4, 16, 39]. We set the value of b as: $b = 2\% \times |\mathcal{U}|$.

Tree Edit Distance (TED): TED measures the structural difference between Q and Q'. Consider that we want to measure TED between two subtrees rooted at nodes v and v', such that $g(v) = g(v')$. $TED(v, v')$ is defined recursively as follows:

$$TED(v, v') = \begin{cases} 0 & \text{if } |c(v)| = 0 \text{ and } |c(v')| = 0 \\ num\_desc(v) & \text{if } |c(v)| > 0 \text{ and } |c(v')| = 0 \\ num\_desc(v') & \text{if } |c(v)| = 0 \text{ and } |c(v')| > 0 \\ \sum\limits_{\substack{x \in c(v),\ x' \in c(v') \\ s.t.\ g(x) = g(x')}} TED(x, x') & \text{if } |c(v)| > 0 \text{ and } |c(v')| > 0 \end{cases}$$

where $num\_desc(v)$ denotes the number of descendent nodes that v has (excluding itself). The rationale is as follows: If both v and v' do not have any children, then they have zero TED (no structural difference). If one of them has children but the other does not, then all descendents must be counted as part of TED. If both v and v' has children, then their TED is computed recursively on pairs (x, x') where x and x' have matching geographical regions, i.e., x' in Q' is the noisy counterpart of x from Q.

Once TED between v and v' is defined as above, it can be used to measure TED between two quadtrees Q and Q' as: $TED(Q, Q') = TED(v_{root}, v'_{root})$ where v_{root} is the root node of Q and v'_{root} is the root node of Q'.

Node Density Difference (NDD): NDD measures the difference between Q and Q' by computing the differences between corresponding nodes' densities.

$$NDD(Q, Q') = \sum_{v \in Q} \varphi(v)$$

where:

$$\varphi(v) = \begin{cases} |d(v) - d(v')| & \text{if } \exists\ v' \in Q' \text{ s.t. } g(v) = g(v') \\ d(v) & \text{otherwise} \end{cases}$$

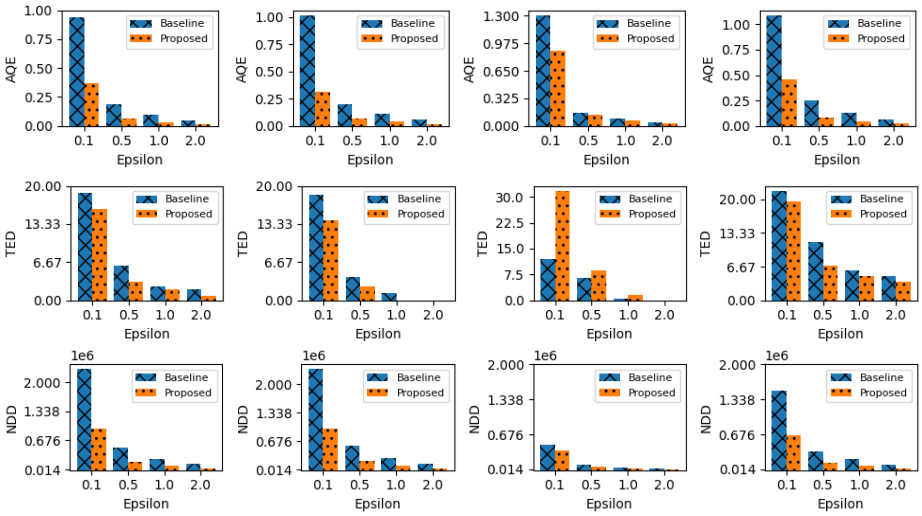

Fig. 2: Results with AQE, TED and NDD metrics (one metric each row) on Brightkite, Gowalla, Kaggle and Foursquare datasets (left to right).

In other words, NDD iterates over each node $v \in Q$ and checks if its counterpart exists in Q', i.e., $v' \in Q'$ with $g(v) = g(v')$. If it exists, then the difference between their densities is computed and added to NDD. However, due to structural differences between Q and Q', it is also possible that such a v' does not exist. In that case, $d(v')$ is assumed to be 0 and therefore $d(v)$ is added directly to NDD.

