Hierarchical Spatial Decompositions Under Local Differential Privacy

by

Ece Alptekin

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Computer Science and Engineering

KOÇ ÜNİVERSİTESİ

October 3, 2024

Hierarchical Spatial Decompositions Under Local Differential Privacy

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Ece Alptekin

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Assist. Prof. Dr. Mehmet Emre Gürsoy (Advisor)

_____

Assoc. Prof. Dr. Alptekin Küpçü

_____

Prof. Dr. Yücel Saygın

Date: _____

to Mother and Father

ABSTRACT

Hierarchical Spatial Decompositions Under Local Differential Privacy
Ece Alptekin
Master of Science in Computer Science and Engineering
October 3, 2024

The popularity of smartphones, GPS-equipped devices, social networks, and connected vehicles continues to increase the volume of spatial data available for collection and analysis. Spatial decompositions assist in handling big spatial data, and they have been commonly used in the centralized differential privacy (DP) literature for range query answering, spatial indexing, count-of-counts histograms, data summarization, and visualization. However, their applications under the emerging local differential privacy (LDP) notion are relatively scarce.

In this thesis, we study the problem of building hierarchical spatial decompositions under LDP, focusing on two methods: quadtrees and kd-trees. We develop two solutions for quadtrees: a baseline solution that is inspired by the centralized DP literature, and a proposed solution that utilizes a single data collection step from users, propagates density estimates to remaining nodes, and performs structural corrections to the quadtree. Since kd-trees rely on node medians which are data-dependent, we observe that it is not feasible to build kd-trees using a single data collection step. We therefore propose an iterative solution that constructs kd-trees in top-down fashion by utilizing a novel algorithm for estimating node medians at each tree depth. We experimentally evaluate our quadtree and kd-tree algorithms using four real-world spatial datasets, multiple utility metrics, varying privacy budgets, and tree parameters. Results demonstrate that our algorithms enable the building of accurate spatial decompositions that provide high utility in practice. Notably, our quadtrees and kd-trees achieve substantially lower errors in answering spatial density queries (up to 10-fold improvement) when compared with a state-of-the-art method.

ÖZETÇE

Lokal Diferansiyel Mahremiyet Korumalı Hiyerarşik Konumsal Ayrışımlar
Ece Alptekin
Bilgisayar Mühendisliği, Yüksek Lisans
3 Ekim 2024

Akıllı telefonlar, GPS donanımlı cihazlar, sosyal ağlar ve bağlantılı araçların popülerliği, toplanması ve analiz edilmesi mümkün olan konumsal verinin hacmini arttırmaya devam etmektedir. Konumsal ayrışımlar, büyük konumsal verilerin işlenmesine yardımcı olur ve merkezi diferansiyel mahremiyet (DP) literatüründe aralık sorgusu yanıtlama, konumsal dizinleme, sayım-histogramlar, veri özetleme ve görselleştirme için sıklıkla kullanılmıştır. Ancak, yeni gelişen lokal diferansiyel mahremiyet (LDP) kavramı altındaki uygulamaları nispeten azdır.

Bu tezde, hiyerarşik konumsal ayrışımların LDP altında oluşturulması problemini inceleyerek, özellikle iki yönteme odaklanıyoruz: dördüzağaçlar ve kd-ağaçları. Dördüzağaçlar için iki çözüm geliştiriyoruz: merkezi DP literatüründen esinlenen bir temel çözüm ve kullanıcılardan tek bir veri toplama adımı kullanan, yoğunluk tahminlerini diğer düğümlere ileten ve dördüzağaç üzerinde yapısal düzeltmeler gerçekleştiren bir önerilen çözüm. Kd-ağaçları, veriye bağımlı olan düğüm medyanlarına dayandığı için, tek bir veri toplama adımını kullanarak kd-ağaçları oluşturmanın mümkün olmadığını gözlemliyoruz. Bu nedenle, her ağaç derinliğinde düğüm medyanlarını kestirmek için yeni bir algoritma kullanan, üstten aşağıya doğru kd-ağaçları oluşturan iteratif bir çözüm öneriyoruz. Dört gerçek konumsal veri kümesi, çeşitli fayda ölçütleri, değişen mahremiyet bütçeleri ve ağaç parametreleri kullanarak dördüzağaç ve kd-ağacı algoritmalarımızı deneysel olarak değerlendiriyoruz. Sonuçlar, algoritmalarımızın pratikte yüksek fayda sağlayan doğru konumsal ayrışımların oluşturulmasını mümkün kıldığını göstermektedir. Özellikle, dördüzağaçlarımız ve kd-ağaçlarımız mevcut bir yöntemle karşılaştırıldığında, konumsal yoğunluk sorgularını yanıtlarken önemli ölçüde daha düşük hata oranlarına ulaşmaktadır (10 kata kadar iyileşme).

ACKNOWLEDGMENTS

I would like to start by expressing my profound gratitude to my advisor and mentor, Assist. Prof. Mehmet Emre Gürsoy, for his consistent guidance and support during my degree. I owe my development as a researcher and the completion of this thesis to his support.

I am grateful to Assoc. Prof. Dr. Alptekin Küpçü from Koç University and Prof. Dr. Yücel Saygın from Sabancı University for participating in my thesis jury committee. Their valuable comments and feedback helped me improve my thesis.

I would like to extend my thanks to Mom and Dad for their love and support throughout my life. I want to convey my gratitude to the members of the SPADE Lab and all other students and co-workers at Koç University with whom I had the opportunity to collaborate. I particularly thank Efehan Güner and Berkay Kemal Balioğlu for the support they provided during my time at Koç University.

Finally, this research was supported by The Scientific and Technological Research Council of Turkiye (TUBITAK) under project number 121E303. I sincerely thank TUBITAK for their support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

LDP	Local Differential Privacy
DP	Differential Privacy
OUE	Optimized Unary Encoding
AQE	Average Query Error
TED	Tree Edit Distance
NDD	Node Density Difference

Chapter 1

INTRODUCTION

The rising popularity of smartphones, GPS-enabled mobile devices, social networks, connected vehicles, and the Internet of Vehicles all contribute to the ubiquity of location-based services (LBSs) and applications. As a consequence, large volumes of spatial data are nowadays available for collection, storage, and analysis. However, ensuring the privacy of spatial data is crucial because it contains sensitive information about an individual's movements, habits, and preferences, potentially revealing personal details such as home and work addresses, frequented places, and social relationships.

Local Differential Privacy (LDP) has recently emerged as a state-of-the-art privacy standard in academia and the industry [Cormode et al., 2018, Ding et al., 2017, Erlingsson et al., 2014, Gursoy et al., 2019, Yang et al., 2020]. With the rising adoption of LDP and the need to protect users' location privacy, there is increasing interest in applying LDP to spatial data. Indeed, LDP has been applied to spatial data aggregation [Chen et al., 2016], LBS usage [Hong et al., 2021, Wang et al., 2022], spatial query answering [Tire and Gursoy, 2024], trajectory collection [Yang et al., 2022] and sharing [Cunningham et al., 2021, Du et al., 2023], as well as indoor positioning [Kim et al., 2018, Navidan et al., 2022]. On the other hand, *spatial decompositions*, which are a common approach to dealing with big spatial data, have not received much attention under LDP. Spatial decompositions partition (decompose) the geographical space into smaller subspaces through the likes of tree-based and grid-based data structures. They have been used in the centralized DP literature for range query answering [Cormode et al., 2012, Shaham et al., 2022, Yan et al., 2020], spatial indexing and modeling [Niknami et al., 2020, Zhang

et al., 2016], count-of-counts histograms [Kuo et al., 2018], data summarization and visualization [Bagdasaryan et al., 2022, Qardaji et al., 2013], as well as synthetic data generation [Gursoy et al., 2020, Gursoy et al., 2018a, He et al., 2015]; however, their applications in LDP are relatively scarce.

Motivated by the above, in this thesis, we study the problem of building hierarchical spatial decompositions for spatial data under LDP. In particular, we focus on two types of popular decompositions: quadtrees and kd-trees. For quadtrees, we first propose a *baseline solution* inspired by a method from the centralized DP literature [Zhang et al., 2016] 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 quadtree structures 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 are achieved for the quadtree in a top-down manner.

For kd-trees, we observe that it is not possible to perform a single step of estimation and then propagate the densities since kd-trees rely on node medians which are data-dependent. Therefore, we propose a solution that iteratively constructs a kd-tree in top-down fashion, which utilizes a novel algorithm for estimating the medians of nodes at each depth of the kd-tree. Our algorithm bucketizes the geographical regions covered by a kd-tree node, obtains LDP density estimates for each bucket, and then finds which bucket contains the median value based on the estimates. To determine the densities of children nodes, we propose two strategies: (i) our initial strategy uses the LDP estimates obtained in the previous step, and (ii) our improved strategy evenly divides the parent's density. We experimentally observe that the im-

proved strategy, albeit simpler, provides higher accuracy than the initial strategy. This is because our median estimation algorithm typically achieves high accuracy in determining the median (higher than bucket-wise density estimation), therefore evenly dividing based on the LDP median is effective.

