Differential Privacy for Nearest Neighbor Queries

by

Emily Lepert

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2022

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

We examine the problem of providing differential privacy for nearest neighbor queries. Very few mechanisms exist that achieve this, most notable geo-indistinguishability in the context of location privacy. However it uses an extended definition of differential privacy and restricts the sensitivity of queries.

This work presents a new mechanism for DP nearest neighbor queries that is general to many applications and is based on tree data-structures and traversal. The biggest challenge with existing local differential private solutions is poor utility, requiring the addition of a restriction on the sensitivity of queries. We provide two variations, one which uses a similar restriction and one that does not. We explore different tree traversal algorithms. We evaluate our method on artificial datasets as well as real world location data. The results show that the variant using a restricted sensitivity does not perform better than geo-indistinguishability, while the unrestricted variant offers a method with good utility.

Acknowledgements

I want to thank my supervisor Florian Kerschbaum for his guidance with this project. His patience, encouragement, and expertise helped me grow as a researcher and as a person. I would also like to thank Simon Oya for being so patient with me and being willing to explain things to me multiple times. Despite the pandemic, my friends from the CrySP lab made me feel part of their community. I would especially like to thank Thomas, Rasoul, Jason, and Shannon for welcoming me to Canada and making my stay here a fun one. To all my friends, thank you for supporting me throughout the highs and lows, with special recognition to Ava for being my buddy throughout this journey. And finally to my family, thank you for everything.

This work benefitted from the use of the CrySP RIPPLE Facility at the University of Waterloo. All experiments were conducted using RIPPLE machines.

Dedication

To my loving and supportive grandfather.

Table of Contents

List of Figures

Chapter 1

Introduction

A key algorithm in data science is to solve the nearest neighbors problem. This consists of finding the nearest value to an input among a set of data points. The nearest neighbors problem is fundamental to numerous data applications such as machine learning and pattern recognition, spatial network problems and various other statistical problems [40]. However often times these applications use sensitive data. For example, imagine a hospital would like to find patients with similar symptoms to evaluate treatment outcomes. These patients might not want to be identified as part of this patient group. As such, much effort has been put into developing nearest neighbor algorithms that protect privacy.

An increasingly common method to ensure data privacy is *differential privacy* which essentially ensures that the output of an analysis done on a set of data is approximately the same whether any one data point is included or not [20]. It is a fairly well established notion with groups like Apple [13], Google [22], the US Census Bureau [36], and Microsoft [16] using it to ensure the privacy of user's data. Most of the work done in ensuring differential privacy in nearest neighbor algorithms focus on protecting the privacy of the data set the algorithm searches rather than the value the algorithm would like to find the neighbor of [27, 46, 41, 56]. Little to no research has been conducted to protect the individual querying the dataset as applications assume that this query is public.

A specific construction of differential privacy, local differential privacy, provides a path towards such a guarantee. This model assumes that the data owners do not trust a third party to aggregate their unprotected data and thus ensure differential privacy of their data themselves. This typically results in an overall loss of utility due to a large amount of noise being included in any aggregate analysis of such protected data. Combining local differential privacy with nearest neighbor algorithms would allow us to protect an individual query's privacy.

A natural application at the intersection of nearest neighbors and local differential privacy is location privacy. For example, imagine someone would like to know what restaurants are nearby without sharing their exact location with the location service. Geo-indistinguishability [2], which ensures differential privacy given a location radius, is, to

our knowledge, the only differentially private mechanism which protects an individual location querying a larger dataset. However it necessitates a high level of privacy which consequently translate to low utility [43].

In this thesis, we propose a new method for performing nearest neighbor searches while preserving the privacy of the querying data point. We do so by leveraging a common data structure used for nearest neighbors, *k-d trees*. These binary trees arrange multi-dimensional data in an easily searchable format [5, 25]. During a non-private nearest neighbor traversal of these trees, every node is compared to the querying value to determine which direction the traversal will continue in. Our method ensures that these comparisons are done privately via the exponential mechanism [37]. This mechanism implements a differentially private selection by which all possible outcomes are evaluated using a utility metric, and one is selected using a probability distribution that maximizes the chances that the "best" element is chosen.

We evaluate the utility of our method and its variants on artificially generated data as well as real world datasets. We conduct an experimental evaluation against geo-indistinguishability. We demonstrate that a variant of our construction achieves usable utility with no restriction on a location radius.

The remainder of the thesis is organized as follows: In Chapter 2 we introduce differential privacy and nearest neighbor algorithms and datastructures. We define our problem and provide the security definition in Chapter 3. We identify relevant literature in Chapter 4. In Chapter 5 we detail our private nearest neighbor algorithm and its variants. Finally, in Chapter 6 we evaluate our solution against geo-indistinguishability.

Chapter 2

Preliminaries

2.1 Differential privacy

Differential privacy (DP) is a concept of privacy that is becoming more and more popular. It was first established by Dwork et al. [20] and allows for analysis of data while maintaining the privacy of the data owners. The guarantee provided by differential privacy is such that the output of an analysis done on a database is approximately the same whether a single user is in the database or not. Large organizations like Microsoft [16], Google [22], Apple [13], and the US Census Bureau [36] have all used differential privacy to ensure privacy of users or citizens.

A common use of differential privacy is the centralized model, where many users give their data to a trusted party. That party then does some analysis that ensures differential privacy and then shares the result of the analysis. We will now define (ϵ, δ)-differential privacy [21], an extension of ϵ-differential privacy [19].

Definition 2.1 *A randomized algorithm $\mathcal{M} : \mathcal{D} \mapsto \mathcal{R}$ is (ϵ, δ)-differentially private if for all neighboring datasets $D, D' \in \mathcal{D}$, and all $Y \subseteq \mathcal{R}$,*

$$\Pr[\mathcal{M}(D) \in Y] \leq e^{\epsilon} \Pr[\mathcal{M}(D') \in Y] + \delta$$

The ϵ parameter determines how private a mechanism $\mathcal{M}$ is, where a smaller value corresponds to a more private mechanism. The δ parameter accounts for the probability of failure of the guarantee. When $\delta = 0$, we can say that $\mathcal{M}$ is ϵ-differentially private. When $\delta \neq 0$, we say that $\mathcal{M}$ has approximate differential privacy. It is standard to consider cases where $\delta < 1/N$ where N is the size of the database [21]. We say two databases D and D' are neighboring if they differ in at most one data point. That is, one data point in D is replaced by another in D'.

Differential privacy includes this concept of neighboring databases within the definition itself. DP uses $Pr[\mathcal{M}(x) = Z] \leq e^{\epsilon d_h(x,x')} Pr[\mathcal{M}(x') = Z]$ as its definition where $d_h(x, x')$ is

the Hamming distance between neighboring datasets and is thus always 1 (so we exclude it when talking about traditional DP). As we will explain later, we can use a different distance metric for different variations on DP.

A very common method to achieve differential privacy is to add random noise to the output of a function. In order to do this, we need to bound the output of this function with the notion of sensitivity.

Definition 2.2 (Definition 3 [21]) *For $f : \mathcal{D} \mapsto \mathcal{R}$, the* L1-sensitivity *of f is*

$$\Delta f = \max_{D,D' \in \mathcal{D}} ||f(D) - f(D')||_1$$

for all D, D' differing in at most one element.

There are many noise distributions that can be added to a function to ensure differential privacy. A very common one is the Laplace distribution.

Definition 2.3 (Laplace Distribution [21]) *The Laplace Distribution (centered at 0) with scale b is the distribution with probability density function:*

$$\text{Lap}(x|b) = \frac{1}{2b}\exp(-\frac{|x|}{b})$$

As a shorthand we denote $\text{Lap}(b)$ to be a Laplace distribution with scale b. The differentially private mechansim that makes use of this distribution is the Laplace Mechanism [21].

Definition 2.4 (Laplace Mechanism [21]) *Given any function $f : \mathcal{D} \mapsto \mathcal{R}$, the Laplace mechanism is defined as:*

$$\mathcal{M}_L(x, f(\cdot), \epsilon) = f(x) + (Y_1, ..., Y_k)$$

where Y_i are independent and identically distributed random variables drawn from Lap($\Delta f/\epsilon$).

We specifically focus on a type of differential privacy called local differential privacy (LDP). In this model, the user does not trust any central party to handle their data and must ensure the privacy of their data themselves. They accomplish this by randomizing their data before a third party accesses it. Local DP is considered the strongest differential privacy construction. However it suffers from a loss of utility since every piece of data is randomized so any aggregate computation based off of this data will suffer from a large amount of noise. Oftentimes, large companies will use local differential privacy to collect information for web browsing information (Google's RAPPOR system [22]) or usage and typing history (Apple [13]). These are all applications where a large amount of data is accumulated, so the influence of all the noise is reduced when looking at large trends.

Along with local differential privacy, we are interested in differentially private selection. One mechanism that provides a DP method for selection is the exponential mechanism [37]. We now define the exponential mechanism.

2.1.1 Exponential Mechanism

Introduced by McSherry and Talwar [37], the exponential mechanism is a commonly used method for differentially private selection. A main component of the mechanism is a utility function for which we need to calculate its sensitivity. Given a set of candidate items $\mathcal{H}$ and a utility function $u : \mathcal{D} \times \mathcal{H} \mapsto R$, the goal is for the mechanism is to output an item $h \in \mathcal{H}$ such that $u(D, h)$ is maximized while preserving differential privacy. The sensitivity of such a function is

Definition 2.5 ([21]) *The sensitivity of a utility function $u : \mathcal{D} \times \mathcal{H} \mapsto R$ is:*

$$\Delta^{(u)} = \max_{h \in \mathcal{H}} \max_{D, D' \in \mathcal{D}} |u(D, h) - u(D', h)|$$

Definition 2.6 (Exponential mechanism [37]) *For any utility function $u(D, h)$, the* exponential mechanism *outputs $h \in \mathcal{H}$ with probability:*

$$Pr[h] = \frac{exp\left(\frac{eu(D,h)}{2\Delta^{(u)}}\right)}{\sum_{i \in \mathcal{H}} exp\left(\frac{eu(D,i)}{2\Delta^{(u)}}\right)}$$

where $\Delta^{(u)}$ is the global sensitivity of $u(D, h)$.

Lemma 2.7 (Theorem 6 [37]) The exponential mechanism guarantees ϵ-differential privacy.

2.1.2 Post processing

Often times, once a value has been calculated via some DP mechanism, more computations are made using it. For example, say that many users share a differentially private count of how many hours they play a game with the game publisher. That central party then sums all those values to share with the public how many hours their game has been played. Because each individual value was differentially private, that summed value is also private.

Definition 2.8 (Post-Processing [21]) *Let $\mathcal{M} : \mathcal{X}^n \to \mathcal{Y}$ be a (ϵ, δ)-differentially private mechanism. Let $f : \mathcal{Y} \to \mathcal{Z}$ be an arbitrary randomized mapping. Then $f \circ \mathcal{M}$ is (ϵ, δ)-differentially private.*

2.1.3 Composition

We can use a DP mechanism multiple times, resulting in the privacy budget increasing. We can compute the final privacy budget via composition.

Theorem 2.9 (Basic composition [21]) *Given $\mathcal{M} = (\mathcal{M}_1, ..., \mathcal{M}_k)$, a series of ϵ-differentially private algorithms, their composition $\mathcal{M}_{1,...,k}(x) = (\mathcal{M}_1(x), ..., \mathcal{M}_k(x))$ is $k\epsilon$-differentially private.*

This is not a particularly tight bound on ϵ, so we next look at advanced composition, introduced by Dwork and Roth [21].

Theorem 2.10 (Advanced composition [21]) *For all $\epsilon, \delta, \delta' \geq 0$, the class of (ϵ, δ)-differentially private mechanisms satisfies $(\epsilon', k\delta + \delta')$-differentially privacy under k-fold adaptive composition for:*

$$\epsilon' = \sqrt{2kln(1/\delta')}\epsilon + k\epsilon(e^{\epsilon} - 1)$$

Advanced composition is applicable to any (ϵ, δ)-differentially private mechanism. For this research we use the exponential mechanism, so we can use composition techniques that result in an even tighter bound on ϵ. Specifically we can use bounded range composition, introduced by Durfee and Rogers [18].

Definition 2.11 (Range-Bounded [18]) *Given a mechanism $\mathcal{M}$ that takes a collection of records in X to outcome set Y, we say that $\mathcal{M}$ is ϵ-range-bounded if for any $y, y' \in \mathcal{Y}$ and any neighboring databases $\mathbf{x}, \mathbf{x}'$ we have:*

$$\frac{\Pr[\mathcal{M}(\mathbf{x}) = y]}{\Pr[\mathcal{M}(\mathbf{x}') = y]} \leq e^{\epsilon} \frac{\Pr[\mathcal{M}(\mathbf{x}) = y']}{\Pr[\mathcal{M}(\mathbf{x}') = y']}$$

where we use the probability density function instead for continuous outcome spaces.