4.3 Results and Discussion

We first keep the h^* and θ parameters constant and vary the ε parameter to observe its impact. The results are reported in Figure 2 ($h^* = 4$, $\theta = 10000$). Four popular (conventional) ε values are used: ε = 0.1, 0.5, 1, 2. Each experiment is repeated 10 times and their average results are reported.

Across all datasets and metrics, we observe that: (i) errors decrease as ε is increased, and (ii) the proposed solution yields lower errors than the baseline solution. The prior is an intuitive result because as ε increases, the noise caused by LDP decreases, therefore it becomes possible to build more accurate quadtrees. With regards to the latter, especially on Brightkite and Gowalla datasets, the proposed solution makes remarkable improvement in terms of the AQE and NDD metrics when ε is low. The difference between the two solutions is relatively lower in terms of the TED metric. Similar observation holds for the Foursquare dataset – while the difference between the baseline solution and proposed solution is high in terms of AQE and NDD metrics, it is relatively less pronounced in terms of the TED metric. The only exception in which the proposed solution yields higher error than the baseline solution is the Kaggle dataset and the TED metric. The reason behind this observation is the significant skew in the spatial distribution of the Kaggle dataset. Users' locations in this dataset are heavily accumulated

Table 1: AQE, TED and NDD of different heights h^* with threshold $\theta = 10000$.

			AQE				TED				NDD (x10^4)			
			$\varepsilon = 0.1$	$\varepsilon = 0.5$	$\varepsilon = 1$	$\varepsilon = 2$	$\varepsilon = 0.1$	$\varepsilon = 0.5$	$\varepsilon = 1$	$\varepsilon = 2$	$\varepsilon = 0.1$	$\varepsilon = 0.5$	$\varepsilon = 1$	$\varepsilon = 2$
Brightkite	$h^* = 3$	Baseline	0.317	0.056	0.028	0.014	0.0	0.0	0.0	0.0	60.2	10.4	5.5	2.3
		Proposed	0.156	0.030	0.014	0.006	0.0	0.0	0.0	0.0	30.1	5.2	2.5	1.2
	$h^* = 4$	Baseline	0.941	0.189	0.095	0.046	18.8	6.0	2.4	2.0	229.5	51.7	26.0	13.8
		Proposed	0.369	0.067	0.032	0.015	16.0	3.2	2.0	0.8	94.1	18.5	9.7	4.0
	$h^* = 5$	Baseline	1.924	0.423	0.213	0.092	105.2	66.8	41.2	19.2	499.1	139.9	74.7	38.8
		Proposed	0.637	0.146	0.080	0.036	110.4	55.2	16.0	4.8	258.3	58.6	27.3	11.7
Gowalla	$h^* = 3$	Baseline	0.292	0.053	0.026	0.011	0.0	0.0	0.0	0.0	50.7	10.4	4.8	2.4
		Proposed	0.140	0.022	0.013	0.005	0.0	0.0	0.0	0.0	24.8	4.8	2.5	1.1
	$h^* = 4$	Baseline	1.015	0.201	0.108	0.057	18.4	4.0	1.2	0.0	234.2	56.5	28.4	14.0
		Proposed	0.313	0.069	0.039	0.016	14.0	2.4	0.0	0.0	95.9	20.9	9.5	4.3
	$h^* = 5$	Baseline	2.076	0.436	0.215	0.094	118.4	68.8	40.0	17.6	554.0	151.7	80.5	42.3
		Proposed	0.850	0.172	0.074	0.032	122.4	49.2	17.6	7.6	285.5	62.1	29.6	13.0
Kaggle	$h^* = 3$	Baseline	0.396	0.058	0.025	0.011	4.0	1.2	0.0	0.0	20.5	4.9	2.2	1.1
		Proposed	0.296	0.051	0.020	0.008	5.2	1.6	0.0	0.0	17.8	3.4	1.5	0.6
	$h^* = 4$	Baseline	1.302	0.155	0.085	0.039	12.0	6.4	0.4	0.0	47.0	9.4	5.1	2.5
		Proposed	0.885	0.137	0.065	0.026	31.6	8.8	1.6	0.0	37.2	7.4	3.2	1.4
	$h^* = 5$	Baseline	3.823	0.562	0.145	0.064	52.0	31.2	4.0	0.0	95.7	16.0	8.5	4.2
		Proposed	1.815	0.290	0.133	0.054	74.0	29.6	8.8	0.8	73.0	14.7	7.6	2.8
Foursquare	$h^* = 3$	Baseline	0.373	0.062	0.038	0.019	0.0	0.0	0.0	0.0	38.5	7.3	3.9	1.8
		Proposed	0.154	0.033	0.017	0.008	0.0	0.0	0.0	0.0	19.5	3.5	1.8	0.8
	$h^* = 4$	Baseline	1.085	0.249	0.128	0.060	21.6	11.6	6.0	4.8	149.5	36.0	20.1	9.8
		Proposed	0.462	0.084	0.045	0.024	19.6	6.8	4.8	3.6	65.8	14.4	8.0	3.8
	$h^* = 5$	Baseline	2.334	0.455	0.207	0.098	94.8	58.4	44.0	22.4	275.3	68.7	37.8	19.5
		Proposed	0.883	0.195	0.087	0.046	109.2	56.0	25.6	17.2	146.2	38.4	17.7	8.9