We experimentally evaluate our quadtree and kd-tree algorithms using four real-world spatial datasets, three utility metrics (AQE, TED, NDD), and typical privacy budget values such as $\varepsilon = 0.1, 0.5, 1, 2$. For quadtrees, results show that the proposed solution consistently outperforms the baseline solution in a variety of settings and quadtree parameters. For kd-trees, results show that the improved strategy yields lower error compared to the initial strategy in the majority of the experiments. Furthermore, upon experimenting with our novel LDP median estimation method, we observe that it can estimate LDP medians with generally $\leq 2\%$ error (compared to the true medians) when $\varepsilon \geq 0.5$. Finally, we compare our quadtree and kd-tree algorithms against a state-of-the-art work from the literature for answering spatial density queries under LDP, abbreviated as ASDQ-LDP [Tire and Gursoy, 2024]. Comparison results show that our algorithms achieve substantially lower errors under various datasets and ε values, offering up to 10-fold improvement in some settings.

1.1 Contributions

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

1. We propose two solutions for building quadtrees under LDP: 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.
2. We propose a solution that iteratively constructs a kd-tree under LDP in top-down fashion, which internally utilizes a novel algorithm for estimating nodes' medians at each depth. Furthermore, two strategies are proposed and evaluated for determining the densities of children nodes in kd-trees.
3. We experimentally evaluate all algorithms and solutions using four real-world

spatial datasets, multiple ε values, multiple metrics, and parameters. Results demonstrate that our algorithms are effective, and they result in accurate spatial decompositions that can be used in practice.

4. Finally, we experimentally compare our quadtrees and kd-trees against ASDQ-LDP [Tire and Gursoy, 2024], a state-of-the-art work for answering spatial density queries under LDP. Results show that our quadtrees and kd-trees achieve substantially lower errors under various settings.

1.2 Thesis Organization

The rest of the thesis is organized as follows. In Chapter 2, we survey the related work and highlight our main differences. In Chapter 3, we establish preliminaries regarding the data model, LDP, quadtrees, and kd-trees. In Chapter 4, we provide our algorithms for building quadtrees under LDP. In Chapter 5, we provide our algorithms for building kd-trees under LDP. Experimental evaluations are reported and experiment results are discussed in Chapter 6. Chapter 7 concludes this thesis.

Chapter 2

RELATED WORK

2.1 LDP for Spatial Data

LDP has gained widespread acceptance as a popular privacy notion in recent years, and it has also been deployed in the industry [Apple, 2020, Cormode et al., 2018, Ding et al., 2017, Erlingsson et al., 2014, Gursoy et al., 2019, Yang et al., 2020]. With the rising popularity of LDP, there is increasing interest in applying it to spatial data, considering the ubiquity of location-based services (LBS) and the need to protect users' location privacy. Chen et al. [Chen et al., 2016] proposed a variant of LDP, called personalized LDP, for spatial data aggregation. Hong et al. [Hong et al., 2021] proposed a perturbation mechanism designed to reduce the error of each perturbed location under LDP. Wang et al. [Wang et al., 2022] developed L-SRR (an LDP version of Staircase Randomized Response) to privately collect users' locations while they remain useful for LBS applications such as traffic density estimation and k-nearest neighbors. Yang et al. [Yang et al., 2022] studied the problem of collecting location trajectories under LDP. Cunningham et al. [Cunningham et al., 2021] proposed a technique that utilizes hierarchical n-grams for real-world trajectory sharing with LDP. Du et al. [Du et al., 2023] proposed LDPTrace for synthesizing realistic location trajectories using aggregate statistics collected with LDP. Kim et al. [Kim et al., 2018] and Navidan et al. [Navidan et al., 2022] applied LDP to indoor positioning data. Recently, Tire and Gursoy [Tire and Gursoy, 2024] studied the problem of answering spatial density queries under LDP. Although these works apply LDP to spatial data, they do not have the goal of building spatial decompositions.

2.2 Spatial Decompositions Under Centralized DP

Spatial decompositions have been relatively less studied under LDP, but there exist several works on building spatial decompositions under the centralized form of differential privacy (DP). Xiao et al. [Xiao et al., 2010] presented a spatial decomposition algorithm based on kd-trees for handling statistical queries (e.g., OLAP queries) under DP. Cormode et al. [Cormode et al., 2012] proposed algorithms to build hierarchical spatial decompositions such as quadtrees and kd-trees under DP. Zhang et al. [Zhang et al., 2016] developed the PrivTree algorithm, which enabled constructing a DP spatial decomposition without requiring a pre-defined recursion limit (height). Grid-based decompositions under DP, which are a popular alternative to hierarchical decompositions, have been introduced by Qardaji et al. [Qardaji et al., 2013] in the form of uniform and adaptive grids. They have later been used in other works such as [Gursoy et al., 2018a, Gursoy et al., 2018b]. Hierarchical reference systems proposed by He et al. [He et al., 2015] employ hierarchically organized grids with different granularities, ordered from coarse to fine-grained.

More recently, Niknami et al. [Niknami et al., 2020] 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. [Li et al., 2021], 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. [Yan et al., 2020] proposed an unbalanced quadtree partitioning algorithm for improving query accuracy in publishing spatial data with DP. Similarly, Liu et al. [Liu et al., 2022] 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. [Shaham et al., 2022], 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.

2.3 Spatial Decompositions Under LDP

As stated above, although hierarchical spatial decompositions have been studied under centralized DP, their applications to local DP (LDP) are novel. Our prior work on this topic [Alptekin and Gursoy, 2023] proposed algorithms for building quadtrees under LDP. In this thesis, we improve [Alptekin and Gursoy, 2023] by proposing algorithms for kd-trees in addition to quadtrees, and by demonstrating utility improvements compared to another state-of-the-art work [Tire and Gursoy, 2024].

Chapter 3

PRELIMINARIES

3.1 Data Model and Background

Let Ω denote the two-dimensional geographical space subject to decomposition, and let $\mathcal{U} = \{u_1, u_2, u_3, ...\}$ denote the set of users. The number of users is represented by $n = |\mathcal{U}|$. The true location of each user $u_i \in \mathcal{U}$ is denoted by l_i, which is a tuple consisting of latitude and longitude coordinates. By definition, $l_i \in \Omega$ for all u_i. We denote by $l_i[x]$ the longitude of l_i (x-axis) and by $l_i[y]$ the latitude of l_i (y-axis). It is assumed that Ω itself is not sensitive but each user's location l_i is sensitive. For example, if users are from the city of Istanbul, then Ω corresponds to the geographic boundaries of Istanbul, which is publicly known. However, each l_i corresponds to specific GPS coordinates of user u_i, which is privacy-sensitive.

A spatial decomposition decomposes the geographical space Ω into smaller subspaces. There are mainly three types of spatial decompositions:

- Grid-based decompositions utilize uniform or data-adaptive grid structures to divide Ω into cells.

- Hierarchical decompositions utilize tree-based data structures (such as kd-trees or quadtrees) for dividing Ω into smaller regions, each represented using a tree node.

- Hybrid decompositions use a mixture of grid-based and hierarchical approaches, e.g., they organize the decomposition as a hierarchy of grids [He et al., 2015, Mokbel et al., 2006].

In this thesis, we focus on hierarchical decompositions which have been commonly used in the literature for purposes such as range query answering [Cormode et al.,

2012, Shaham et al., 2022, Yan et al., 2020], spatial indexing and modeling [Niknami et al., 2020, Zhang et al., 2016], count-of-counts histograms [Kuo et al., 2018], data summarization and visualization [Bagdasaryan et al., 2022, Qardaji et al., 2013], and synthetic data generation [Gursoy et al., 2018b, He et al., 2015]. Among hierarchical decompositions, we focus on two fundamental ones: quadtrees and kd-trees [Cormode et al., 2012, Li et al., 2021, Samet, 1984]. We propose methods to build quadtrees and kd-trees while protecting the privacy of each user's location l_i using local differential privacy (LDP).

3.2 Local Differential Privacy

Local Differential Privacy (LDP) has recently emerged as a state-of-the-art privacy standard in academia and the industry [Cormode et al., 2018, Ding et al., 2017, Erlingsson et al., 2014, Thakurta et al., 2017, Xiong et al., 2020]. LDP enables each user to locally perturb their true data on their own device using a randomized algorithm Ψ and then send the perturbed output to the data collector (i.e., the server). After the server collects perturbed data from many users, it performs estimation to recover aggregate statistics pertaining to the general user 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:*

$$\forall y \in Range(\Psi): \quad \frac{Pr[\Psi(l_i) = y]}{Pr[\Psi(l_i^*) = y]} \leq e^{\varepsilon} \tag{3.1}$$

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 other third party who observes y) will not be able to distinguish between the user's actual location l_i and an incorrect location l_i^* beyond the probability odds ratio controlled by e^{ε}. The strength of the privacy protection is controlled by the parameter ε, commonly known as the *privacy budget*. Lower ε yields stronger privacy.

LDP satisfies the sequential composition property [Dwork et al., 2014, Wang et al., 2018, Ye et al., 2019], which enables the total ε budget to be spent in multiple steps. This is beneficial in the construction of hierarchical spatial decompositions since we can construct each step of the hierarchy with a portion of the privacy budget, such that the overall summation satisfies ε-LDP by sequential composition.