Lemma 2.12 (Lemma 4.3 [18]) *The exponential mechanism (Definition 2.6) is ϵ-BR.*

Durfee and Rogers [18] proved a tighter composition bound using this form of DP. This bound was later improved upon by Dong et al. [17] by computing the supremum of the KL divergence.

Theorem 2.13 (Proposition 4 [17]) *If mechanisms $\mathcal{M}_i$ are ϵ_i-BR for $i = 1, 2, ..., c$ then their n-fold adaptive composition is (ϵ, δ)-DP with*

$$\epsilon = \min \left\{ \sum_{i=1}^{c} \epsilon_i, \sum_{i=1}^{c} \left(\frac{\epsilon_i}{1 - e^{-\epsilon_i}} - 1 - \ln \left(\frac{\epsilon_i}{1 - e^{-\epsilon_i}} \right) \right) + \sqrt{\frac{1}{2} \sum_{i=1}^{c} \epsilon_i^2 \ln(1/\delta)} \right\}$$

2.2 Geo-indistinguishability

The focus of this research is on differentially private nearest neighbor searches. As we will describe in Chapter 3, a major application for this is private location searches. In this scenario, a user would like to use their location to retrieve some information from a database. For example, someone in Toronto would like to look up nearby restaurants. However, they do not want to reveal their exact location to the database owner. Instead of revealing their exact position x, they reveal an approximate location z. They do so by adding noise to their location, similar to the Laplace mechanism. In fact, adding noise in two dimensions (2D-LDP), is an effective way of ensuring the privacy of a location.

What does location privacy really mean? Does a user want to ensure that their data is private within the range of their neighborhood or rather their entire city? The range within which to protect a user's privacy can vary depending on use case and location. This is where *geo-indistinguishability* (geo-ind) comes in [2]. Under this scheme, the level of privacy is evaluated within a radius. We say that a user has ϵ^**-privacy within* r if any two locations separated by at most r produce analyses with "similar" distributions where the "level of similarity" depends on ϵ^* [2]. ϵ^* represents the level of privacy for a radius where the smaller ϵ^* is the more private it is.

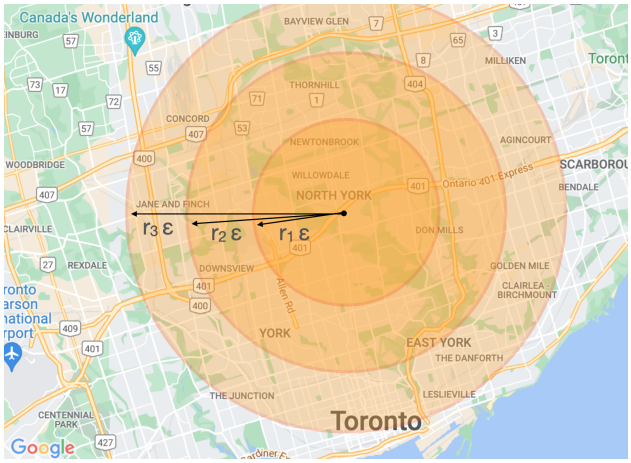


Figure 2.1: Example of geo-indistinguishability

Definition 2.14 ([2]) *A mechanism satisfies ϵ-geo-indistinguishability (ϵ-geo-ind) iff for any radius $r > 0$, the user enjoys ϵr-privacy within r.*

We define $\epsilon = \frac{\epsilon^*}{r}$, where ϵ^* is the privacy level, r is the privacy radius, and ϵ is the privacy parameter that is used in DP protocols. Notice that the level of privacy is inversely proportional to what the radius r and privacy value ϵ are. That is, say we want to achieve the same privacy level ϵ^* for two different radii $r_1 > r_2$. This results in $\epsilon_1 = \frac{\epsilon^*}{r_1}$ and $\epsilon_2 = \frac{\epsilon^*}{r_2}$. For ϵ_1, since r_1 is big, we have the same privacy level ϵ^*, but within a large radius. For ϵ_2,

since r_2 is smaller, we have the same privacy level ϵ^*, but within a small radius. The actual privacy parameter used in the protocol ϵ_1, ϵ_2 are different to reflect these guarantees. The user is always protected no matter what r is, however, the level $\epsilon^* = \epsilon r$ of privacy changes.

Definition 2.15 (Geo-Indistinguishability [2]) *A mechanism $\mathcal{M}$ satisfies ϵ-geo-indistinguishability iff for all x, x'*

$$\Pr[\mathcal{M}(x) = Z] \leq e^{\epsilon d(x,x')} \Pr[\mathcal{M}(x') = Z]$$

where $d(x, x') \leq r$ is the Euclidian distance between x and x'.

Geo-indistinguishability essentially means that the distance between the distributions of the output of $\mathcal{M}(x)$ and $\mathcal{M}(x')$ should be at most $\epsilon^* = \epsilon d(x, x')$. Notice the similarity between geo-indistinguishability and differential privacy. In fact geo-indistinguishability is a generalized variant of DP that uses a specialized metric between datasets (in this case Euclidian distance instead of Hamming distance). This is known as *extended differential privacy* [9] and can be considered as a relaxation of local differential privacy.

It is important to note that when discussing ϵ in geo-indistinguishability, we are assigning it a unit of measurement. ϵ is the privacy assigned to one unit of measurement and is affected by what that unit is. Since r is a physical quantity with some unit of measurement, ϵ must have the inverse of that unit of measurement in order to fulfill $\epsilon^* = \epsilon r$. If distances are in meters and $\epsilon^* = 0.1$, then the privacy parameter ϵ for points a kilometer away is $\epsilon^*/1000m = 0.0001m^{-1}$. If we now make the unit of measurement kilometers, ϵ must now be $0.1km^{-1}$ to keep the same level of privacy [2].

Andrés et al. [2] describe how to implement geo-indistinguishability and we direct the reader to their paper for further information. We implement geo-indistinguishability in Chapter 6 to compare our method to it.

2.3 Tree data-structures

The algorithms described in this research are based off of tree data-structures. Specifically we work with Binary Search Trees (BST) and k-d trees [5]. In this section, we will describe what each of these data-structures are.

2.3.1 Balanced Binary Search Trees

A binary search tree is simply a data structure representing a binary search of a sorted 1-dimensional array. They are a well established data-structure [34].

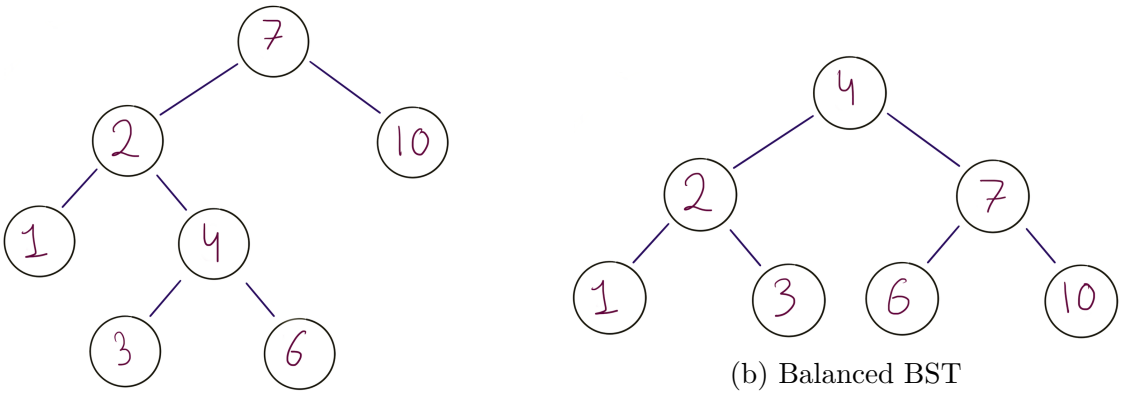
Definition 2.16 (Binary Search Tree [47]) *A* binary search tree *(BST) is a binary tree where each node has a comparable key (and an associated value) and satisfies the restriction that the key in any node is larger than the keys in all nodes in that node's left sub-tree and smaller than the keys in all nodes in that node's right sub-tree.*

We describe the insert procedure for a BST with Algorithm 1.

Algorithm 1 Insert operation for BST

Inputs: *tree* - Root node of existing BST
k - Value to be added

```
1:  function INSERT(tree, k)
2:      if tree is empty then return CREATENODE(k)
3:      CurrNode ← tree
4:      if k == CurrNode.key then return CurrNode          ▷ No duplicates
5:      if k < CurrNode.key then                           ▷ Key goes in left sub-tree
6:          if CurrNode.left == Null then
7:              CurrNode.left ← CREATENODE(k)
8:          else
9:              INSERT(CurrNode.left, k)
10:     else                                               ▷ Key goes in right sub-tree
11:         if CurrNode.right == Null then
12:             CurrNode.right ← CREATENODE(k)
13:         else
14:             INSERT(CurrNode.right, k)
```



(a) Unbalanced BST

(b) Balanced BST

Figure 2.2: Example of unbalanced and balanced binary search tree

We explain in detail the nearest neighbor search procedure of a BST in Section 2.5.1. Since the purpose of using a BST is to perform a search operation, it is advantageous for us

to specifically use *balanced* binary search trees (Figure 2.2). Such trees balance themselves after every insert or delete operation to minimize the height of the tree. This is done by having the parent node of two sub-trees be the median element of all the elements of the sub-trees. The height of a balanced BST is $log_2 N$ where N is the number of elements in the tree. As such the search algorithm has $O(log_2 N)$ operations which is quite efficient for a search algorithm.

2.3.2 K-d trees

A k-d tree is a multi-dimensional extension of a binary search. In fact, a binary search tree is essentially a 1-dimensional k-d tree. A k-d tree organizes k dimensional data into a tree. To describe the construction of a k-d tree we will first define some notation as first explained by Bentley [5].

1. Each node has an associated "discriminator" which is an integer between 0 and $k-1$.
2. The k keys of a node G are referred to as $K_0(G), ..., K_{k-1}(G)$.
3. The left sub-tree of a node G is $LEFT(G)$ and the right sub-tree is $RIGHT(G)$. The discriminator for G is $DISC(G)$.

The properties of a k-d tree are as follows:

1. For any node G and $j = DISC(G)$, then for any node $L \in LEFT(G)$, $K_j(L) < K_j(G)$.
2. Similarly, for any node $R \in RIGHT(G)$, $K_j(R) > K_j(G)$.
3. All nodes on the same level of the tree have the same discriminator. The root node has a discriminator of 0, the root's children have 1, and so on until $k-1$ is reached. At that point the discriminators cycle back to 0. We can define NextDisc(i) $= i+1 \pmod k$ where NextDisc(DISC(P))= $DISC(RIGHT(P))$ and $DISC(LEFT(P))$

Fig 2.3 shows an example of a 2-d tree. We can also represent it in a planar form (Fig 2.4). Notice that nodes B, D, E, G are all to the left of A in the planar example. This is because their 0th elements are smaller than A's 0th element. Using the planar representation we can see how k-d trees partition a space efficiently.