within a small range of latitude and longitude coordinates, and much fewer location readings exist outside this range. Consequently, in the noise-free quadtree, there exist few branches with high depth whereas remaining branches are shallow. However, the proposed solution which constructs a fully balanced quadtree first and then performs top-down structural corrections in its final step has a higher tendency to end up with a more breadth-balanced quadtree than the noise-free version, thus causing high TED.

Next, we fix $\theta = 10000$ and vary the h^* and ε parameters. The results are shown in Table 1. We again observe that the proposed solution performs better than the baseline solution. For both solutions, it seems that as we increase h^*, errors tend to increase. For the baseline solution, this shows the ineffectiveness of splitting the original ε budget into $h^* - 1$ parts. Another interesting observation is that TED results are good when h^* is low, such as $h^* = 3$. This shows that for the first few levels of the quadtree (which are close to the root node), there is relatively small structural error. For example, when $h^* = 3$, TEDs are usually 0 meaning that our LDP quadtrees have identical structure to noise-free quadtrees. However, as we increase h^* to 4 and 5, there is an increasing amount of structural inequality as demonstrated by increasing TEDs. Overall, this shows that LDP quadtrees are more likely to have structural errors towards their leaf nodes whereas their structure close to the root node remains more accurate.

Table 2: AQE with privacy budget $\varepsilon = 1$, varying h^* and θ.

		Baseline				Proposed			
		$\theta = 1000$	$\theta = 5000$	$\theta = 10000$	$\theta = 20000$	$\theta = 1000$	$\theta = 5000$	$\theta = 10000$	$\theta = 20000$
Brightkite	$h^* = 3$	0.027	0.029	0.032	0.022	0.015	0.015	0.012	0.012
	$h^* = 4$	0.100	0.099	0.103	0.084	0.036	0.036	0.035	0.039
	$h^* = 5$	0.278	0.229	0.195	0.182	0.078	0.072	0.085	0.082
Gowalla	$h^* = 3$	0.025	0.025	0.026	0.030	0.012	0.013	0.014	0.013
	$h^* = 4$	0.097	0.106	0.100	0.108	0.034	0.028	0.036	0.041
	$h^* = 5$	0.257	0.221	0.214	0.175	0.084	0.067	0.076	0.085
Kaggle	$h^* = 3$	0.046	0.030	0.023	0.024	0.031	0.024	0.021	0.020
	$h^* = 4$	0.170	0.085	0.073	0.075	0.089	0.065	0.073	0.062
	$h^* = 5$	0.396	0.268	0.150	0.161	0.173	0.156	0.177	0.160
Foursquare	$h^* = 3$	0.035	0.035	0.035	0.028	0.016	0.018	0.018	0.017
	$h^* = 4$	0.134	0.130	0.134	0.098	0.046	0.044	0.042	0.043
	$h^* = 5$	0.318	0.294	0.210	0.135	0.096	0.095	0.083	0.088