Definition 2 (Sequential Composition) *Consider m 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 [Gursoy et al., 2022, Gursoy et al., 2019, Wang et al., 2017]. New systems and applications often use these protocols as building blocks. In this thesis, we use the *Optimized Unary Encoding (OUE)* protocol as our building block due to its higher accuracy compared to many other protocols [Wang et al., 2017]. 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:

$$\Pr\Big[B'_i[j] = \Psi(B_i[j]) = 1\Big] = \begin{cases} \frac{1}{2} & \text{if } B_i[j] = 1 \\ \frac{1}{e^{\varepsilon}+1} & \text{if } B_i[j] = 0 \end{cases} \tag{3.2}$$

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

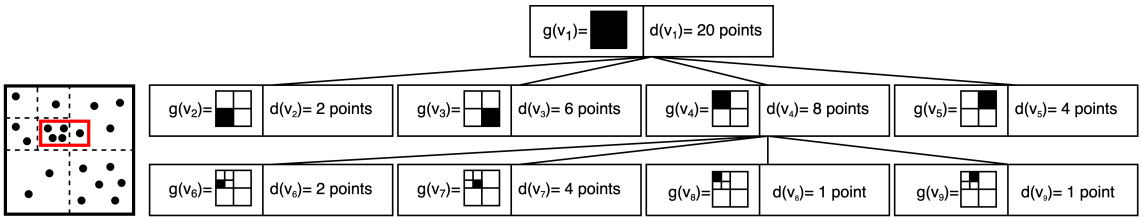

Figure 3.1: Sample users with their locations represented as points (on the left) and the corresponding quadtree (on the right).

the reported counts of each position $j \in [1, |B_i|]$, which we denote by $\widehat{C}(j)$:

$$\widehat{C}(j) = \sum_{i=1}^{n} B_i'[j] \tag{3.3}$$

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:

$$\bar{C}(j) = \frac{2 \cdot \left((e^{\varepsilon} + 1) \cdot \widehat{C}(j) - |\mathcal{U}|\right)}{e^{\varepsilon} - 1} \tag{3.4}$$

3.3 Quadtrees

Quadtrees are a popular and well-known method for hierarchical spatial decompositions. 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 3.1. 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 location points than a pre-defined threshold θ.

We establish some notation and terminology with the help of Figure 3.1. 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 3.1, $h(v_1) = 1$, $h(v_2) = h(v_3) = 2$, $h(v_7) = 3$. 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 3.1, 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 3.1, $d(v_1) = 20$, $d(v_2) = 2$, $d(v_3) = 6$, 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 3.1, $c(v_4) = \{v_6, v_7, v_8, v_9\}$. By definition of quadtrees, 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 $c(v_i) = \{v_{i1}, v_{i2}, v_{i3}, v_{i4}\}$. It holds that:

$$g(v_i) = g(v_{i1}) \cup g(v_{i2}) \cup g(v_{i3}) \cup g(v_{i4}) \tag{3.5}$$

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:

$$d(v_i) = d(v_{i1}) + d(v_{i2}) + d(v_{i3}) + d(v_{i4}) \tag{3.6}$$

One of the key applications of quadtrees is in answering spatial count (density) queries [Cormode et al., 2012, Li et al., 2021, Zhang et al., 2016]. 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.

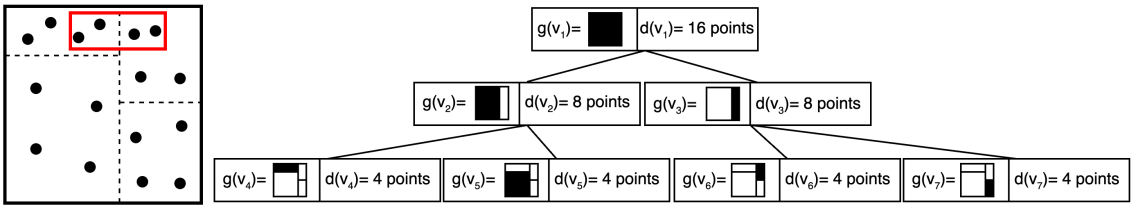

Figure 3.2: Sample users with their locations represented as points (on the left) and the corresponding kd-tree (on the right).

We exemplify this process with the help of Figure 3.1. 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 $d(v_7) \times \frac{||g(v_7) \cap q||}{||g(v_7)||}$.

3.4 Kd-trees

Another fundamental and popular method for hierarchical spatial decompositions is kd-trees (k-dimensional trees). At each step, a kd-tree alternates between the x-axis and the y-axis, and divides the geographical space into two regions according to the median of the chosen axis. Since the division is according to the median, it is necessary to find the median in a data-dependent manner, and the resulting child nodes may have different geographic sizes. This is different from quadtrees in which the resulting child nodes have equal sizes. Similar to quadtrees, in kd-trees, the division of a node into two regions will keep occurring recursively until: (i) the maximum height limit h^* is reached, or (ii) the current node contains fewer location points than a pre-defined threshold θ.

A sample kd-tree is shown in Figure 3.2. The division at the root is performed with respect to the x-axis, according to the median of the location points. Hence, 16 points initially in v_1 are divided into v_2 and v_3 evenly. Note that the geographic sizes $g(v_2)$ and $g(v_3)$ are indeed different – since there are more points on the right

compared to the left, the median is towards the right rather than the middle of the x-axis. Next, the y-axis is used when dividing v_2 and v_3. Since the median of v_2 is found independently from the median of v_3, the division of v_2 into v_4 and v_5 results in different sized nodes ($g(v_4)$ is small whereas $g(v_5)$ is large). In contrast, since the median of v_3 is towards the middle of the y-axis, the division results in similar sized v_6 and v_7. This highlights the need for determining the median of each node separately and in a data-dependent fashion.

For kd-trees, we inherit most of the notation established in Section 3.3 for quadtrees. To differentiate between a kd-tree and a quadtree, we denote a kd-tree by K. The nodes of K are denoted by $\mathcal{V} = \{v_1, v_2, v_3, ...\}$, the depth of node v is denoted by $h(v)$, the density of v is denoted by $d(v)$, the geographic space of v is denoted by $g(v)$, children of v are denoted by $c(v)$, and the parent of v is denoted by $p(v)$. In a kd-tree, a node can have no children or exactly two children. For node v_i and its children $c(v_i) = \{v_{i1}, v_{i2}\}$, the following equation holds:

$$g(v_i) = g(v_{i1}) \cup g(v_{i2}) \tag{3.7}$$

For kd-trees, we define two additional notations as follows: Let $g_x(v)$ denote the longitude bounds (x-axis bounds) of node v and let $g_y(v)$ denote the latitude bounds (y-axis bounds) of v. We denote by $min(g_x(v))$ the minimum longitude, $max(g_x(v))$ the maximum longitude, $min(g_y(v))$ the minimum latitude, and $max(g_y(v))$ the maximum latitude of v.

Chapter 4

BUILDING QUADTREES UNDER LDP

In this chapter, we describe two solutions for building quadtrees for spatial data under LDP. 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 4.1. Then, our baseline solution is given in 4.2. We observe that although the iterative noise addition approach used in the baseline solution satisfies LDP, it is not desirable in practice since it causes large noise accumulation and therefore low utility. This motivates the proposal of our new approach in Section 4.3, which yields higher utility than the baseline solution.

4.1 Quadtrees under Centralized DP

Consider the quadtree building algorithm presented in [Zhang et al., 2016], 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 [Dwork et al., 2014]. 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.

4.2 Baseline Solution for Quadtrees

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 $\hat{\varepsilon} = \varepsilon/(h^* - 1)$.

Algorithm 1: Baseline quadtree solution

Input : $\mathcal{U}$, Ω, θ, h^*, ε

Output : Quadtree Q

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

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 = 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 in the next paragraph. 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 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 3.2. 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.

Algorithm 2: Get_Estimates function

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

Output: *estimates*

`// Server-side`

Server sends $\mathcal{N}$ to each user in $\mathcal{U}$

`// User-side`

foreach $u_i \in \mathcal{U}$ **do**

  u_i constructs his/her bitvector B_i with length $|\mathcal{N}|$ such that:

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

  u_i perturbs B_i to satisfy $\hat{\varepsilon}$-LDP according to Equation 3.2

  u_i sends resulting perturbed bitvector B_i' to server

`// Server-side`

estimates $\leftarrow [\,]$ `// initialize empty`

for $j = 1$ *to* $|\mathcal{N}|$ **do**

  Server computes reported count of position j as: $\widehat{C}(j) = \sum_{i=1}^{|\mathcal{U}|} B_i'[j]$

  Server computes estimate $\bar{C}(j)$ as: $\bar{C}(j) = \frac{2 \cdot \left((e^{\hat{\varepsilon}}+1) \cdot \widehat{C}(j) - |\mathcal{U}|\right)}{e^{\hat{\varepsilon}}-1}$

  Server appends $\bar{C}(j)$ to *estimates*

return *estimates*

4.3 Proposed Solution for Quadtrees

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.

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 3.6 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

Algorithm 3: Proposed quadtree 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

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.

Chapter 5

BUILDING KD-TREES UNDER LDP

In this chapter, we describe our solution for building kd-trees for spatial data under LDP. As opposed to quadtrees, we cannot perform a single step of estimation and then propagate the densities upwards in kd-trees because the kd-tree requires finding the medians of each node, which are node-dependent and data-dependent. As a result, our solution learns medians from $\mathcal{U}$ in every depth of the kd-tree. We describe our general solution approach in Section 5.1 and then describe its details in Sections 5.2 and 5.3.

5.1 Overview of our Kd-tree Solution

Our proposed solution for building a kd-tree under LDP is shown in Algorithm 4. On line 1, the algorithm divides the total ε budget into $h^* - 1$ pieces, such that each piece will be used in learning medians of nodes at depths $1, 2, ..., h^* - 1$. The kd-tree is initialized with root node v_1, and by definition of kd-trees, $g(v_1) = \Omega$ and $d(v_1) = |\mathcal{U}|$. Then, between lines 6-22, the algorithm iterates depth-by-depth, and at each iteration, it estimates the medians and performs necessary node splits at the current depth. Initially, the depth is set as $i \leftarrow 1$ (line 4). Considering that the kd-tree alternates between splitting the x-axis and y-axis, the initial axis for median computation and node splitting is selected as x-axis (on line 5). Note that it is also possible to select the y-axis as the initial axis by trivially modifying line 5.