When constructing the tree, we are adding node Q and are evaluating node P where $j = DISC(P)$ and would like to determine what is Successor(P, Q), a function that determines which, if any, of P's children to move on to. If $K_j(Q) \neq K_j(P)$ then we compare $K_j(Q)$ and $K_j(P)$ and return either $LEFT(P)$ or $RIGHT(P)$ depending on the

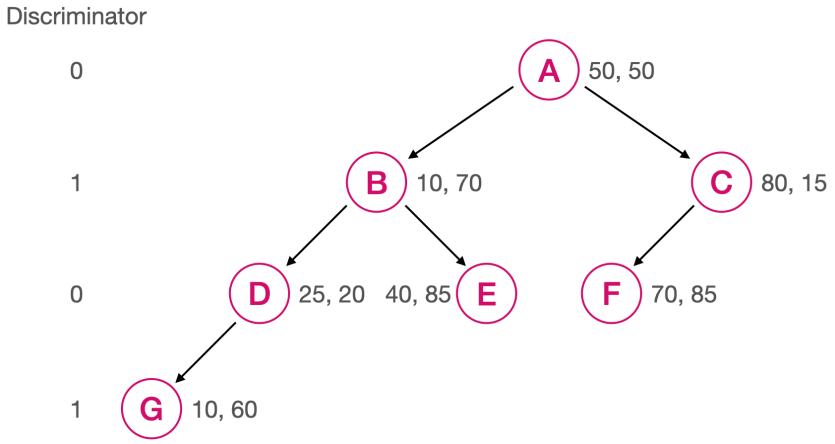


Figure 2.3: Tree representation of k-d tree

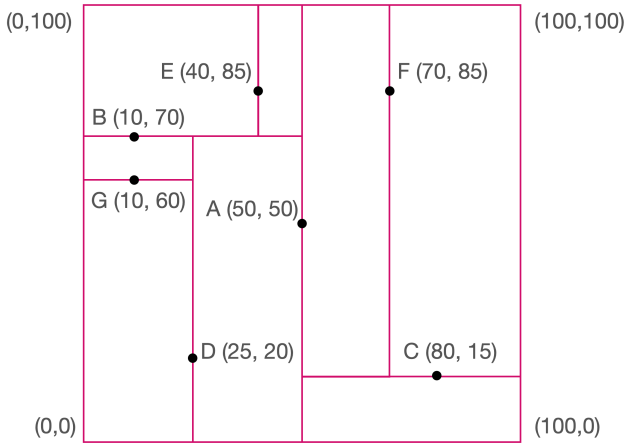


Figure 2.4: Planar representation of k-d tree

result. If $K_j(Q) = K_j(P)$, then both keys are equal and a special procedure must occur. A "superkey" of P is defined as

$$S_j(P) = K_j(P)K_{j+1}(P)...K_{k-1}(P)K_0(P)...K_{j-1}(P)$$

If $S_j(Q) < S_j(P)$ then SUCCESSOR(P, Q)= $LEFT(P)$, and if $S_j(Q) > S_j(P)$ then SUCCESSOR(P, Q)= $RIGHT(P)$. If $S_j(Q) = S_j(P)$ are equal then Q is not added to the tree as we do not allow duplicates.

Now that we have stated the properties of a k-d tree, we present the insert procedure in Algorithm 2.

Algorithm 2 Insert operation for k-d tree

Inputs: *root* - Root node of existing k-d tree
Q - Node to be added

```
1: function INSERT(Q, root)
2:     if root is empty then                          ▷ Check for empty tree
3:         Q.left, Q.right ← Null, Null
4:         Q.disc ← 0
5:         return Q
6:     else
7:         P ← root
8:     notEqual ← False
9:     for 0 ≤ i ≤ k − 1 do                           ▷ Check for duplicates
10:        if K_i(P) ≠ K_i(Q) then
11:            notEqual ← True
12:    if notEqual then
13:        ▽ Successor returns either the left or right child
14:        ChildNode ← SUCCESSOR(P, Q)
15:        if ChildNode = Nulll then
16:            Q.right, Q.left ← Null, Null
17:            Q.disc ← NEXTDISC(Disc(P))
18:            P.ChildNode ← Q
19:        else
20:            P ← ChildNode
21:            return INSERT(Q,P)
22:    else
23:        return P
```

The disadvantage of this construction is that it is not optimized for searching since the worst case height of a k-d tree is N, the total number of elements. Similarly to the BST we want to work with a balanced k-d tree. Friedman et al. [25] built upon this construction to define a balanced k-d tree by ensuring that equal numbers of nodes are found in both subtrees of any node. We use this construction for our analysis. In fact, our set up entails a party building a k-d tree from an array of k dimensional values. Algorithm 3 demonstrates how to build a balanced k-d tree from an existing array.

Algorithm 3 Build k-d tree from array

```
Inputs: P - List of values
k - Dimension of elements
d - Discriminator
1: function CREATE(P, k, d)
2:     if P is empty then
3:         ▽ Creates a node object. First element is the value, 2nd and 3rd are the left
   and right children, and last is the discriminator for the node
4:         return CREATENODE(Null, Null, Null, d)
5:     SORT(P, d)                          ▷ Sorts P by the dth element of every value
```

```
    median ← ⌊len(P)/2⌋
    loc ← P[median]
    left ← CREATE(P[:median], k, d + 1 (mod k))
    right ← CREATE(P[median+1:], k, d + 1 (mod k))
    return CREATENODE(loc, left, right, d)
```

We explain in detail the search procedure for k-d trees in Section 2.5.2. The benefits of k-d trees are mostly felt when k (the number of dimensions) remains relatively small with respect to the height of the tree. When k approaches N (the number of elements), the search becomes near linear. Generally k-d trees are advantageous when $N >> 2^k$[26].

Different types of neighbors We take this opportunity to discuss two different definitions of neighbors. The first is the more standard concept of neighbors, the values that are closest to one another. For example, in Figure 2.4 the neighbors of G are B, A, and E. We will call this definition the *true neighbors* of a value. Notice that these nodes are not directly next to G in Figure 2.3. The second concept refers to the more literal neighbors of a node in a tree. We call this the *tree neighbors* of a node. For example, the tree neighbors of G might be D, E, and B (depending on how far away we are looking). Notice that true neighbors and tree neighbors may overlap but are not necessarily the same.

2.4 Nearest neighbors (NN)

The nearest neighbors problem has been extensively explored and is fundamental to numerous applications such as spatial network problems (geographical information systems), artificial intelligence and pattern recognition, outlier detection, and various statistical problems [40]. In its most basic form, the problem is as follows: given a set of points $\mathcal{P}$ and a query point q, find the closest point in $\mathcal{P}$ to the query point q.

Algorithm 4 Brute force NN algorithm

```
1: currentNN ← 0
2: for 0 ≤ i < n do
3:     d ← COMPUTEDISTANCE(P[i], q)
4:     if d < d_m then
5:         d_m ← d
6:         currentNN ← P[i]
7: return currentNN
```

Given a set of n points $\mathcal{P}$, a query point q, and an initial minimum distance $d_m = \infty$, the simplest algorithm to do this is Algorithm 4.

Variations on the nearest neighbor problem exist. The k-nearest-neighbor (kNN) problem involves finding the k nearest objects to the query point. The reverse-nearest-neighbor (rNN) problem involves finding all points in a dataset for which q is their nearest neighbor. [40]

2.5 Tree based NN algorithms

In this chapter, we discuss nearest neighbor search algorithms involving trees in one and multiple dimensions. The simplest nearest search algorithms involve some form of a binary search. Binary trees (Section 2.3) are an intuitive way of representing data for a binary search.

2.5.1 1-Dimensional Data

A basic example of a nearest neighbor algorithm involves a 1-dimensional array of data. The data are organized into a binary search tree (Section 2.3.1) where parent nodes are the medians of their sub-trees and smaller values are placed in the left sub-tree with bigger ones in the right sub-tree.

The nearest neighbor algorithm (Algorithm 5) starts at the root of the tree and calculates the distance from the node to the search value. In the 1-dimension case this is the absolute value of the difference between both values. A record of the closest node found so far is kept. The search continues by traversing to the children of the parent node. If the search value is greater than the parent, then the next node considered is the right child. If the search value is smaller, then the left child is considered. The distance between the chosen child and the search value is calculated. If it is smaller than any previous distance found, then the child is saved as the nearest neighbor found so far. The search continues as described until it arrives at a leaf. At that point the nearest neighbor found along the way is the true nearest neighbor for the data. If at any point, the distance between a node and the search value is 0, then the algorithm stops and returns that node.

Algorithm 5 1D Nearest neighbor search

Inputs: *data* - Input array of data to search through
Q - Value to find nearest neighbor of

```
1: function 1DGetClosest(data, Q)
2:     P ← CreateBinarySearchTree(data)        ▷ Returns the root node of a BST
3:     bestDistance ← MAXINT
4:     bestNN ← MAXINT
5:     while P.isLeaf == False do
6:         distance ← |Q − P.data|
7:         if distance < bestDistance then
```

```
            bestDistance ← distance
            bestNN ← P.data
        if Q < P.data then
            P ← P.leftChild
        else
            P ← P.rightChild
    return bestNN, bestDistance
```

There are many variations on the nearest neighbor algorithm. As mentioned in Chapter 2, finding approximate nearest neighbours or k-nearest neighbors are common use cases. A 1-dimensional binary search tree is also a k-d tree of 1-dimension.

2.5.2 K Dimensional Data

Oftentimes nearest neighbor searches are done on data that have more than 1-dimension. There are many NN algorithms for data with k dimensions optimized for different use cases. Approximate nearest neighbors and k-nearest neighbors are often used in machine learning and computer vision applications [48]. We specifically focus on using *k-d trees* as they are a simple way of representing k dimensional data for ease of search.

Algorithm 6 K-d Tree Nearest neighbor search

```
    Inputs: P - Root node of k-d tree to search through
    Q - Value to find nearest neighbor of
    d - Tree depth
    k - Dimension of data
function KDGetClosestPoint(P, Q, d, k)
    if P is None then
        return None
    axis ← d (mod k)
    ▽ Check which direction is most promising
    if Q[axis] < P.data[axis] then
        subtree ← P.left
        oppositeSubtree ← P.right
    else
        subtree ← P.right
        oppositeSubtree ← P.left
    ▽ Recursively get the best point from the subtree
    subtreeBest ← KDGetClosestPoint(Q, subtree, d + 1, k)
    best ← ChooseClosest(Q, subtreeBest, P.data, k)
    bestDistance ← GetDistance(Q, best, d)
    distanceToSection ← |Q[axis] − P.data[axis]|
```

17: ▽ Check whether unexplored sub-tree is worth a visit
18: **if** *bestDistance* > *distanceToSection* **then**
19: subtreeBest ← KDGetClosestPoint(Q, oppositeSubtree, $d+1, k$)
20: best ← ChooseClosest(Q, subtreeBest, best, k)
21: **return** best

To find a nearest neighbor using a k-d tree, a similar traversal as the 1-dimensional case is done with an additional backtracking portion added (Algorithm 6). The root is visited, with the distance between the search value and the root being recorded. The distance metric used depends on use cases, but for location data, the Euclidian distance is used. Using whichever discriminator was used to build that level of the tree, the node is compared to the search value. Depending on the result of the comparison, the traversal will continue to the left or right until a leaf is found while keeping track of the best node found (Lines 5-14). Once a leaf is reached, the backtracking process begins.

At every level, the difference between the search value and the node at the appropriate discriminator for that tree depth is calculated (Line 16). If this distance is smaller than the best distance found in the sub-tree, then that implies that there is a chance that the opposite sub-tree (the one that was not chosen originally) has a point that is closer and is worth a visit. The algorithm looks at that sub-tree using the same process and calculates the best node in that sub-tree. Then the best node in this new sub-tree is compared to the original best node found in the original sub-tree and the best of the two is chosen. This process of checking the sub-tree not originally visited is repeated up until the root of the tree is visited. At that point the algorithm releases the best node found overall.

The worst case complexity of the nearest neighbor search using k-d trees is $O(N)$ as we may need to search the whole tree. However it has an average runtime of $O(\log N)$.

Algorithm 7 Helper for 6. Get Euclidian distance

Inputs: p_1 - Node
p_2 - Node
k - Dimension of data
1: **function** GetDistance(p_1, p_2, k)
2: preSqrDistance ← 0
3: **for** $i = 0$ k **do**
4: preSqrDistance ← $preSqrDistance + (p_1[i] - p_2[i])^2$
5: **return** $\sqrt{preSqrDistance}$

Algorithm 8 Helper for 6. Get closest point between two choices

Inputs: *point* - Value to get closest node of
p_1 - Node
p_2 - Node
k - Dimension of data

```
1: function ChooseClosest(point, p_1, p_2, k)
2:     d_1 ← GetDistance(point, p_1, k)
3:     d_2 ← GetDistance(point, p_2, k)
4:     if d_1 ≤ d_2 then
5:         return p_1
6:     else
7:         return p_2
```

Chapter 3

Problem statement

In this chapter we will describe scenarios where a differentially private nearest neighbors search would be useful. In all of these scenarios we assume a client with some kind of data they would like to keep private and a server with a dataset that the client will query. We will also describe the security protocol we need.

3.1 Use cases

Location privacy Perhaps the most obvious use case is in location privacy [35, 3]. In this scenario a user would like to use their position x to query nearby locations from the server. For example, someone in Toronto might want to know what restaurants are nearby while keeping their exact location private. For this use case data have two dimensions. Geo-indistinguishability is one mechanism that achieves this.

Time A one dimensional use case involves events. Say a user has an event that occurs at time x and would like to know what other events are occurring adjacent to their event. This is of particular interest in the field of epidemiology where public health officials would like to identify clusters of disease occurring at the same time [8].

Encodings While the focus of this research is on systems in low dimensions, possible future applications could be nearest neighbor searches for encodings. For example, a complex object like a PDF document could be encoded in some fashion. The client asks the server to identify whether it is likely that their document is malicious. The server does this by finding the most similar documents to the client's and checking whether these documents are malicious to classify the client's document. A large area of application is bio-metrics like fingerprint matching [30].