Finally, in Table 2, we fix $\varepsilon = 1$ and vary the h^* and θ parameters. For low h^* such as $h^* = 3$, different values of θ do not seem to cause substantial changes in AQE on Brightkite, Gowalla and Foursquare datasets. However, for $h^* = 4$ and $h^* = 5$, changes to θ yield higher differences in terms of AQE. Usually, $\theta = 5000$ or 10000 seem to be the ideal choice when $h^* = 4$ and $h^* = 5$ for the proposed solution. In contrast, higher θ such as 10000 or 20000 seem to be better for the baseline solution. This is because increasing θ decreases the risk of creating erroneous nodes caused by LDP noise. When θ is higher, it is less likely that LDP noise causes node densities to erroneously increase to higher than θ and cause an erroneous split. Similarly, higher θ implies shorter quadtrees for the baseline solution, which eliminates the creation of deeper nodes that have higher risk of being dominated by LDP noise.

5 Related Work

LDP for spatial data. In recent years, LDP has emerged as an accepted privacy standard and it has also been successfully deployed in the industry [1, 7, 10, 13, 18, 37]. With the popularity of LDP, there is rising interest in applying LDP to spatial data, considering the ubiquity of location-based services and sensitive nature of users' locations. Wang et al. [30] developed L-SRR to privately collect users' locations while they remain useful for LBS applications such as traffic density estimation and k-nearest neighbors. Hong et al. [20] proposed a perturbation mechanism designed to reduce the error of each perturbed location under LDP. Yang et al. [36] studied the problem of collecting individual trajectories under LDP. Cunningham et al. [9] proposed a technique which utilizes hierarchical n-grams for real-world trajectory sharing with LDP. Kim et al. [21] and Navidan et al. [25] applied LDP to indoor positioning data. Finally, Chen et al. [5] proposed a variant of LDP, called personalized LDP, for spatial data aggregation.

Spatial decompositions under DP. The above works apply LDP to spatial data, but they do not have the goal of building spatial decompositions. Although spatial decompositions have not been studied under LDP, they have been studied under centralized DP. Cormode et al. [8] proposed algorithms to build tree-based spatial decompositions such as quadtrees and kd-trees under DP. Later, the PrivTree algorithm developed by Zhang et al. [39] enabled constructing a DP spatial decomposition without requiring a pre-defined recursion limit (height). A popular alternative to tree-based decompositions is grid-based decompositions such as uniform and adaptive grids, as introduced by Qardaji et al. [27] and later used in other works [16, 17]. Hierarchical reference systems proposed by He et al. [19] employ hierarchically organized grids with different granularities, ordered from coarse to fine-grained.

More recently, Niknami et al. [26] proposed personalized DP and personalized noise addition for indexing geometric objects in spatial databases. Quadtrees and kd-trees are used as spatial indices. In the work of Li et al. [23], a data-dependent adaptive density grid decomposition is used at the first layer, and then a quadtree decomposition is adopted for further splitting. Yan et al. [34] proposed an unbalanced quadtree partitioning algorithm for improving query accuracy in publishing spatial data with DP. Similarly, Liu et al. [24] also proposed a quadtree algorithm for improving query accuracy under DP, by balancing noise error and uniformity error. A new decomposition structure called Homogeneous Tree Framework (HTF) was proposed by Shaham et al. [29], which shares similarities to kd-trees. All of these works operate under the assumptions of centralized DP rather than LDP, hence they are not comparable to our work.

6 Conclusion

In this paper we studied the problem of building quadtrees, a popular spatial decomposition method, while each user's location remains protected by LDP. We proposed two solutions to this problem: a baseline solution which adapts an iterative strategy inspired from the centralized DP literature, and a newly proposed solution which relies on a single LDP data collection step. We compared the baseline and proposed solutions using three metrics (AQE, TED, NDD) and four real-world spatial datasets. Results demonstrated the superiority of the proposed solution compared to the baseline solution across many settings and quadtree parameters. In future work, we plan to extend our solutions to other tree-based spatial decomposition methods such as k-dimensional trees and octrees.

Acknowledgements We gratefully acknowledge the support by The Scientific and Technological Research Council of Türkiye (TUBITAK) under project number 121E303.