The main loop of the algorithm (lines 6-22) iterates depth-by-depth. First, it retrieves the *nodes* at the current depth (line 7). Then, for all *nodes*, it uses the FIND_MEDIANS function to retrieve the medians and densities of potential children (line 8). More details regarding the FIND_MEDIANS function are given in Section 5.2. Afterward, between lines 9-17, each node $v \in$ *nodes* is checked regarding whether it

Algorithm 4: Proposed kd-tree solution

Input : $\mathcal{U}$, Ω, θ, h^*, ε, α

Output : Kd-tree K

1 $\hat{\varepsilon} \leftarrow \varepsilon/(h^* - 1)$
2 Initialize kd-tree K with root node v_1
3 Set $g(v_1) = \Omega$ and $d(v_1) = |\mathcal{U}|$
4 $i \leftarrow 1$ `// current depth`
5 $axis \leftarrow$ "x" `// alternate between x and y`
6 **while** $i \leq h^*$ **do**
7 $\quad nodes \leftarrow$ list of nodes in K with depth $= i$
8 $\quad medians,\ c_1\_dens,\ c_2\_dens \leftarrow$ FIND_MEDIANS($nodes, \mathcal{U}, \hat{\varepsilon}, \alpha, axis$)
$\quad$ `// for each node at depth` i`, determine if it should split`
9 $\quad$ **for** $j = 1$ *to* $|nodes|$ **do**
10 $\quad\quad v \leftarrow nodes[j]$
11 $\quad\quad median \leftarrow medians[j]$
12 $\quad\quad$ **if** $d(v) \geq \theta$ *and* $i + 1 \leq h^*$ **then**
13 $\quad\quad\quad$ Split v into two children v_{c_1} and v_{c_2} according to *median*
14 $\quad\quad\quad$ Add v_{c_1} and v_{c_2} to K
15 $\quad\quad\quad$ Set densities of v_{c_1} and v_{c_2} `// Sec 5.3`
16 $\quad\quad$ **else**
17 $\quad\quad\quad$ Do not split v
18 $\quad i \leftarrow i + 1$
19 $\quad$ **if** $axis =$ "x" **then**
20 $\quad\quad axis \leftarrow$ "y"
21 $\quad$ **else**
22 $\quad\quad axis \leftarrow$ "x"
23 **return** K

satisfies the splitting conditions, i.e., density $\geq \theta$ and depth $\leq h^*$ (line 12). If so, v is split into its two children v_{c_1}, v_{c_2} according to its *median*, and the children are

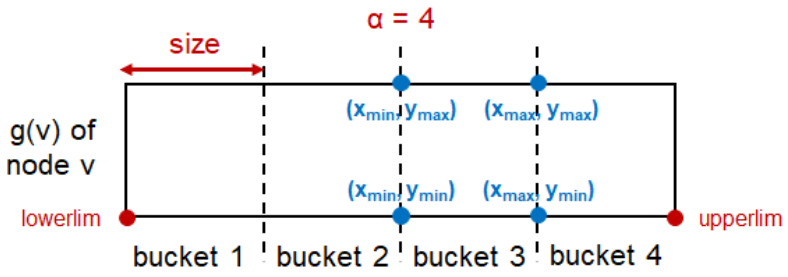

Figure 5.1: Sample division of $g(v)$ into $\alpha = 4$ equi-sized buckets. Bounds x_{min}, x_{max}, y_{min}, y_{max} shown in blue are for bucket 3. Since all buckets are derived from node v, the owner of all buckets is v.

added to K. For determining the densities of the children, it is possible to make use of the children densities (c_1_dens and c_2_dens) returned by FIND_MEDIANS. This is done on line 15; we describe two alternative strategies for this part in Section 5.3. Finally, between lines 18-22, the current depth i is advanced by 1, and the axis is changed from x to y or y to x in order to alternate between them. On the last line, the algorithm returns the final kd-tree K.

5.2 Finding Medians

The FIND_MEDIANS function is a key component of our kd-tree solution. This function takes as input a list of kd-tree *nodes*, user population $\mathcal{U}$, allocated privacy budget $\hat{\varepsilon}$, bucket count parameter α, and the *axis* that will be split. It returns three lists with sizes equal to the size of *nodes*: $medians[j]$ stores the estimated (using LDP) median of $nodes[j]$, $c_1\_dens[j]$ stores the first child's estimated density, and $c_2\_dens[j]$ stores the second child's estimated density. Its pseudocode is given in Algorithm 5. We explain the algorithm in three main steps.

Step 1: Server-side computation of buckets. For finding the median of a node v, we make use of *buckets*. Buckets divide the geographic space $g(v)$ into small pieces with respect to the *axis*. The number of buckets is determined by the α parameter. Each bucket has an *owner*, i.e., if the bucket results from dividing v into small pieces, then the owner of this bucket is v. An example of dividing v into $\alpha = 4$ buckets with respect to the x-axis is shown in Figure 5.1. We can observe

Algorithm 5: Find_Medians function

Input : $nodes, \mathcal{U}, \hat{\varepsilon}, \alpha, axis$

Output: $medians$, c_1_dens, c_2_dens

`// Step 1: Server-side computation of buckets`

1 $buckets \leftarrow$ Initialize empty list
2 **foreach** v *in* $nodes$ **do**
3 **if** $axis =$ "x" **then**
  `// Splitting by x-axis`
4 $upperlim \leftarrow max(g_x(v))$
5 $lowerlim \leftarrow min(g_x(v))$
6 **else**
  `// Splitting by y-axis`
7 $upperlim \leftarrow max(g_y(v))$
8 $lowerlim \leftarrow min(g_y(v))$
9 $size \leftarrow (upperlim - lowerlim)/\alpha$
10 $curr \leftarrow lowerlim$
11 **while** $curr + size \leq upperlim$ **do**
12 **if** $axis =$ "x" **then**
13 Create *bucket* with $x_{min} = curr$, $x_{max} = curr + size$, $y_{min} = min(g_y(v))$ and $y_{max} = max(g_y(v))$
14 **else**
15 Create *bucket* with $x_{min} = min(g_x(v))$, $x_{max} = max(g_x(v))$, $y_{min} = curr$ and $y_{max} = curr + size$
16 Set owner of *bucket* as v
17 Append *bucket* to *buckets*
18 $curr \leftarrow curr + size$

`// Step 2: Get LDP estimates for buckets from users`

19 $estimates \leftarrow$ GET_ESTIMATES($buckets, \mathcal{U}, \hat{\varepsilon}$)
20 **for** $j = 1$ *to* $|buckets|$ **do**
21 Set density of $buckets[j]$ as $estimates[j]$

`// Step 3: Construct medians, c1_dens, and c2_dens`

22 Initialize $medians$, c_1_dens, c_2_dens as empty lists
23 **foreach** v *in* $nodes$ **do**
24 $relbuckets \leftarrow$ Retrieve those buckets for which owner $= v$
25 $med, c1, c2 \leftarrow$ CALC_MED($relbuckets, axis$)
26 Append med to $medians$
27 Append $c1$ to c_1_dens
28 Append $c2$ to c_2_dens
29 **return** $medians$, c_1_dens, c_2_dens

that $g(v)$ is split into 4 equal-sized buckets such that the horizontal axis (x-axis) is divided into 4, and the bounds of the y-axis remain the same across all buckets. This exemplifies the division of one v into 4 buckets; between lines 2-18 of Algorithm 5,

this process is performed for all nodes v in *nodes*. All buckets are collectively stored in a list named *buckets*. Variables *upperlim*, *lowerlim*, *size* used in Algorithm 5 are also exemplified in Figure 5.1.

Step 2: Get LDP estimates for buckets from users. Now consider that all buckets have been created and stored in the list named *buckets*. To find medians, we need to estimate the densities of all buckets according to users' locations. We make use of the GET_ESTIMATES function previously established in Algorithm 2 for this purpose. This function spends $\hat{\varepsilon}$ and returns the density of each bucket in *buckets*.

Step 3: Construct medians, c_1_dens, and c_2_dens. Now that we have *buckets* and their densities, the goal of this step is to construct the three return values of the FIND_MEDIANS function: *medians*, c_1*_dens*, and c_2*_dens*. When computing the median and densities of the two children of node v, Algorithm 5 retrieves the relevant buckets (*relbuckets*) for this computation, i.e., those buckets for which the owner of the bucket is v. Then, Algorithm 5 calls the CALC_MED function, which is formally described in Algorithm 6.