3.2 Security definition

The communication between server and client is not the focus of this research. We will describe the security model we assume for the two party computation scheme in this section. We assume the server is honest but curious.

The ideal functionality (Algorithm 9) of the protocol is such that the client $\mathcal{C}$ inputs its value x. The server $\mathcal{S}$ inputs the tree $\mathcal{P}$ it creates from its data. Both the client and server give their inputs to a trusted third party (TTP) $\mathcal{T}$. This TTP computes the nearest neighbor x_n of the client value and sends both the nearest neighbor and the tree to the client. The server gets no output.

Algorithm 9 Tree traversal functionality $\mathcal{F}$

Parameters: Client $\mathcal{C}$ - has a value x
Server $\mathcal{S}$ - has a tree $\mathcal{P}$

1: **function** FUNCTIONALITY
2: $\mathcal{C}$ sends x to $\mathcal{T}$, $\mathcal{S}$ sends $\mathcal{P}$ to $\mathcal{T}$
3: $\mathcal{T}$ sends x_n and $\mathcal{P}$ to $\mathcal{C}$

Let Π be the protocol and $\mathcal{F}$ be the functionality Π implements. We claim that our protocol is secure by the indistinguishable computation differential privacy definition, a notion presented by Mironov et al. [38].

Definition 3.1 (IND-CDP-2PC [29]) *A 2-party protocol Π for computing function f satisfies $(\epsilon_A(\kappa), \epsilon_B(\kappa))$-indistinguishable computationally differential privacy (IND-CDP-2PC) if $VIEW_A^{\Pi}(D_A, \cdot)$ satisfies $\epsilon_B(\kappa)$-IND-CDP, i.e. for any probabilistic polynomial-time (in κ) adversary T, for any neighboring databases (D_B, D'_B) differing in a single row,*

$$Pr[T(VIEW_A^{\Pi}(D_A, D_B)) = 1] \leq e^{\epsilon} Pr[T(VIEW_A^{\Pi}(D_A, D'_B)) = 1] + \text{negl}(\kappa)$$

The real world implementation of this functionality is straightforward. The server gives its tree to the client and the client does all of the computation. The view of the server is just the tree. Practically, this would create a very large communication burden due to the fact that the server is sharing the entire database with the client. We implement the functionality via a more interactive protocol.

We execute the protocol Π as in Algorithm 11. During the protocol, the server sends individual nodes of the tree to the client, starting with the root. The client computes a bit indicating which subtree the server should consider next to the server. This bit is calculated based off of the client's data and the node value. For example, if the client's data is smaller than the node value, then the bit would indicate the left sub-tree. However, this bit is computed via the exponential mechanism, ensuring that the client's data is differentially private and introducing randomness. The server continues on to the next node indicated by this bit and repeats the process until a leaf is found.

For this protocol, the view of the server $VIEW_A^{\Pi}$ is the full tree as well as the messages between the server and client, in this case the indicator bit the client sends to the server. The views of the server are computationally indistinguishable (as defined by IND-CDP-2PC) from the view of the simulator of $\mathcal{F}$ since the client communicates a bit to the server that was generated via a differentially private mechanism. It is not sufficient for the server and client to use a 2-party protocol to traverse the server's tree; they must do so while ensuring differential privacy for the client's data.

Chapter 4

Related Work

4.1 Extended local differential privacy

Local differential privacy is the guarantee that an individual does not have to trust any third party with their non-private data. It is mainly used to collect a lot of data and extract some aggregate analysis from it [22, 13, 36]. To our knowledge very little work has been done in protecting a user's data when used to query information from a larger database. A relaxation of differential privacy, extended differential privacy [9], allows for more broad uses of distance metrics and improves the utility of differentialy private mechanisms.

The existing work done with extended differential privacy consists of work in the centralized model [31, 23] or aggregating data in the local model [53]. Weggenmann et al. [52] use extended differential privacy for directional data. Geo-indistinguishability [2] is the most common use of extended local differential privacy for the sake of querying a central database. The metric it uses is the Euclidian metric which limits its use to location-based queries. Fernandes et al. [24] propose a technique using angular distance which is optimal in high dimensional spaces.

4.2 Differentially private nearest neighbors

When discussing the nearest neighbors problem with differential privacy, the majority if not all of the work so far is focused on keeping the database differentially private. The search value is typically shared in plaintext and no effort is made to keep it private. To our knowledge, no nearest neighbors scheme exists which ensures differential privacy for the search value.

Gursoy et al. [27] propose a radius neighbors classifier that provides differential privacy to k-NN classifiers. Rauch et al. [46] specifically work on differentially private outlier detection using k-NN and data partitioning. Nearest neighbors classification is a large

component of machine learning research. Papernot et al. [41] demonstrated PATE, a general approach to providing DP guarantees for training data. Zhu et al. [56] present a DP k-NN method for computer vision.

4.3 Differentially private location data

Location-based services (LBSs) are very useful but also have a high potential of violating privacy [50]. To address privacy concerns, LBSs have attempted to protect users' privacy. However most of the proposed solutions are not optimal. Some have lower accuracy [42], others create a lot of network traffic (e.g., by creating dummy requests [49]), and others don't integrate well with LBSs [12]. Anonymization techniques like k-anonimity, l-diversity, and t-closeness have been applied to location data [39, 55] however these techniques assume that the background knowledge the adversary has is limited or known in advance.

Location generalization [14] combined with differential privacy shows promise for improving accuracy and privacy. Dewri [15] added differential privacy to k-anonymity, a mechanism that shares k values to a LBSs in an attempt to obfuscate the true location of the user.

Andres et al. [2] proposed the notion of geo-indistinguishability, a differentially private method of achieving location privacy. They specifically implement geo-indistinguishability using the Laplace mechanism. Primault et al. [43] evaluated geo-indistinguishability and demonstrated that only the highest levels of privacy successfully obfuscated a user's point of interests (POIs). However high levels of privacy result in low accuracy of results. Geo-indistinguishability alternatives using linear optimization [7], remap and coarse grid mechanisms [10], and a multi-step algorithm alongside spatial indexing [1] have also been proposed. For all these methods, there is a tradeoff between utility and computation cost.

Private spacial decomposition (PSD) [44, 28] is another technique for differential private publishing of location data. It entails using spatial histograms by partitioning data into cells and each cell keeps a record of how many data points are within it. PSD techniques can be classified under data-dependent [54, 44, 51, 32] and data-independent methods [45]. However PSD requires the use of a trusted server.

Of the available location differential private preserving methods, only geo-indistinguishability collects and publishes microdata [33]. PSD and local differential privacy utilize aggregate data. Geo-indistinguishability is the only technique to allow privacy-preserving location-based query processing.

Chapter 5

Differentially private tree-based NN algorithms

5.1 DP basic tree traversal (basic DP-TT)

Several modifications to the nearest neighbor algorithms are made in order to make the whole process differentially private. We will describe the process using the problem statement from Chapter 3. Recall that we have a client and server who would like to cooperatively find the client's nearest neighbors in the server's data. However the client would like its data to remain differentially private. We call the method described in this section *DP-Tree Traversal* or (basic) *DP-TT*.

Overview As a preface to this chapter, we describe the overview of node selection. During the non-private NN traversal, at every node a decision must be made about which sub-tree to consider next. To do this privately, we avoid using complex methods like oblivious indexing or private information retrieval by letting the server index in plain.

Our concept is that every decision made is differentially private with respect to the client. Specifically, the client communicates to the server what to do in a DP way. Since the server's privacy is not the concern in this method, the client does not have to share any of its data with the server. In fact, we use simple communication that implements our security protocol (Section 3.2) where the server sends relevant nodes to the client in plain. The client uses these nodes in a DP protocol (described below) and generates a bit that it shares with the server. This bit indicates which sub-tree to look at next. Since this bit is differentially private and is the only thing the client shares with the server, the entire process is DP while the server can plainly access its database.

5.1.1 Setup

Server. The server creates a modified k-d tree with its data. Recall that a 1-d tree is a binary search tree.

Algorithm 10 Build a modified k-d tree from array

Inputs: $\mathcal{P}$ - List of values
k - Dimension of elements
d - Discriminator

1: **function** CREATE($\mathcal{P}, k, d$)
2: **if** $\mathcal{P}$ is empty **then**
3: ▽ Creates a node object. First element is the value, 2nd and 3rd are the left and right children, and last is the discriminator for the node
4: **return** CREATENODE(Null, Null, Null, d)
5: **if** $len(\mathcal{P}) == 1$ **then**
6: **return** CREATENODE($\mathcal{P}$[0], Null, Null, d)
7: SORT($\mathcal{P}$, d) ▷ Sorts $\mathcal{P}$ by the dth element of every value
8: median $\leftarrow \lfloor len(\mathcal{P})/2 \rfloor$
9: loc $\leftarrow \mathcal{P}$[median]
10: left $\leftarrow$ CREATE($\mathcal{P}$[:median], k, $d+1 \pmod k$)
11: right $\leftarrow$ CREATE($\mathcal{P}$[median:], k, $d+1 \pmod k$)
12: **return** CREATENODE(loc, left, right, d)

The k-d tree is modified such that all data is found in the leaves of the tree. Algorithm 10 demonstrates how to build such a tree from a starting array. The inner nodes are still the medians of the sub-tree (meaning they are part of the data), but the data is not considered added to the tree until it is found on a leaf. A tree with N leaves has a height of $\log_2(N) + 1$.

There is no guarantee that all data is included in the inner nodes. In the non-private NN algorithm with a k-d tree, a record of the best node found so far is kept and released once all backtracking is completed. In our scheme, the server has no way of knowing which node is better than another (since any information regarding the client is kept private) and no such record can be kept. Thus once a leaf is reached, the server has no way of knowing whether to backtrack or which of the traversed nodes to reveal to the client. In fact, our private construction does not include backtracking. By ensuring that all data is found in leaves, the server uses reaching a leaf as the end point of the algorithm. It can release the leaf it found and be sure that all data is able to potentially be found.

Recall that we have two different concepts of neighbors: true and tree neighbors. Now that all data nodes are found in the leaves, we amend the tree neighbors to be only *leaves* close to a the relevant node.

Client. The client does need to do anything in particular to prepare for the protocol.

5.1.2 Process

The server evaluates the client's data x relative to the root of the tree using some utility metric. We consider two different utility functions. The first is a distance metric (Algorithm 13) which calculates the Euclidian distance between the client data and the current tree node's children. In k dimensions, say x is the client's data and h is either the left or right children's data.

$$u_1(x, h) = \sqrt{\sum_{i}^{k} (x_i, h_i)^2}$$

We use DP-TT-DIS to refer to the DP-TT algorithm using the distance metric. The second utility metric (Algorithm 14) uses a simple comparison metric that evaluates whether the client data x is greater than or smaller than the tree node. This comparison is done on the element h at the discriminator the k-d tree is built on at that tree depth.

$$u_2(x, h) = x[i] < h[i]$$

We use DP-TT-CMP to refer to the DP-TT algorithm using the comparison metric. Since both of these utility functions are calculated via a 2 party protocol, the result of these functions are kept secret from the server, so the server does not know the result. The client is allowed to see the result and uses the exponential mechanism [37] to determine which direction the algorithm should go in.

Tree traversal (Algorithm 11) Using the exponential mechanism and either utility metric, the client communicates to the server which sub-tree is should consider next. The server moves on to the appropriate node and repeats the process until it reaches at leaf. A that point the server shares the value of the leaf with the client.