References

1. Apple – learning with privacy at scale. `https://machinelearning.apple.com/docs/learning-with-privacy-at-scale/appledifferentialprivacysystem.pdf` (2020)
2. Bagdasaryan, E., Kairouz, P., Mellem, S., Gascón, A., Bonawitz, K., Estrin, D., Gruteser, M.: Towards sparse federated analytics: Location heatmaps under distributed differential privacy with secure aggregation. Proceedings on Privacy Enhancing Technologies **4**, 162–182 (2022)

3. Bassily, R., Smith, A.: Local, private, efficient protocols for succinct histograms. In: Proceedings of the 47th Annual ACM Symposium on Theory of Computing. pp. 127–135. ACM (2015)
4. Chen, R., Acs, G., Castelluccia, C.: Differentially private sequential data publication via variable-length n-grams. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security. pp. 638–649 (2012)
5. Chen, R., Li, H., Qin, A., Kasiviswanathan, S.P., Jin, H.: Private spatial data aggregation in the local setting. In: 2016 IEEE 32nd International Conference on Data Engineering (ICDE). pp. 289–300. IEEE (2016)
6. Cho, E., Myers, S.A., Leskovec, J.: Friendship and mobility: User movement in location-based social networks. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 1082–1090. Association for Computing Machinery, New York, NY, USA (2011)
7. Cormode, G., Jha, S., Kulkarni, T., Li, N., Srivastava, D., Wang, T.: Privacy at scale: Local differential privacy in practice. In: Proceedings of the 2018 International Conference on Management of Data. pp. 1655–1658. ACM (2018)
8. Cormode, G., Procopiuc, C., Srivastava, D., Shen, E., Yu, T.: Differentially private spatial decompositions. In: 28th IEEE International Conference on Data Engineering. pp. 20–31. IEEE (2012)
9. Cunningham, T., Cormode, G., Ferhatosmanoglu, H., Srivastava, D.: Real-world trajectory sharing with local differential privacy. Proceedings of the VLDB Endowment **14**(11), 2283–2295 (2021)
10. Ding, B., Kulkarni, J., Yekhanin, S.: Collecting telemetry data privately. In: Advances in Neural Information Processing Systems. pp. 3571–3580 (2017)
11. Dwork, C., Roth, A., et al.: The algorithmic foundations of differential privacy. Foundations and Trends® in Theoretical Computer Science **9**(3–4), 211–407 (2014)
12. ECML/PKDD: Taxi trajectory prediction dataset, `https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i/data`
13. Erlingsson, Ú., Pihur, V., Korolova, A.: Rappor: Randomized aggregatable privacy-preserving ordinal response. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. pp. 1054–1067. ACM (2014)
14. Gursoy, M.E., Liu, L., Chow, K.H., Truex, S., Wei, W.: An adversarial approach to protocol analysis and selection in local differential privacy. IEEE Transactions on Information Forensics and Security **17**, 1785–1799 (2022)
15. Gursoy, M.E., Rajasekar, V., Liu, L.: Utility-optimized synthesis of differentially private location traces. In: IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA). pp. 30–39. IEEE (2020)
16. Gursoy, M.E., Liu, L., Truex, S., Yu, L.: Differentially private and utility preserving publication of trajectory data. IEEE Transactions on Mobile Computing **18**(10), 2315–2329 (2018)
17. Gursoy, M.E., Liu, L., Truex, S., Yu, L., Wei, W.: Utility-aware synthesis of differentially private and attack-resilient location traces. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 196–211 (2018)
18. Gursoy, M.E., Tamersoy, A., Truex, S., Wei, W., Liu, L.: Secure and utility-aware data collection with condensed local differential privacy. IEEE Transactions on Dependable and Secure Computing (2019)
19. He, X., Cormode, G., Machanavajjhala, A., Procopiuc, C.M., Srivastava, D.: Dpt: differentially private trajectory synthesis using hierarchical reference systems. Proceedings of the VLDB Endowment **8**(11), 1154–1165 (2015)
20. Hong, D., Jung, W., Shim, K.: Collecting geospatial data with local differential privacy for personalized services. In: 2021 IEEE 37th International Conference on Data Engineering (ICDE). pp. 2237–2242 (2021)