We explain Algorithm 6 using the illustration in Figure 5.2, which is a continuation of the example in Figure 5.1. At the start of Algorithm 6, the relevant four buckets have been identified and their densities have been estimated (written inside the buckets in Figure 5.2). The goal is to find the median of $g(v)$ and the densities of the two resulting children if v was split according to this median. Algorithm 6 first computes the sum of the estimated densities (denoted by *dsum*), which is $dsum = 87$ in Figure 5.2. Then, the algorithm searches which bucket the median falls into, starting with the leftmost bucket. It keeps track of the sum of densities observed in the variable $currsum$, e.g., after visiting bucket 1 currsum is 12; after visiting bucket 2 currsum is 31; after visiting bucket 3 currsum is 47, etc. When currsum becomes equal to or higher than $dsum/2$, then that means the most recently visited bucket contains the median (line 11). In that case, the median is retrieved, e.g., as the middle point of bucket 3 in Figure 5.2. Then, the density of the bucket containing the median is distributed among the first child and second child of v. Thus, in Figure 5.2 v_{c_1} gets a total density of $12 + 19 + 26/2 = 44$, whereas v_{c_2}

Algorithm 6: Calc_Med function

Input : *relbuckets*, *axis*

Output: *med*, *c*1, *c*2

```
1  dsum ← Sum of densities of all buckets in relbuckets
2  if axis = "x" then
3  |   Sort relbuckets in increasing order of their x_min
4  else
5  |   Sort relbuckets in increasing order of their y_min
6  currsum ← 0
7  for j = 1 to |relbuckets| do
8  |   bucket ← relbuckets[j]
9  |   den ← density of bucket
10 |   currsum ← currsum + den
11 |   if currsum ≥ dsum/2 then
   |   |   // This is the bucket containing median
12 |   |   if axis = "x" then
13 |   |   |   Retrieve x_max and x_min of bucket
14 |   |   |   med ← (x_max + x_min)/2
15 |   |   else
16 |   |   |   Retrieve y_max and y_min of bucket
17 |   |   |   med ← (y_max + y_min)/2
18 |   |   break
19 c1 ← currsum − ⌊den/2⌋
20 c2 ← dsum − c1
21 return med, c1, c2
```

gets a total density of 87 - 44. These values are stored in the appropriate variables in Algorithm 6 and returned (lines 19-21) to Algorithm 5.

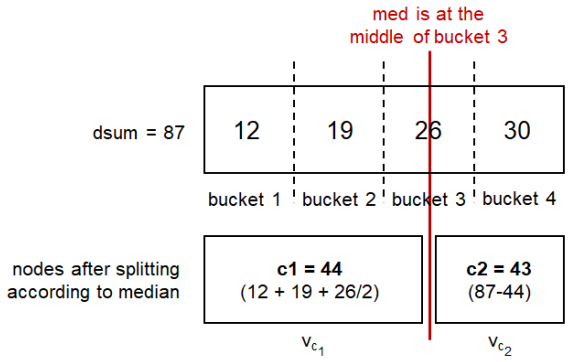

Figure 5.2: Illustration of computing the median of v and estimated densities of two children v_{c_1}, v_{c_2} (Algorithm 6).

5.3 Determining Child Densities

Another key component of our kd-tree solution (Algorithm 4) is determining the densities of children v_{c_1} and v_{c_2}. The densities of the children can be determined directly using the results of LDP estimation, as described in the previous section (Figure 5.2). That is, line 15 of Algorithm 4 is replaced by:

$$\text{Set} \quad d(v_{c_1}) \leftarrow c_1\_dens[j]$$
$$\text{Set} \quad d(v_{c_2}) \leftarrow c_2\_dens[j]$$

We call this our initial strategy.

On the other hand, we experimentally observed that our initial strategy can yield erroneous densities since LDP estimations are noisy. Therefore, instead of using this strategy, we developed a second strategy in which we utilize the key property of kd-trees: since nodes are split according to their median, the resulting two children will have equal or near-equal densities. Thus, the density of the parent can be directly divided into two. Examples of this property were also shown in Figure 5.2. In this case, line 15 of Algorithm 4 is replaced by:

$$\text{Set} \quad d(v_{c_1}) \leftarrow \frac{d(v)}{2}$$
$$\text{Set} \quad d(v_{c_2}) \leftarrow \frac{d(v)}{2}$$

We call this our improved strategy.

There is the following trade-off between the initial strategy and the improved strategy. If the median computed under LDP is close to the real median (and therefore node v is split from the right location) then the two children will indeed have equal or near-equal densities by definition of kd-trees. However, if the median computed under LDP is far from the real median, then the kd-tree will contain error. In contrast, the accuracy of the initial strategy relies on the correctness of LDP estimations for bucket densities. Consequently, if median computation is less noisy than bucket density estimation, one should choose the improved strategy over the initial strategy. If vice versa is true, then one should choose the initial strategy over the improved strategy. In practice, we observed that estimating the median is less noisy under LDP since it is one numeric value, whereas bucket estimations are a vector of numeric values that are more prone to LDP noise. In addition, our experiment results also showed that our median computation strategy is quite accurate. Hence, our improved strategy usually yields better results compared to our initial strategy, and therefore, we recommend using our improved strategy in practice.

Chapter 6

EXPERIMENTAL EVALUATION

6.1 Experiment Setup

We implemented all algorithms in Python. In this chapter, we experimentally compare them under varying ε, θ, h^*, and α parameters, as well as using multiple evaluation metrics. We use four real-world location datasets in our experiments: Kaggle, Brightkite, Gowalla, and Foursquare. We also perform an experimental comparison against a state-of-the-art work for answering spatial density queries under LDP [Tire and Gursoy, 2024] to demonstrate the practical benefits of our work and its superiority in this task. Each experiment is repeated 10 times and the average results are reported.

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 [ECML/PKDD,]. While the dataset contains full taxi trips (multiple location readings per trip), we preprocessed 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 [Cho et al., 2011]. 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 [Cho et al., 2011]. 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 [Yang et al., 2015]. 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.

6.2 Utility Metrics

We define our utility metrics using quadtrees (Q and Q'), but for kd-trees one may simply replace Q and Q' by K and K' and the remaining definitions of the metrics would stay the same. Let Q denote the noise-free quadtree that would be built without privacy protection, and 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 Q and 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:

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

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) [Chen et al., 2012, Gursoy et al., 2018a, Zhang et al., 2016]. 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 descendants must be counted as part of TED. If both v and v' have 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 trees 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:

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

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.

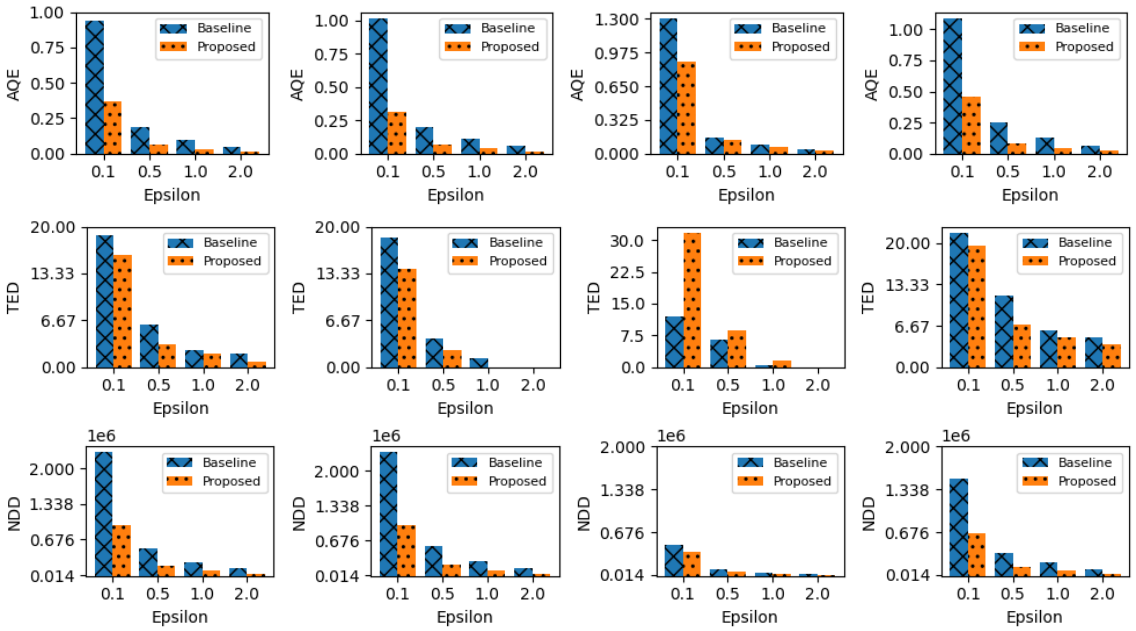

Figure 6.1: Results of our baseline vs proposed quadtree solutions with AQE, TED and NDD metrics (one metric each row) on Brightkite, Gowalla, Kaggle and Foursquare datasets (left to right).

6.3 Results of Quadtree Algorithms

We first evaluate our quadtree algorithms (baseline solution and proposed solution) from Section 4. In the first set of experiments, we keep the h^* and θ parameters constant ($h^* = 4$, $\theta = 10000$) and vary the ε parameter to observe its impact. The results are reported in Figure 6.1. Four popular (conventional) ε values are used: $\varepsilon = 0.1, 0.5, 1, 2$. 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. A similar observation holds for the Foursquare dataset – while the difference between the baseline solution and the 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 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 the 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 6.1 and 6.2. 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.

Finally, in Table 6.3, 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

Table 6.1: AQE and TED quadtrees under different heights h^* with threshold $\theta = 10000$.