Algorithm 11 Simple Tree Traversal Algorithm

Inputs: $\mathcal{P}$ - Root node of k-d tree
Q - Client value
ϵ - Privacy value
Δ - Sensitivity

1: **function** NNSEARCH($\mathcal{P}, Q, \epsilon, \Delta$)
2: **while** $\mathcal{P}$.isLeaf = False **do**
3: leftChild $\leftarrow \mathcal{P}$.left
4: rightChild $\leftarrow \mathcal{P}$.right
5: DPQuery $\leftarrow$ EMQUERY(Q, $\mathcal{P}$.data, leftChild.data, rightChild.data, Δ, ϵ)
6: **if** DPQuery = 0 **then**
7: $\mathcal{P} \leftarrow$ leftChild

8: **else if** DPQuery = 1 **then**
9: $\mathcal{P} \leftarrow$ rightChild
10: **return** $\mathcal{P}$

Algorithm 12 Exponential Mechanism (Distance)

Inputs: P - Parent value
Q - Client value
L - Left child value
R - Right child value
ϵ - Privacy value
Δ - Sensitivity

1: **function** EMQuery(Q, P, L, R, Δ, ϵ)
2: $L_u, R_u \leftarrow$ Utility(P, L, R, Q)
3: leftProb $\leftarrow exp(\frac{\epsilon * L_u}{2 * \Delta})$
4: rightProb $\leftarrow exp(\frac{\epsilon * R_u}{2 * \Delta})$
5: totalProb $\leftarrow$ leftProb + rightProb
6: $X = \begin{cases} 0 & \text{w.p. } \dfrac{leftProb}{totalProb} \\ 1 & \text{w.p. } \dfrac{rightProb}{totalProb} \end{cases}$
7: **return** X

Algorithm 13 Utility (Distance)

Inputs: P - Parent value
Q - Client value
L - Left child value
R - Right child value

1: **function** Utility(P, L, R, Q)
2: leftDis $\leftarrow$ EuclidianDistance(L, Q)
3: rightDis $\leftarrow$ EuclidianDistance(R, Q)
4: $\triangledown$ Differences are inverted since bigger distances are worse, but utility function rewards high numbers.
5: $L_u \leftarrow$ rightDis
6: $R_u \leftarrow$ leftDis
7: **return** L_u, R_u

Algorithm 14 Utility (Randomized Response)

Inputs: P - Parent value
Q - Client value
L - Left child value
R - Right child value

```
function UTILITY(P, L, R, Q)
    trueAns ← P < Q
    if trueAns == 1 then
        L_u ← 0
        R_u ← 1
    else
        L_u ← 1
        R_u ← 0
    return L_u, R_u
```

5.2 Privacy

We will now show that DP-TT is differentially private with respect to the client's data. As a reminder, the server's privacy is not considered as we do not restrict the client from making arbitrary number of queries which can reveal the entire tree. We operate in an open system where the client can make infinite queries.

As stated by Lemma 2.7, the exponential mechanism is ϵ-differentially private (ϵ-DP). We apply the exponential mechanism as many times as the tree considered is tall. At every level of the tree, a decision of which sub-tree to consider next must be made. Every decision consists of using the exponential mechanism.

We use Theorem 2.13 to calculate the overall ϵ of the mechanism. The number of compositions is equal to the height of the tree minus 1. The height of the tree is $\log_2(N)+1$ for a tree with N leaves (created from a database of N elements), so the total number of compositions is $\log_2(N)$. If all ϵ_i's are equal and we say that $c = \log_2(N)$, then this results in:

$$\epsilon = \min\left\{c\epsilon_i, \left(\frac{c\epsilon_i}{1-e^{-c\epsilon_i}} - 1 - \ln\left(\frac{c\epsilon_i}{1-e^{-c\epsilon_i}}\right)\right) + \sqrt{\frac{1}{2}c\epsilon_i^2\ln(1/\delta)}\right\}$$

5.3 Limitations

Beyond the randomness introduced by using differential privacy to select the sub-trees to traverse to, the modifications made to the traversal algorithms themselves reduce the starting accuracy. The backtracking process used by k-d trees is not used as it introduces further inaccuracy for a higher privacy budget.

Backtracking For DP-TT, backtracking is a double edged sword. On one hand, allowing the protocol to backtrack would necessitate additional protections to preserve differential privacy. In fact, backtracking would entail an additional two comparisons per node which

need to be executed via a DP method. The first is to check whether it is worth backtracking into the opposite sub-tree. If so, the second is to compare the best node in the sub-tree with the current best node. On the other hand, the hopes of backtracking are to improve the accuracy of the protocol.

However, backtracking is initiated when the sub-tree that was not visited shows promise relative to the best value found so far. The issue is that with DP-TT, there is no guarantee that the best value found so far is accurate. So we would be comparing a potentially inaccurate best distance with the backtracking distance, and then employing the exponential mechanism to choose between them which further introduces inaccuracy. At worst, the exponential mechanism would be used two more times per node than the basic method, which increases the final privacy budget. We will demonstrate in Chapter 6 that this trade-off is not favorable. We choose not to implement backtracking in DP-TT and instead opt to explore other methods to increase the accuracy.

5.3.1 Mitigating limitations

An inherent level of randomness is introduced when using a DP mechanism to decide which sub-tree to visit next. There is always a probability that the "wrong" sub-tree will be visited. In some cases this lowers accuracy, however in cases where the correct answer requires backtracking, this can be an advantage. For example, suppose that we have a k-d tree and in the non-private mechanism, the right sub-tree would have eventually been chosen as the optimal sub-tree via backtracking (meaning that initially the left sub-tree would have been considered optimal). Now imagine that in the private mechanism, the exponential mechanism resulted in the "wrong" sub-tree being selected, that is the un-noised answer would be to go the left sub-tree, but the right sub-tree is instead chosen. This is actually beneficial as it increases the likelihood that the true answer is found.

Further, to increase the likelihood that the correct answer is found, we introduce parallel traversal, splitting, early stopping, and releasing neighborhoods.

5.4 DP complex tree traversal (complex DP-TT)

In this section we will discuss modifications to basic DP-TT described in Section 5.1. All of these different algorithms are evaluated in Chapter 6.

As mentioned before, to increase the likelihood of outputting a correct answer, we can simply increase the number of values outputted. In this case, the server shares all of the values retrieved through the tree traversal process with the client.

The first way to do this is by splitting off the search threads. Instead of the search only looking at one sub-tree, the search splits off into two threads, each taking one sub-tree. Within this there are multiple ways to use this multiple thread strategy. We evaluate

several such mechanisms. The other way of increasing the number of values is by releasing neighborhoods.

For all of these mechanisms, recall that the number of compositions for one simple traversal of the tree is one fewer than the height of the tree.

Parallel traversal The first mechanism is splitting at the top a certain number of times s (Algorithm 15). The search threads do not use the exponential mechanism at the top of the tree and just split off into multiple threads. Once the requisite number of splits has occurred, the DP traversal starts, resulting in several parallel traversals. This increases the number of times composition occurs and increases the final privacy budget. If s splits occur at the top of the tree of height $\log_2 N + 1$, the total number of compositions c that occur is

$$c = (\log_2 N - s) * 2^s$$

We will refer to this variant as DP-TT PX where X is the number of splits at the top.

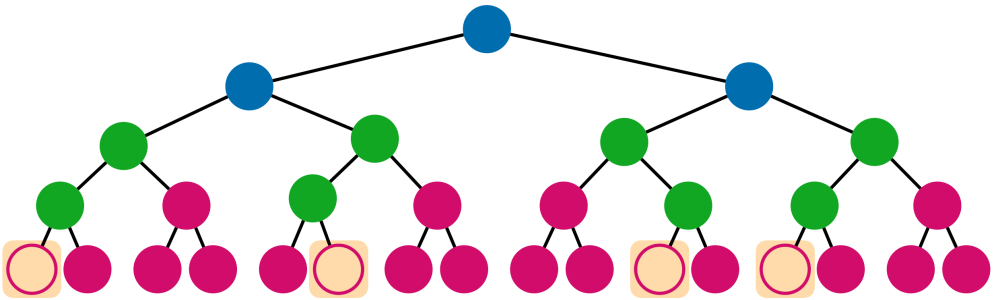


Figure 5.1: Example of parallel traversal (DP-TT P2). Green nodes indicate a use of the Exponential mechanism. Blue nodes indicate the search thread splitting. Highlighted nodes indicate nodes shared to the client.

Algorithm 15 Parallel Tree Traversal Algorithm DP-TT P

Inputs: $\mathcal{P}$ - Root node of k-d tree
Q - Client value
ϵ - Privacy value
Δ - Sensitivity
S - Number of splits remaining

1: **function** NNSearchParallel($\mathcal{P}, Q, \epsilon, \Delta, S$)
2: **while** $\mathcal{P}$.isLeaf = False **do**
3: **if** $S == 0$ **then**
4: leftChild $\leftarrow \mathcal{P}$.left
5: rightChild $\leftarrow \mathcal{P}$.right
6: DPQuery $\leftarrow$ EMQuery(Q, $\mathcal{P}$.data, leftChild.data, rightChild.data, Δ, ϵ)
7: **if** DPQuery = 0 **then**

```
            P ← leftChild
        else if DPQuery = 1 then
            P ← rightChild
    else
        L_R ← NNSearchSplitTop(P.left, Q, ε, Δ, S − 1)
        R_R ← NNSearchSplitTop(P.right, Q, ε, Δ, S − 1)
        return L_R + R_R
return P
```

Early stopping The second consists of one search thread stopping before it reaches a leaf, then splitting at the bottom of the tree (Algorithm 16). The DP traversal occurs and then a certain number of levels before the end of the tree, the search threads split off. This actually decreases the number of times composition occurs given that only one thread uses the exponential mechanism, and the exponential mechanism is not used when splitting occurs. If s splits occur at the bottom of the tree of height $\log_2 N + 1$, the total number of compositions c that occur is

$$c = \log_2 N - s$$

Note that the result of early stopping are a set of tree neighbors. That is, they are not necessarily nearest neighbors in the database, but in the tree they are neighbors. We will refer to this variant as DP-TT EX where X is the number of splits at the bottom.

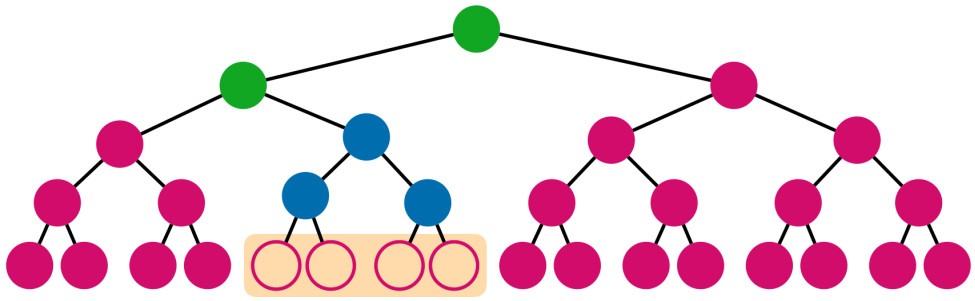


Figure 5.2: Example of early stopping traversal DP-TT E2. Green nodes indicate a use of the Exponential mechanism. Blue nodes indicate the search thread splitting. Highlighted nodes indicate nodes shared to the client.

Algorithm 16 Early Stopping Tree Traversal Algorithm DP-TT E

Inputs: $\mathcal{P}$ - Root node of k-d tree
Q - Client value
ϵ - Privacy value
Δ - Sensitivity
S - Number of splits remaining

```
1: function NNSearchEarlyStop(P, Q, ε, Δ, S)
2:     while P.isLeaf = False do
3:         if P.height > S then
4:             leftChild ← P.left
5:             rightChild ← P.right
6:             DPQuery ← EMQuery(Q, P.data, leftChild.data, rightChild.data, Δ, ε)
7:             if DPQuery = 0 then
8:                 P ← leftChild
9:             else if DPQuery = 1 then
10:                P ← rightChild
11:        else
12:            L_R ← NNSearchSplitBottom(P.left, Q, ε, Δ, S − 1)
13:            R_R ← NNSearchSplitBottom(P.right, Q, ε, Δ, S − 1)
14:            return L_R + R_R
15:    return P
```

Greedy splitting The third method uses the exponential mechanism at every level, but also splits off at every level (Algorithm 17). The exponential mechanism determines which sub-tree to continue to and the server creates two threads. One goes down the chosen path and will continue to split. The other goes down the sub-tree not chosen and will not continue to split. This results in as many outputs as the height of the tree minus 1 and increases the final ϵ value due to more compositions occurring. For a tree of height $\log_2 N + 1$, the total number of compositions c is

$$c = \sum_{k=1}^{\log_2 N} k = 1 + 2 + ... + \log_2 N = \frac{(\log_2 N)(\log_2 N + 1)}{2}$$

We will refer to this variant as DP-TT GS.

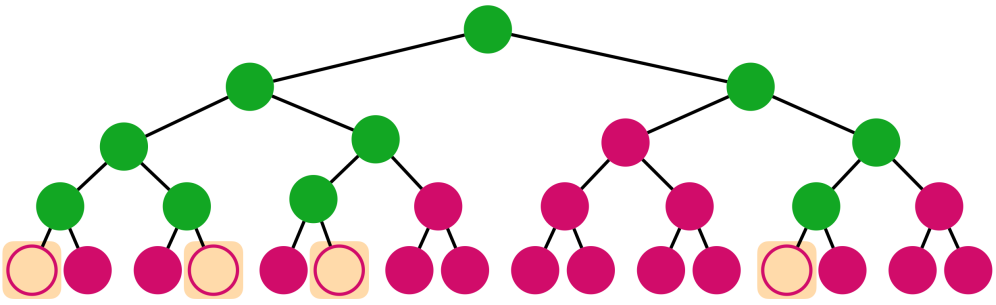


Figure 5.3: Example of splitting traversal (DP-TT GS). Green nodes indicate a use of the Exponential mechanism. Highlighted nodes indicate nodes shared to the client.