21. Kim, J.W., Kim, D.H., Jang, B.: Application of local differential privacy to collection of indoor positioning data. IEEE Access **6**, 4276–4286 (2018)
22. Kuo, Y.H., Chiu, C.C., Kifer, D., Hay, M., Machanavajjhala, A.: Differentially private hierarchical count-of-counts histograms. Proceedings of the VLDB Endowment **11**(11), 1509–1521 (2018)
23. Li, S., Geng, Y., Li, Y.: A differentially private hybrid decomposition algorithm based on quad-tree. Computers & Security **109**, 102384 (2021)
24. Liu, G., Tang, Z., Wan, B., Li, Y., Liu, Y.: Differential privacy location data release based on quadtree in mobile edge computing. Transactions on Emerging Telecommunications Technologies **33**(6), e3972 (2022)
25. Navidan, H., Moghtadaiee, V., Nazaran, N., Alishahi, M.: Hide me behind the noise: Local differential privacy for indoor location privacy. In: 2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). pp. 514–523. IEEE (2022)
26. Niknami, N., Abadi, M., Deldar, F.: A fully spatial personalized differentially private mechanism to provide non-uniform privacy guarantees for spatial databases. Information Systems **92**, 101526 (2020)
27. Qardaji, W., Yang, W., Li, N.: Differentially private grids for geospatial data. In: 2013 IEEE 29th International Conference on Data Engineering (ICDE). pp. 757–768 (2013)
28. Samet, H.: The quadtree and related hierarchical data structures. ACM Computing Surveys (CSUR) **16**(2), 187–260 (1984)
29. Shaham, S., Ghinita, G., Ahuja, R., Krumm, J., Shahabi, C.: Htf: Homogeneous tree framework for differentially-private release of large geospatial datasets with self-tuning structure height. ACM Transactions on Spatial Algorithms and Systems (2022)
30. Wang, H., Hong, H., Xiong, L., Qin, Z., Hong, Y.: L-srr: Local differential privacy for location-based services with staircase randomized response. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. p. 2809–2823. Association for Computing Machinery, New York, NY, USA (2022)
31. Wang, S., Huang, L., Wang, P., Nie, Y., Xu, H., Yang, W., Li, X.Y., Qiao, C.: Mutual information optimally local private discrete distribution estimation. arXiv preprint arXiv:1607.08025 (2016)
32. Wang, T., Blocki, J., Li, N., Jha, S.: Locally differentially private protocols for frequency estimation. In: Proc. of the 26th USENIX Security Symposium. pp. 729–745 (2017)
33. Wang, T., Li, N., Jha, S.: Locally differentially private frequent itemset mining. In: IEEE Symposium on Security and Privacy (SP). IEEE (2018)
34. Yan, Y., Gao, X., Mahmood, A., Feng, T., Xie, P.: Differential private spatial decomposition and location publishing based on unbalanced quadtree partition algorithm. IEEE Access **8**, 104775–104787 (2020)
35. Yang, D., Zhang, D., Zheng, V.W., Yu, Z.: Modeling user activity preference by leveraging user spatial temporal characteristics in lbsns. IEEE Transactions on Systems, Man, and Cybernetics: Systems **45**(1), 129–142 (2015)
36. Yang, J., Cheng, X., Su, S., Sun, H., Chen, C.: Collecting individual trajectories under local differential privacy. In: 2022 23rd IEEE International Conference on Mobile Data Management (MDM). pp. 99–108. IEEE (2022)
37. Yang, M., Lyu, L., Zhao, J., Zhu, T., Lam, K.Y.: Local differential privacy and its applications: A comprehensive survey. arXiv preprint arXiv:2008.03686 (2020)
38. Ye, Q., Hu, H., Meng, X., Zheng, H.: Privkv: Key-value data collection with local differential privacy. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 317–331. IEEE (2019)
39. Zhang, J., Xiao, X., Xie, X.: Privtree: A differentially private algorithm for hierarchical decompositions. In: Proceedings of the 2016 International Conference on Management of Data. pp. 155–170 (2016)