			AQE				TED			
			$\varepsilon = 0.1$	$\varepsilon = 0.5$	$\varepsilon = 1$	$\varepsilon = 2$	$\varepsilon = 0.1$	$\varepsilon = 0.5$	$\varepsilon = 1$	$\varepsilon = 2$
Brightkite	$h^* = 3$	Baseline	0.317	0.056	0.028	0.014	0.0	0.0	0.0	0.0
		Proposed	0.156	0.030	0.014	0.006	0.0	0.0	0.0	0.0
	$h^* = 4$	Baseline	0.941	0.189	0.095	0.046	18.8	6.0	2.4	2.0
		Proposed	0.369	0.067	0.032	0.015	16.0	3.2	2.0	0.8
	$h^* = 5$	Baseline	1.924	0.423	0.213	0.092	105.2	66.8	41.2	19.2
		Proposed	0.637	0.146	0.080	0.036	110.4	55.2	16.0	4.8
Gowalla	$h^* = 3$	Baseline	0.292	0.053	0.026	0.011	0.0	0.0	0.0	0.0
		Proposed	0.140	0.022	0.013	0.005	0.0	0.0	0.0	0.0
	$h^* = 4$	Baseline	1.015	0.201	0.108	0.057	18.4	4.0	1.2	0.0
		Proposed	0.313	0.069	0.039	0.016	14.0	2.4	0.0	0.0
	$h^* = 5$	Baseline	2.076	0.436	0.215	0.094	118.4	68.8	40.0	17.6
		Proposed	0.850	0.172	0.074	0.032	122.4	49.2	17.6	7.6
Kaggle	$h^* = 3$	Baseline	0.396	0.058	0.025	0.011	4.0	1.2	0.0	0.0
		Proposed	0.296	0.051	0.020	0.008	5.2	1.6	0.0	0.0
	$h^* = 4$	Baseline	1.302	0.155	0.085	0.039	12.0	6.4	0.4	0.0
		Proposed	0.885	0.137	0.065	0.026	31.6	8.8	1.6	0.0
	$h^* = 5$	Baseline	3.823	0.562	0.145	0.064	52.0	31.2	4.0	0.0
		Proposed	1.815	0.290	0.133	0.054	74.0	29.6	8.8	0.8
Foursquare	$h^* = 3$	Baseline	0.373	0.062	0.038	0.019	0.0	0.0	0.0	0.0
		Proposed	0.154	0.033	0.017	0.008	0.0	0.0	0.0	0.0
	$h^* = 4$	Baseline	1.085	0.249	0.128	0.060	21.6	11.6	6.0	4.8
		Proposed	0.462	0.084	0.045	0.024	19.6	6.8	4.8	3.6
	$h^* = 5$	Baseline	2.334	0.455	0.207	0.098	94.8	58.4	44.0	22.4
		Proposed	0.883	0.195	0.087	0.046	109.2	56.0	25.6	17.2

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.

Table 6.2: NDD of quadtrees under different heights h^* with threshold $\theta = 10000$.

			$\varepsilon = 0.1$	$\varepsilon = 0.5$	$\varepsilon = 1$	$\varepsilon = 2$
Brightkite	$h^* = 3$	Baseline	60.2	10.4	5.5	2.3
		Proposed	30.1	5.2	2.5	1.2
	$h^* = 4$	Baseline	229.5	51.7	26.0	13.8
		Proposed	94.1	18.5	9.7	4.0
	$h^* = 5$	Baseline	499.1	139.9	74.7	38.8
		Proposed	258.3	58.6	27.3	11.7
Gowalla	$h^* = 3$	Baseline	50.7	10.4	4.8	2.4
		Proposed	24.8	4.8	2.5	1.1
	$h^* = 4$	Baseline	234.2	56.5	28.4	14.0
		Proposed	95.9	20.9	9.5	4.3
	$h^* = 5$	Baseline	554.0	151.7	80.5	42.3
		Proposed	285.5	62.1	29.6	13.0
Kaggle	$h^* = 3$	Baseline	20.5	4.9	2.2	1.1
		Proposed	17.8	3.4	1.5	0.6
	$h^* = 4$	Baseline	47.0	9.4	5.1	2.5
		Proposed	37.2	7.4	3.2	1.4
	$h^* = 5$	Baseline	95.7	16.0	8.5	4.2
		Proposed	73.0	14.7	7.6	2.8
Foursquare	$h^* = 3$	Baseline	38.5	7.3	3.9	1.8
		Proposed	19.5	3.5	1.8	0.8
	$h^* = 4$	Baseline	149.5	36.0	20.1	9.8
		Proposed	65.8	14.4	8.0	3.8
	$h^* = 5$	Baseline	275.3	68.7	37.8	19.5
		Proposed	146.2	38.4	17.7	8.9

6.4 Results of Kd-tree Algorithms

In this section, we evaluate our kd-tree algorithms from Chapter 5. In the first set of experiments, we keep $\theta = 10000$ constant and vary the ε budget and h^*

Table 6.3: AQE of quadtrees with privacy budget $\varepsilon = 1$, varying h^* and θ.

		Baseline				Proposed			
	θ	1000	5000	10000	20000	1000	5000	10000	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

parameters. The results are shown in Table 6.4. First, recall that we had two strategies for determining child densities in kd-trees: the initial strategy and the improved strategy. Upon comparing them, we observe from Table 6.4 that the improved strategy yields lower AQE in the majority of the experiments. This shows that LDP median computation is indeed less noisy compared to LDP bucket density estimation. Therefore, in general, using the improved strategy rather than the initial strategy is preferable. Second, as ε increases, we observe that errors decrease both for the initial and improved strategies. This aspect is similar to quadtrees and it suits our expectations. Third, errors increase as h^* increases from 3 to 5, which is also similar to quadtrees. However, the speed of increase is slower in kd-trees compared to quadtrees. For example, although AQEs in quadtrees are lower than kd-trees when $h^* = 3$, vice versa can be true when $h^* = 4$ or 5. This shows that kd-trees are less affected by the increase in error as h^* increases. Fourth, we analyze the

Table 6.4: AQE of kd-trees with threshold $\theta = 10000$, varying ε and h^*.

			$\varepsilon = 0.1$	$\varepsilon = 0.5$	$\varepsilon = 1$	$\varepsilon = 2$
Brightkite	$h^* = 3$	Initial	0.378	0.139	0.077	0.073
		Improved	0.236	0.087	0.017	0.012
	$h^* = 4$	Initial	0.685	0.178	0.091	0.058
		Improved	0.383	0.148	0.046	0.060
	$h^* = 5$	Initial	1.158	0.324	0.169	0.104
		Improved	0.534	0.188	0.181	0.079
Gowalla	$h^* = 3$	Initial	0.324	0.093	0.079	0.077
		Improved	0.244	0.051	0.021	0.007
	$h^* = 4$	Initial	0.593	0.200	0.161	0.157
		Improved	0.343	0.139	0.083	0.076
	$h^* = 5$	Initial	0.972	0.288	0.213	0.167
		Improved	0.491	0.193	0.159	0.122
Kaggle	$h^* = 3$	Initial	0.308	0.081	0.046	0.032
		Improved	0.095	0.023	0.015	0.012
	$h^* = 4$	Initial	0.614	0.129	0.081	0.050
		Improved	0.253	0.056	0.032	0.028
	$h^* = 5$	Initial	1.029	0.167	0.118	0.081
		Improved	0.465	0.090	0.061	0.040
Foursquare	$h^* = 3$	Initial	0.391	0.099	0.042	0.021
		Improved	0.166	0.031	0.022	0.011
	$h^* = 4$	Initial	0.637	0.193	0.096	0.051
		Improved	0.377	0.107	0.059	0.032
	$h^* = 5$	Initial	1.419	0.301	0.184	0.092
		Improved	0.645	0.205	0.097	0.065

errors across different datasets. Interestingly, kd-trees yield lower error on Kaggle and Foursquare datasets, whereas these two datasets had the highest amount of error in the case of quadtrees. This shows that datasets and data distributions that are not handled well by quadtrees can be handled by kd-trees, and vice versa. Hence, considering that they work well on different datasets, kd-trees and quadtrees can complement each other in real-world tasks.

In the next set of experiments, we fix $\varepsilon = 1$ and $h^* = 4$, and vary the θ parameter. The results are shown in Table 6.5. We observe that there is no direct relationship

Table 6.5: AQE of kd-trees with the improved strategy, $\varepsilon = 1.0$ and $h^* = 4$. θ is varied between 5000 and 40000.

	$\theta = 5000$	$\theta = 10000$	$\theta = 20000$	$\theta = 40000$
Brightkite	0.098	0.046	0.104	0.100
Gowalla	0.083	0.083	0.094	0.078
Kaggle	0.036	0.032	0.029	0.021
Foursquare	0.056	0.059	0.065	0.067

between θ and AQE. For example, on the Brightkite dataset, AQE drops as θ is increased from 5000 to 10000, then increases as θ is increased from 10000 to 20000, and remains similar as θ is increased from 20000 to 40000. In contrast, on the Kaggle dataset, AQE drops consistently as θ is increased from 5000 to 40000. On the other hand, on the Foursquare dataset, AQE increases consistently as θ is increased from 5000 to 40000. Hence, it is not possible to establish a consistent relationship between the value of θ and AQE – the best value of θ can be different under different conditions.

6.5 Accuracy of LDP Median Computation

Recall from Chapter 5 that a significant component of our kd-tree solution is to find the medians of kd-tree nodes while satisfying LDP. In this section, we measure the accuracy of our proposed LDP median computation method, by comparing the medians found using our method versus the true medians. Let η denote the true median that would be found if all data could be accessed in plaintext (no LDP), and let η' denote the median found using our method satisfying LDP. The Median Error (ME) is defined as:

$$ME = \frac{|\eta' - \eta|}{max - min} * 100\% \tag{6.1}$$

where max and min denote the bounds of the axis for which median is being computed (x-axis or y-axis).