This method is essentially the implementation of backtracking assuming that every opposite sub-tree is worth investigating.

Algorithm 17 Greedy Splitting Tree Traversal Algorithm DP-TT GS

Inputs: $\mathcal{P}$ - Root node of k-d tree
Q - Client value
ϵ - Privacy value
Δ - Sensitivity
continueSplit - Indicates whether search thread should split

```
1:  function NNSearchGreedySplit(P, Q, ε, Δ, continueSplit)
2:      while P.isLeaf = False do
3:          leftChild ← P.left
4:          rightChild ← P.right
5:          DPQuery ← EMQuery(Q, P.data, leftChild.data, rightChild.data, Δ, ε)
6:          if continueSplit = False then
7:              if DPQuery = 0 then
8:                  P ← leftChild
9:              else if DPQuery = 1 then
10:                 P ← rightChild
11:         else
12:             if DPQuery = 0 then
13:                 L_R ← NNSearchSplitBottom(P.left, Q, ε, Δ, True)
14:                 R_R ← NNSearchSplitBottom(P.right, Q, ε, Δ, False)
15:                 return L_R + R_R
16:             else
17:                 L_R ← NNSearchSplitBottom(P.left, Q, ε, Δ, False)
18:                 R_R ← NNSearchSplitBottom(P.right, Q, ε, Δ, True)
19:                 return L_R + R_R
20:     return P
```

Neighborhoods The last way of increasing the number of values outputted is by releasing the found value as well as the B true neighbors of that found element (Algorithm 18). This is done by using the non-private nearest neighbors algorithm. This does not change the final ϵ value due to post-processing (Definition 2.8). We will refer to this variant as DP-TT NX where X is the number of values in a neighborhood.

Algorithm 18 Tree Traversal Algorithm w/ NN DP-TT N

Inputs: $\mathcal{P}$ - Root node of k-d tree
Q - Client value
ϵ - Privacy value
Δ - Sensitivity
B - Number of neighbors to the found result to release

```
1:  function KNNSearch(P, Q, ε, Δ, B)
2:      originalTree ← P
```

```
3:    while 𝒫.isLeaf = False do
4:        leftChild ← 𝒫.left
5:        rightChild ← 𝒫.right
6:        DPQuery ← EMQuery(Q, 𝒫.data, leftChild.data, rightChild.data, Δ, ϵ)
7:        if DPQuery = 0 then
8:            𝒫 ← leftChild
9:        else if DPQuery = 1 then
10:           𝒫 ← rightChild
11:   ▽ Finds the B nearest neighbors to 𝒫
12:   return KNN(𝒫, B, originalTree)
```

Finding the B nearest neighbors of the value found through the DP mechanism does not change the privacy guarantee. We evaluate all of these modifications in Chapter 6 and combine splitting with sharing neighborhoods.

Chapter 6

Experimental evaluation

6.1 Setup

The purpose of the experimental evaluation is to assess the feasibility and accuracy of the DP guarantee of DP-TT. As such, the specifics of the communication between the client and server are left to future work (Section 3.2). We assume that through some secure method, the server is able to communicate relevant node values to the client and the client can communicate indicator bits to the server. A simple way to implement this is a 2-party computation where both parties send values in plain to each other (since server privacy is not a concern).

We run experiments on artificially generated random data. We run DP-TT on 1-dimensional data, uniformly drawing 100,000 and 1,000,000 values sampled from a 0 to 1 billion normal distribution. We also run it on two dimensional data, uniformly drawing 100,000 and 1,000,000 pairs of values sampled from a 0 to 1 billion normal distribution. When discussing this data we do not specify a unit of measurement as any unit could be applied.

We also run the experiments on the real world datasets Gowalla and Brightkite [11]. We restrict the points of interest to two locations, San Francisco, California and Austin, Texas. For San Francisco, the specific region is between the latitude coordinates (37.5395, 37.7910) and longitude (-122.5153, -122.3789) which encompasses a high density of points in San Francisco (SF). For Austin, the specific region is between the latitude coordinates (29.833, 30.838) and longitude (-96.700, -98.592) which encompasses a high density of points in Austin as well as a lower density around Austin. We quantize the area into 500 x 200 cells and consider the center of the cells as our dataset. In total, the SF dataset has 6890 points and the Austin dataset has 4305.

We compare our method (TT) to geo-indistinguishability (GEO) as to our knowledge it is the only known method which protects an individual query with DP (Section 4). We accomplish this by using the Laplace mechanism in 1-dimension and the planar Laplace

mechanism in two dimensions [2]. We use the final ϵ value calculated via Bounded Range composition (Theorem 2.13) for DP-TT as the privacy level ϵ^* for GEO. Recall that the Laplace mechanism (Definition 2.4) requires the sensitivity of the function the noise is being added to. In our case, the function is the raw search value, so the sensitivity is the maximum difference between values which is equivalent to 1 billion. As we will demonstrate, using this global sensitivity leads to bad accuracy, which is why geo-indistinguishability is useful.

Following the principles of geo-indistinguishability, we vary the r value in the privacy level guarantee allowing us to demonstrate the accuracy of the local mechanism given certain privacy levels. For example, for our distance metric experiments we set $\epsilon^* = 1$. We vary r such that $1,000,000,000 \leq r \leq 1,000$. When $r = 1,000,000$, the privacy parameter is $\epsilon = \epsilon^*/r = 0.000001$. When $r = 1,000$ the privacy parameter $\epsilon = 0.001$. To demonstrate the accuracy that is attained when $d(x, x') \leq r$, we set the sensitivity of the Laplace mechanism to be smaller than its global sensitivity. Recall that the Laplace mechanism adds noise drawn from $\text{Lap}(\Delta f/\epsilon^*)$ where f is the "position" value in our case, so Δf is the sensitivity. For GEO and the distance metric DP-TT, we vary the sensitivity such that $1,000,000,000 \leq \Delta f \leq 1,000$ incrementing by a degree of magnitude at every interval.

Communication comparison A key difference between DP-TT and GEO is the communication cost. GEO only requires one round of communication. DP-TT is an interactive protocol and requires as many rounds as times the exponential mechanism is used. Specifically, for basic DP-TT, $\log_2 N$ rounds are necessary. Figure 6.1 shows the percentage of data points sent by the various DP-TT variants as well as the naive approach of the server sending the entire tree to the client. We can see that the relationship between the percentage of data points and the size of the database is near linear.

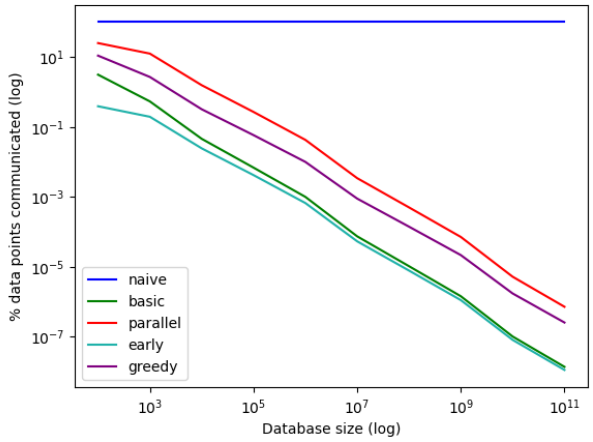


Figure 6.1: Accuracy of basic private NN search for 1D (DP-TT-DIS and GEO)

Distance utility (DIS) In order to replicate this concept of bounding the r value of geo-indistinguishability in our DP-TT-DIS method, we similarly bound the sensitivity of

the utility function of the Exponential mechanism. Recall that the utility metric in k dimensions, where x is the client's data and h is either the left or right children's data is:

$$u_1(x, h) = \sqrt{\sum_{i}^{k} (x_i, h_i)^2}$$

In the worst case the sensitivity of u_1 is 1 billion, but we bound this sensitivity similar to GEO.

Comparison metric (CMP) This metric is independent of distances so we do not bound any sensitivity. The sensitivity is 1, given that the output of this utility metric is a boolean bit.

6.2 Results

We evaluate two types of accuracy. The first is the *raw accuracy* (NN Acc). This refers to the percentage of the time that the true nearest neighbor of the search value is found. The second is the *top 5 accuracy* (Top 5 Acc). This is a broader accuracy metric and measures how often any of the five closest neighbors are found. If any number of those 5 neighbors are found, then the top 5 accuracy is the same as if only one had been found.

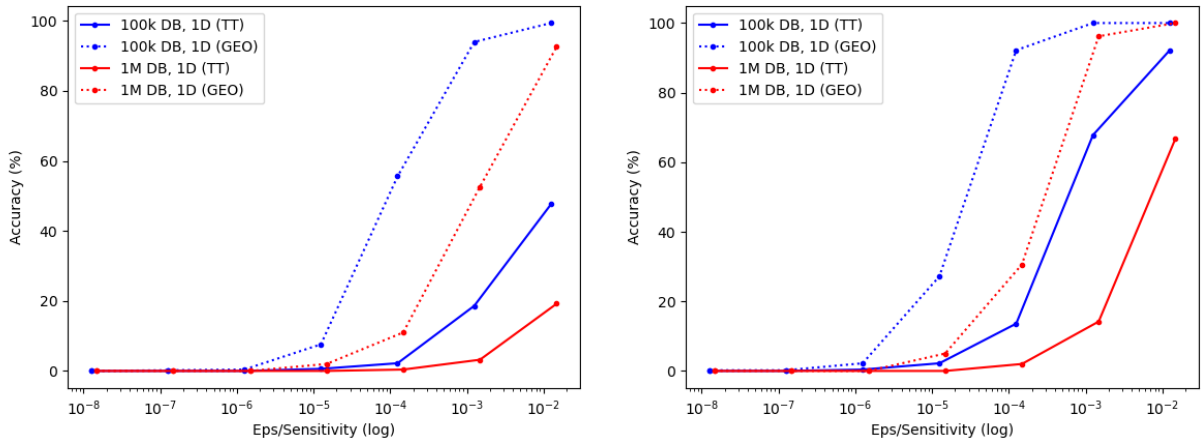
For example, if three results are returned by DP-TT and of those three, one is the true nearest neighbor and another is the second closest nearest neighbor, then the raw accuracy and top 5 accuracy are the same (100%). However, if only the second closest nearest neighbor is among the three results, then the raw accuracy is 0% while the top 5 accuracy is 100%.

6.2.1 Distance metric (DIS)

For GEO and the DP-TT-DIS, we set $\epsilon = 1$ and vary the sensitivity such that $1,000,000,000 \leq \Delta f \leq 1,000$ incrementing by a degree of magnitude at every interval. Both GEO and DP-TT-DIS generate noise proportional to $\epsilon/\Delta f$ so we use this as our experiment variable. Notice that ϵ and Δf can vary relative to one another and achieve the same noise distribution.

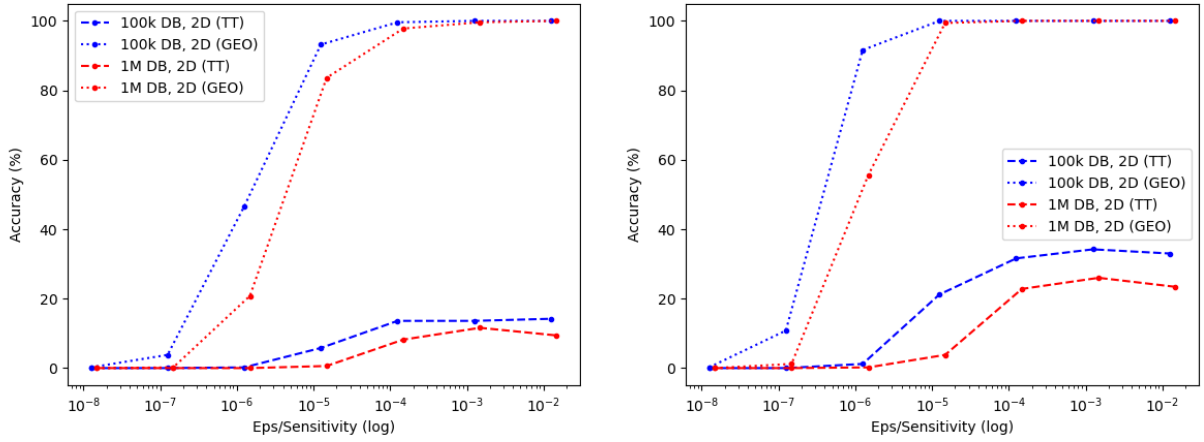
Figures 6.2 and 6.3 show the accuracy of basic DP-TT-DIS (Section 5.1) relative to GEO in 1 and 2-dimensions. As mentioned above, the x-axis represents the varying of the sensitivities. Overall, increasing the size of the database lowers the accuracy per privacy radius.