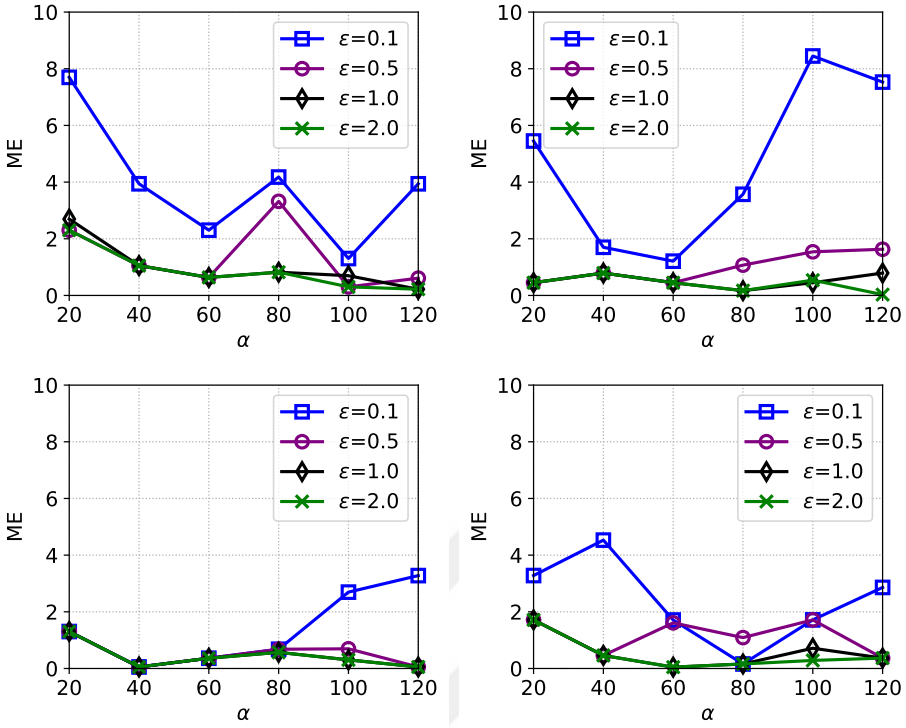

Figure 6.2: ME of medians found using our LDP algorithm with varying ε and datasets (order from left to right: Brightkite, Gowalla in the top row, Kaggle, Foursquare in the bottom row).

The α parameter plays an important role in median estimation since it controls the number of buckets. When there are few buckets, then the density of each bucket is high (better signal-to-LDP-noise ratio) but median computation is coarse since intra-bucket distributions cannot be known. In the extreme case, when $\alpha = 1$, the median is always computed as the middle point of $g_x(v)$ or $g_y(v)$, i.e., no benefit is gained by data collection using LDP. On the other hand, when there are too many buckets, then the density of each bucket is low (LDP noise may overwhelm the true densities) but median computation is fine-grained since each bucket covers a small region. To better understand which factor dominates and to choose a good α value for practical use, it is valuable to experimentally study the utility impacts of α.

In Figure 6.2, we plot the MEs across varying α between 20 and 120 for different ε values (0.1, 0.5, 1, 2) and datasets. It can be observed from Figure 6.2 that MEs are usually quite low. We observe ME $\geq 4\%$ only when $\varepsilon = 0.1$, but for other ε,

MEs are generally $\leq 2\%$. Especially when ε is 1 or 2, we observe low MEs across all α values. This shows that our median computation method works well in practice, and enables accurate computation of the medians under LDP.

In the case of high ε values (such as 1 or 2), the impact of α on MEs is not very high. However, when privacy is stricter (such as $\varepsilon = 0.1$), MEs are relatively more impacted by the choice of α. In the low ε regimes, the change in ME across $\alpha \in [20, 120]$ resembles a U-shaped curve. When α is low (such as $\alpha = 20$), ME is high. This is because a low number of buckets causes median computation to be too coarse, and therefore even if our algorithm identifies the correct bucket, the precision of the median computation is negatively affected. On the other hand, when α is high (such as $\alpha = 100$ or 120), we observe that MEs can also be high. This is because few location points fall into each bucket, causing low bucket density. This hurts the correctness of identifying the right bucket for the median, which increases ME. Overall, to minimize error, our results and analyses show that instead of choosing α values that are too low or too high, it is preferable to choose values towards the middle, e.g., $\alpha = 60$ or 80.

6.6 Comparison Against Prior Work

In this section, we perform an experimental comparison between our quadtree solution (the "proposed" version), kd-tree solution (using the "improved" strategy), and a state-of-the-art work for answering spatial density queries under LDP [Tire and Gursoy, 2024], abbreviated as ASDQ-LDP. To perform the experimental comparison, we generate $N = 600$ random queries and compute their true answers and noisy answers under LDP using each of the three approaches (our quadtree solution, our kd-tree solution, and ASDQ-LDP [Tire and Gursoy, 2024]) separately.

In Figure 6.3, we plot the AQEs of each of the three approaches with different ε values (0.1, 0.5, 1, 2) and datasets. We observe that our quadtree and kd-tree solutions achieve substantially lower errors compared to ASDQ-LDP in all cases. The improvements can be up to 10-fold in some settings. We further observe that the errors of ASDQ-LDP are quite high when $\varepsilon = 0.1$ and 0.5, but improve when

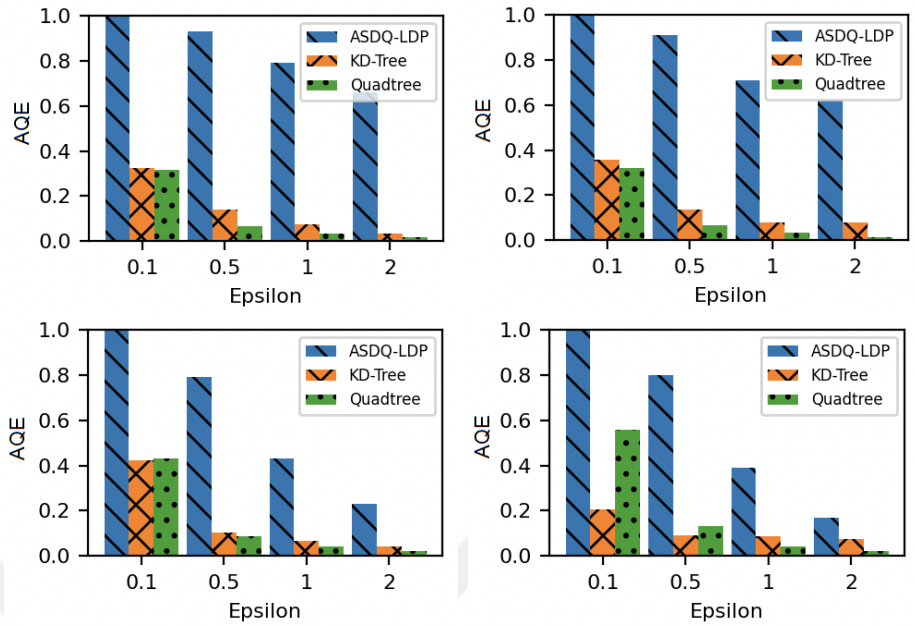

Figure 6.3: Errors of quadtrees, kd-trees and ASDQ-LDP [Tire and Gursoy, 2024] with varying ε and datasets (from left to right: Brightkite, Gowalla in the top row, Foursquare, Kaggle in the bottom row).

$\varepsilon = 1$ and 2. Our kd-tree and quadtree solutions also have relatively higher errors under $\varepsilon = 0.1$, but their errors remain low for $\varepsilon \geq 0.5$. These results from Figure 6.3 agree with the results in Figure 6.1, since we had also observed in Figure 6.1 that errors decrease substantially as we move from $\varepsilon = 0.1$ to $\varepsilon \geq 0.5$.

In addition to demonstrating the improvement achieved by our work compared to [Tire and Gursoy, 2024], Figure 6.3 also enables us to compare quadtrees and kd-trees with one another. When $\varepsilon = 0.1$, errors of quadtrees and kd-trees are usually similar (apart from the Kaggle dataset on which kd-trees achieve much lower error). For remaining ε values, errors of quadtrees are usually slightly lower compared to kd-trees. Quadtrees achieve particularly low errors of $\leq 2.5\%$ when $\varepsilon = 2$.

Chapter 7

CONCLUSION

In this thesis, we studied the problem of building hierarchical spatial decompositions under LDP, focusing on two popular types of decompositions: quadtrees and kd-trees. We proposed two solutions for quadtrees: a baseline solution that adapts an iterative strategy inspired by the centralized DP literature and a newly proposed solution that relies on a single LDP data collection step. Furthermore, we proposed a solution for building kd-trees which relies on a novel algorithm for learning medians from the user population in every depth of the kd-tree. Two variants of the kd-tree algorithm were developed based on their strategies for determining children nodes' densities ("initial" and "improved" strategies).

We experimentally evaluated all algorithms using four real-world datasets, several ε values, and metrics. Results showed the superiority of the proposed solution compared to the baseline solution for quadtrees and the superiority of the improved strategy over the initial strategy for kd-trees. We also validated that our median computation method enables accurate estimation of nodes' medians under LDP. Finally, we showed that both our quadtree and kd-tree solutions achieve substantial utility improvement in answering spatial density queries under LDP, compared to a state-of-the-art solution [Tire and Gursoy, 2024].

In future work, we plan to study other spatial decomposition methods, such as octrees, grid-based decompositions (e.g., adaptive grids), and hybrid decompositions under LDP. In addition, we plan to study the adaptation and application of our LDP median computation method in alternative domains and more general data analysis tasks.

BIBLIOGRAPHY

[Alptekin and Gursoy, 2023] Alptekin, E. and Gursoy, M. E. (2023). Building quadtrees for spatial data under local differential privacy. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 22–39. Springer.

[Apple, 2020] Apple (2020). Learning with privacy at scale. `https://machinelearning.apple.com/docs/learning-with-privacy-at-scale/appledifferentialprivacysystem.pdf`.

[Bagdasaryan et al., 2022] Bagdasaryan, E., Kairouz, P., Mellem, S., Gascón, A., Bonawitz, K., Estrin, D., and Gruteser, M. (2022). Towards sparse federated analytics: Location heatmaps under distributed differential privacy with secure aggregation. *Proceedings on Privacy Enhancing Technologies*, 4:162–182.

[Chen et al., 2012] Chen, R., Acs, G., and Castelluccia, C. (2012). Differentially private sequential data publication via variable-length n-grams. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, pages 638–649.

[Chen et al., 2016] Chen, R., Li, H., Qin, A., Kasiviswanathan, S. P., and Jin, H. (2016). Private spatial data aggregation in the local setting. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 289–300. IEEE.

[Cho et al., 2011] Cho, E., Myers, S. A., and Leskovec, J. (2011). 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*, page 1082–1090, New York, NY, USA. Association for Computing Machinery.

[Cormode et al., 2018] Cormode, G., Jha, S., Kulkarni, T., Li, N., Srivastava, D., and Wang, T. (2018). Privacy at scale: Local differential privacy in practice. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1655–1658. ACM.

[Cormode et al., 2012] Cormode, G., Procopiuc, C., Srivastava, D., Shen, E., and Yu, T. (2012). Differentially private spatial decompositions. In *28th IEEE International Conference on Data Engineering*, pages 20–31. IEEE.

[Cunningham et al., 2021] Cunningham, T., Cormode, G., Ferhatosmanoglu, H., and Srivastava, D. (2021). Real-world trajectory sharing with local differential privacy. *Proceedings of the VLDB Endowment*, 14(11):2283–2295.

[Ding et al., 2017] Ding, B., Kulkarni, J., and Yekhanin, S. (2017). Collecting telemetry data privately. In *Advances in Neural Information Processing Systems*, pages 3571–3580.

[Du et al., 2023] Du, Y., Hu, Y., Zhang, Z., Fang, Z., Chen, L., Zheng, B., and Gao, Y. (2023). Ldptrace: Locally differentially private trajectory synthesis. *Proceedings of the VLDB Endowment*, 16(8):1897–1909.

[Dwork et al., 2014] Dwork, C., Roth, A., et al. (2014). The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407.

[ECML/PKDD,] ECML/PKDD. Taxi trajectory prediction dataset.

[Erlingsson et al., 2014] Erlingsson, Ú., Pihur, V., and Korolova, A. (2014). Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1054–1067. ACM.

[Gursoy et al., 2022] Gursoy, M. E., Liu, L., Chow, K.-H., Truex, S., and Wei, W. (2022). An adversarial approach to protocol analysis and selection in local differen-

tial privacy. *IEEE Transactions on Information Forensics and Security*, 17:1785–1799.

[Gursoy et al., 2018a] Gursoy, M. E., Liu, L., Truex, S., and Yu, L. (2018a). Differentially private and utility preserving publication of trajectory data. *IEEE Transactions on Mobile Computing*, 18(10):2315–2329.

[Gursoy et al., 2018b] Gursoy, M. E., Liu, L., Truex, S., Yu, L., and Wei, W. (2018b). Utility-aware synthesis of differentially private and attack-resilient location traces. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 196–211.

[Gursoy et al., 2020] Gursoy, M. E., Rajasekar, V., and Liu, L. (2020). Utility-optimized synthesis of differentially private location traces. In *IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pages 30–39. IEEE.

[Gursoy et al., 2019] Gursoy, M. E., Tamersoy, A., Truex, S., Wei, W., and Liu, L. (2019). Secure and utility-aware data collection with condensed local differential privacy. *IEEE Transactions on Dependable and Secure Computing*.

[He et al., 2015] He, X., Cormode, G., Machanavajjhala, A., Procopiuc, C. M., and Srivastava, D. (2015). Dpt: differentially private trajectory synthesis using hierarchical reference systems. *Proceedings of the VLDB Endowment*, 8(11):1154–1165.

[Hong et al., 2021] Hong, D., Jung, W., and Shim, K. (2021). Collecting geospatial data with local differential privacy for personalized services. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 2237–2242.

[Kim et al., 2018] Kim, J. W., Kim, D.-H., and Jang, B. (2018). Application of local differential privacy to collection of indoor positioning data. *IEEE Access*, 6:4276–4286.

[Kuo et al., 2018] Kuo, Y. H., Chiu, C. C., Kifer, D., Hay, M., and Machanavajjhala, A. (2018). Differentially private hierarchical count-of-counts histograms. *Proceedings of the VLDB Endowment*, 11(11):1509–1521.

[Li et al., 2021] Li, S., Geng, Y., and Li, Y. (2021). A differentially private hybrid decomposition algorithm based on quad-tree. *Computers & Security*, 109:102384.

[Liu et al., 2022] Liu, G., Tang, Z., Wan, B., Li, Y., and Liu, Y. (2022). Differential privacy location data release based on quadtree in mobile edge computing. *Transactions on Emerging Telecommunications Technologies*, 33(6):e3972.

[Mokbel et al., 2006] Mokbel, M. F., Chow, C.-Y., and Aref, W. G. (2006). The new casper: Query processing for location services without compromising privacy. In *VLDB*, volume 6, pages 763–774.

[Navidan et al., 2022] Navidan, H., Moghtadaiee, V., Nazaran, N., and Alishahi, M. (2022). Hide me behind the noise: Local differential privacy for indoor location privacy. In *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 514–523. IEEE.

[Niknami et al., 2020] Niknami, N., Abadi, M., and Deldar, F. (2020). A fully spatial personalized differentially private mechanism to provide non-uniform privacy guarantees for spatial databases. *Information Systems*, 92:101526.

[Qardaji et al., 2013] Qardaji, W., Yang, W., and Li, N. (2013). Differentially private grids for geospatial data. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 757–768.

[Samet, 1984] Samet, H. (1984). The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260.

[Shaham et al., 2022] Shaham, S., Ghinita, G., Ahuja, R., Krumm, J., and Shahabi, C. (2022). Htf: Homogeneous tree framework for differentially-private release of

large geospatial datasets with self-tuning structure height. *ACM Transactions on Spatial Algorithms and Systems*.

[Thakurta et al., 2017] Thakurta, A. G., Vyrros, A. H., Vaishampayan, U. S., Kapoor, G., Freudinger, J., Prakash, V. V., Legendre, A., and Duplinsky, S. (2017). Emoji frequency detection and deep link frequency. US Patent App. 15/640,266.

[Tire and Gursoy, 2024] Tire, E. and Gursoy, M. E. (2024). Answering spatial density queries under local differential privacy. *IEEE Internet of Things Journal*.

[Wang et al., 2022] Wang, H., Hong, H., Xiong, L., Qin, Z., and Hong, Y. (2022). 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*, page 2809–2823, New York, NY, USA. Association for Computing Machinery.

[Wang et al., 2017] Wang, T., Blocki, J., Li, N., and Jha, S. (2017). Locally differentially private protocols for frequency estimation. In *Proc. of the 26th USENIX Security Symposium*, pages 729–745.

[Wang et al., 2018] Wang, T., Li, N., and Jha, S. (2018). Locally differentially private frequent itemset mining. In *IEEE Symposium on Security and Privacy (SP)*. IEEE.

[Xiao et al., 2010] Xiao, Y., Xiong, L., and Yuan, C. (2010). Differentially private data release through multidimensional partitioning. In *7th VLDB Workshop on Secure Data Management (SDM)*, pages 150–168. Springer.

[Xiong et al., 2020] Xiong, X., Liu, S., Li, D., Cai, Z., and Niu, X. (2020). A comprehensive survey on local differential privacy. *Security and Communication Networks*, 2020:1–29.

[Yan et al., 2020] Yan, Y., Gao, X., Mahmood, A., Feng, T., and Xie, P. (2020). Differential private spatial decomposition and location publishing based on unbalanced quadtree partition algorithm. *IEEE Access*, 8:104775–104787.

[Yang et al., 2015] Yang, D., Zhang, D., Zheng, V. W., and Yu, Z. (2015). Modeling user activity preference by leveraging user spatial temporal characteristics in lbsns. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(1):129–142.

[Yang et al., 2022] Yang, J., Cheng, X., Su, S., Sun, H., and Chen, C. (2022). Collecting individual trajectories under local differential privacy. In *2022 23rd IEEE International Conference on Mobile Data Management (MDM)*, pages 99–108. IEEE.

[Yang et al., 2020] Yang, M., Lyu, L., Zhao, J., Zhu, T., and Lam, K.-Y. (2020). Local differential privacy and its applications: A comprehensive survey. *arXiv preprint arXiv:2008.03686*.

[Ye et al., 2019] Ye, Q., Hu, H., Meng, X., and Zheng, H. (2019). Privkv: Key-value data collection with local differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 317–331. IEEE.

[Zhang et al., 2016] Zhang, J., Xiao, X., and Xie, X. (2016). Privtree: A differentially private algorithm for hierarchical decompositions. In *Proceedings of the 2016 International Conference on Management of Data*, pages 155–170.