Notice that in 1-dimension GEO is as good as DP-TT-DIS but at a higher privacy radius. The top 5 accuracy of GEO for a database of size 100k is about 90% with a privacy



(a) Raw accuracy (b) Top 5 accuracy

Figure 6.2: Accuracy of basic private NN search for 1D (DP-TT-DIS and GEO)

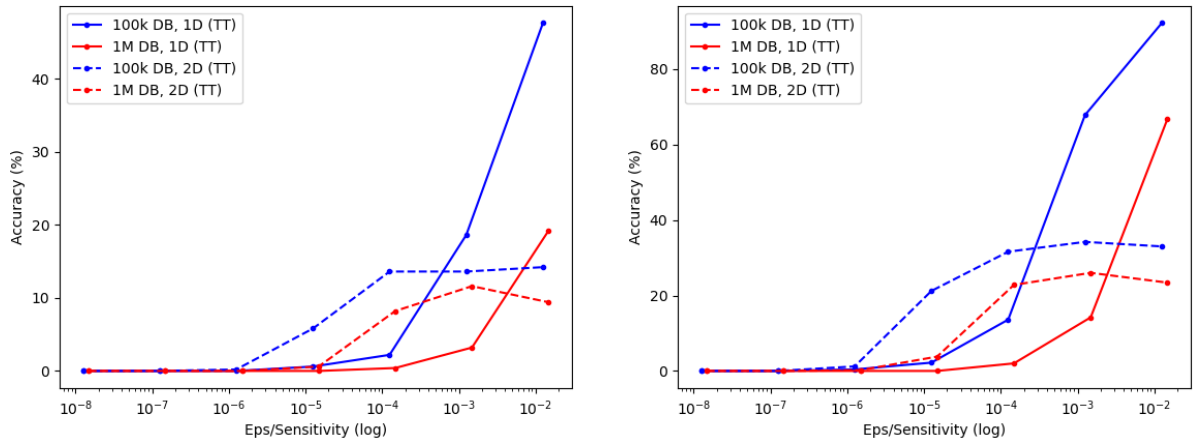


(a) Raw accuracy (b) Top 5 accuracy

Figure 6.3: Accuracy of basic private NN search for 2D (DP-TT-DIS and GEO)

radius of 100,000 while DP-TT-DIS achieves that same accuracy at a privacy radius of 1,000. In 2-dimensions, we notice that adding a dimension degrades the accuracy by quite a bit. The top 5 accuracy plateaus around 30%, but is larger than in 1-dimension at higher privacy radii (Figure 6.4).

We can see that DP-TT-DIS does work but with a higher ϵ or smaller sensitivity than GEO. We also notice that as the size of the database increases, the accuracy at the same privacy radius decreases. We next evaluate increasing the number of values output by introducing splitting and releasing the nearest neighbors of the found values. We present graphs for the database of size 100k as the database of size 1 million exhibits the same trends, just shifted to lower privacy radii.

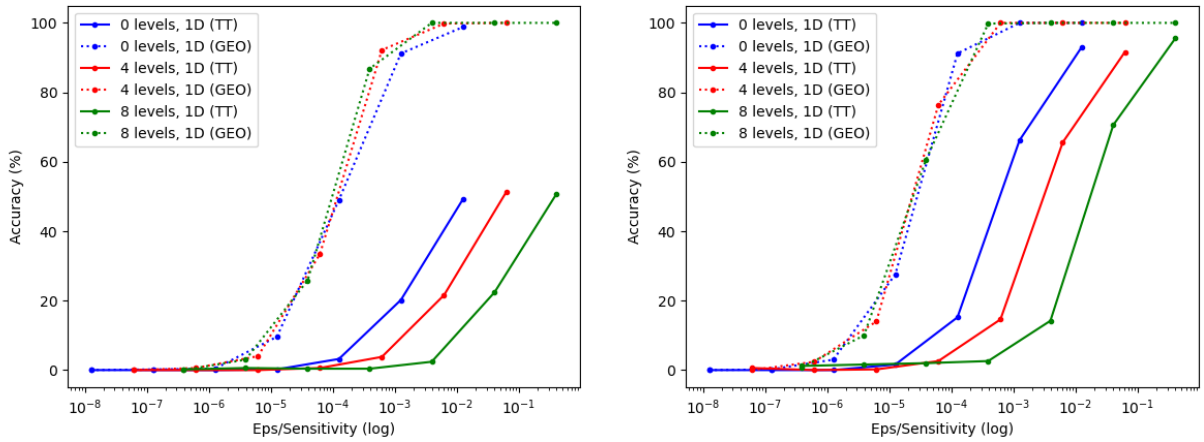


(a) Raw accuracy (b) Top 5 accuracy

Figure 6.4: Comparing accuracy of DP-TT-DIS for 1 and 2-dimensions

Parallel traversal

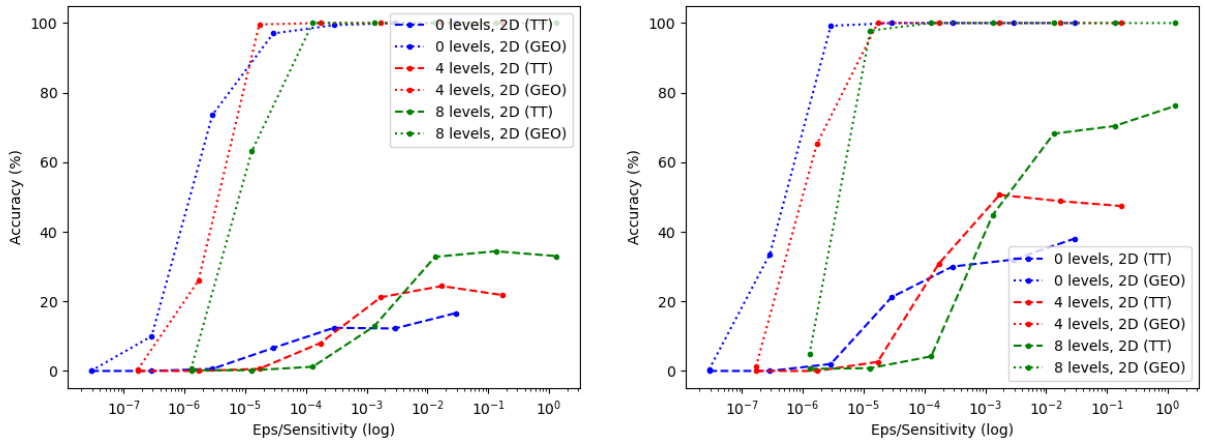
When evaluating any DP-TT variant that returns multiple values such as parallel traversal or neighborhoods, we correspondingly increase the number of times GEO is run. Specifically, for the parallel traversal method evaluated next, we run GEO as many times as the number of final values released by DP-TT-DIS P.



(a) Raw accuracy (b) Top 5 accuracy

Figure 6.5: Accuracy of DP-TT-DIS P for database of size 100k in 1-dimension

Figures 6.5 and 6.6 show the performance of DP-TT-DIS P with parallel traversals of the tree. When there are 0 splits of the thread, this is equivalent to basic DP-TT-DIS. In all scenarios, GEO performs better than DP-TT-DIS. In 1-dimension (Figure 6.5), we can see that parallel traversals does not improve the accuracy of DP-TT-DIS, in fact the more splits at the top, the worse the accuracy at the same privacy radius. We attribute this to



(a) Raw accuracy

(b) Top 5 accuracy

Figure 6.6: Accuracy of DP-TT-DIS P for database of size 100k in 2-dimensions

the fact that the final ϵ composition factor is 2^s times bigger, where s is the number of splits.

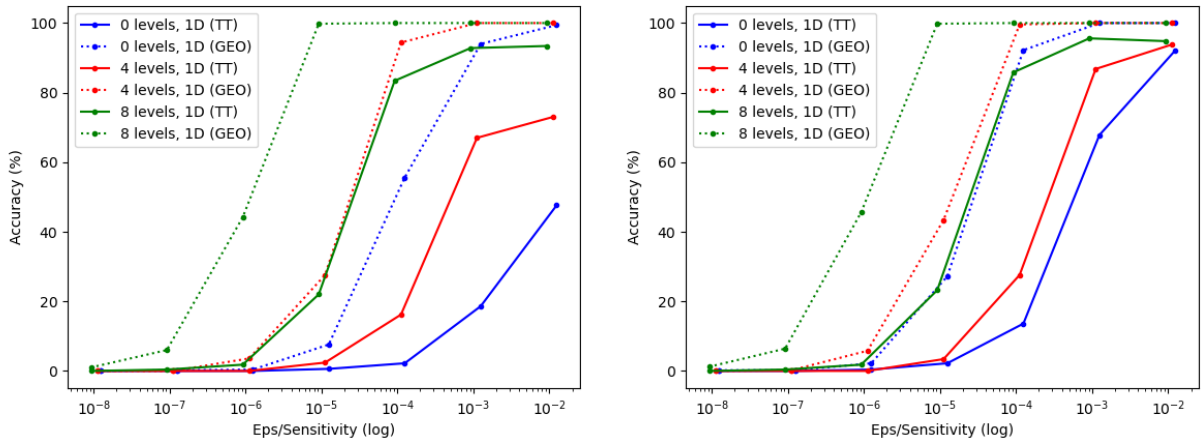
However in 2-dimensions (Figure 6.6), parallel traversal overtakes basic DP-TT-DIS. We attribute this to the fact that backtracking is more essential to a 2-d tree than to a BST (for 1-dimensions). Since basic DP-TT-DIS initially performs better than having parallel traversals, we believe that there comes a point where the privacy cost is sufficiently small enough that having more threads is beneficial. The 2-dimensional search exhibits a similar plateau behavior as basic DP-TT-DIS and the raw accuracy plateaus at 40%.

We next evaluate early stopping of the search thread.

Early stopping

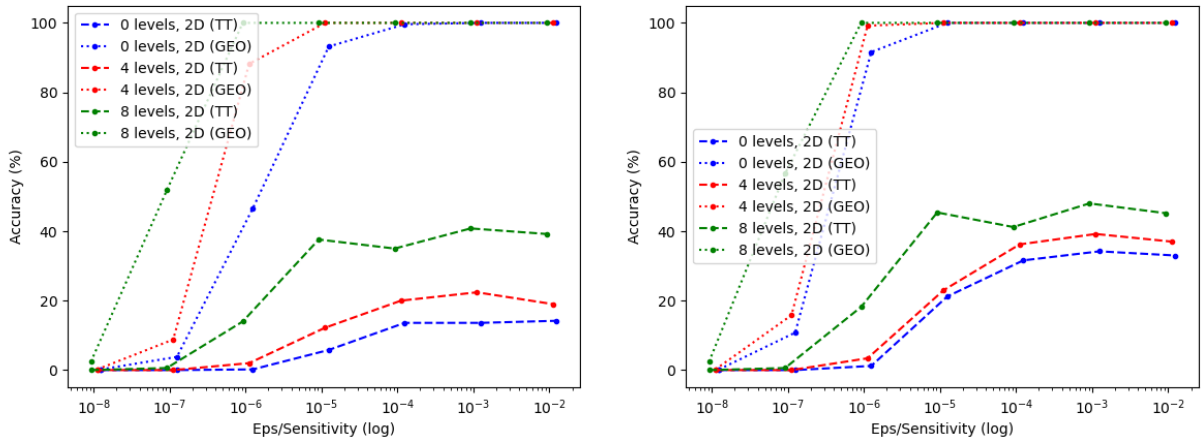
For the DP-TT-DIS E, to fairly evaluate GEO, instead of running GEO as many times as the number of values outputted, we run GEO once and find the neighbors for the output. The number of the neighbors is equivalent to the number of values outputted by the tree. We make this decision due to the fact that the values outputted by the search thread splitting at the bottom will all be tree neighbors. We recognize that in 1-dimension, tree neighbors are equivalent to true neighbors, but in higher dimensions that is not the case. However this is the fairest comparison we could implement.

Figures 6.7 and 6.8 show the performance of DP-TT-DIS E. Here we notice that the earlier the splits occur the higher the accuracy at the same level of privacy. We attribute this to the fact that since the splits happen at the bottom, it reduces the final value of ϵ while increasing the number of values outputted. Further, since these values are tree neighbors, it removes the inaccuracy that the last few levels would have incurred had they gone through the standard path selection. In 1-dimension, DP-TT-DIS E achieves high accuracy, but in 2-dimensions, it plateaus around 40%.



(a) Raw accuracy (b) Top 5 accuracy

Figure 6.7: Accuracy of GEO and DP-TT-DIS E for database of size 100k in 1-dimension



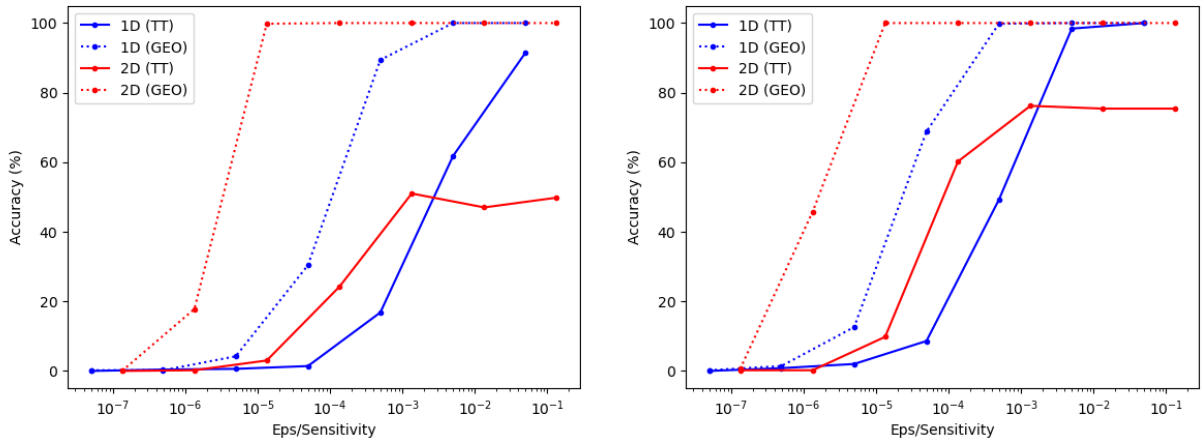
(a) Raw accuracy (b) Top 5 accuracy

Figure 6.8: Accuracy of GEO and DP-TT-DIS E for database of size 100k in 2-dimension

Greedy splitting

We next evaluate greedy splitting at every level. Similar to parallel traversal, GEO is run for as many times as values outputted by the tree since those values are not tree or true neighbors.

Figure 6.9 shows DP-TT-DIS GS for a database of size 100k in 1 and 2-dimensions. Once again we notice that the accuracy of the 2-dimensional database plateaus earlier than in 1-dimension.

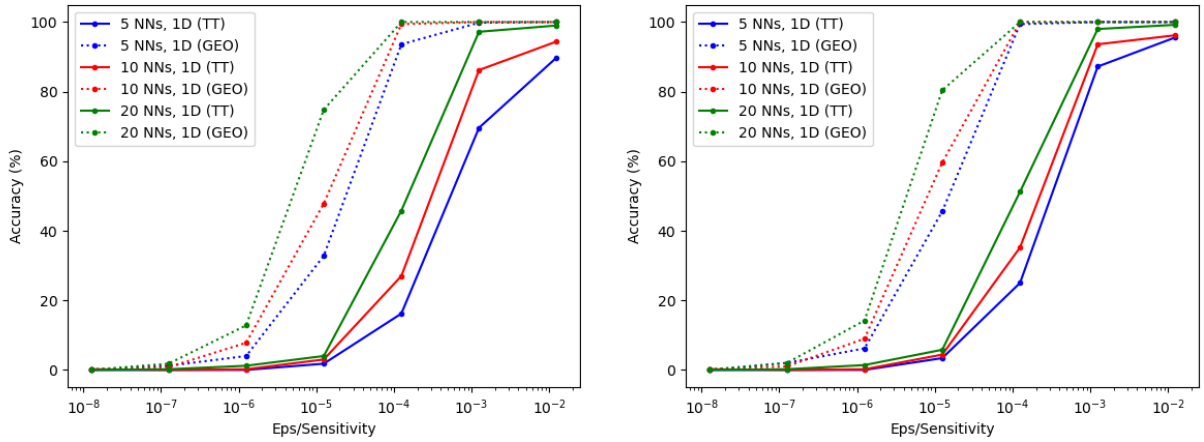


(a) Raw accuracy (b) Top 5 accuracy

Figure 6.9: Accuracy of GEO and DP-TT-DIS GS for database of size 100k in 1 and 2-dimensions

Neighborhoods

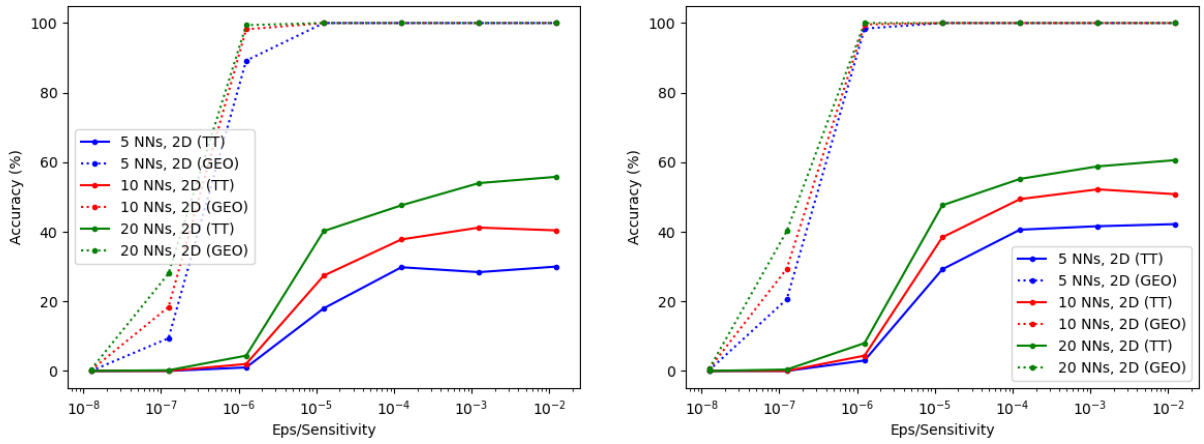
We now evaluate releasing neighborhoods of values found by the tree. This does not change the final ϵ value compared to basic DP-TT.



(a) Raw accuracy (b) Top 5 accuracy

Figure 6.10: Accuracy of DP-TT-DIS N for database of size 100k in 1-dimension

Figures 6.10 and 6.11 show the performance of releasing neighborhoods. In 1-dimension there is a small benefit to releasing a larger neighborhood while in 2-dimensions releasing a neighborhood of 20 values roughly doubles the accuracy from releasing 5 of them.

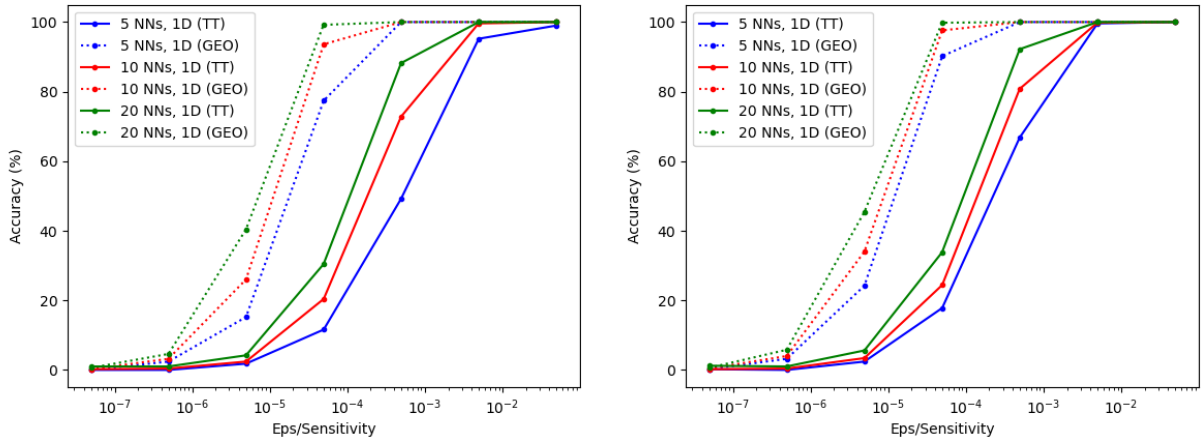


(a) Raw accuracy (b) Top 5 accuracy

Figure 6.11: Accuracy of DP-TT-DIS N for database of size 100k in 2-dimensions

Combining neighborhoods with splitting

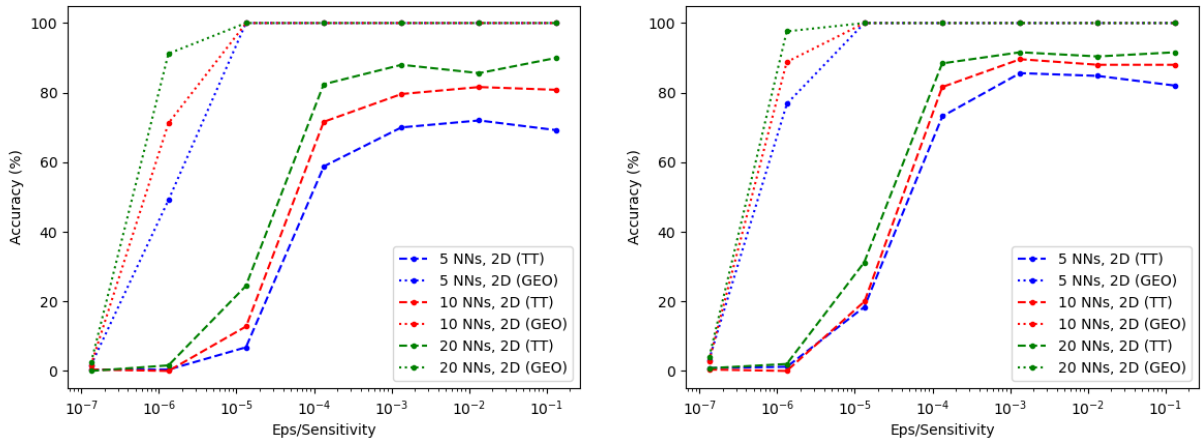
We now combine releasing neighborhoods as well as splitting at every level (DP-TT-DIS GS-N). For comparison to GEO, we run GEO as many times as DP-TT splits and then for every value found we find the neighborhood of that value.



(a) Raw accuracy (b) Top 5 accuracy

Figure 6.12: Accuracy of DP-TT-DIS GS-N for database of size 100k in 1-dimension

Figures 6.12 and 6.13 show the performance for DP-TT-DIS GS-N. In 1-dimension, we see a marginal benefit to releasing a larger neighborhood. In 2-dimension, we see that the raw accuracy benefits the most from a larger neighborhood while the top 5 accuracy does not show much improvement.

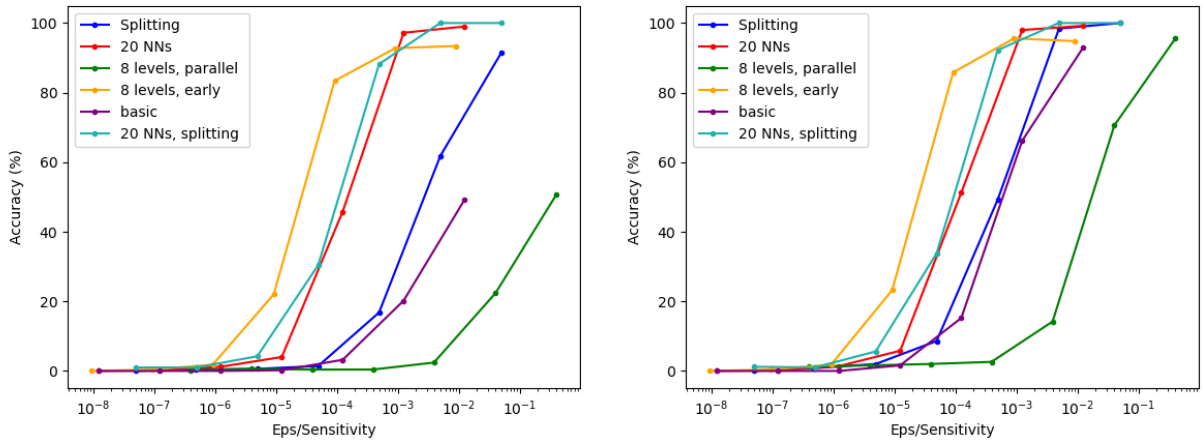


(a) Raw accuracy (b) Top 5 accuracy

Figure 6.13: Accuracy of DP-TT-DIS GS-N for database of size 100k in 2-dimensions

Comparing methods

We now compare all these methods and identify the best one.

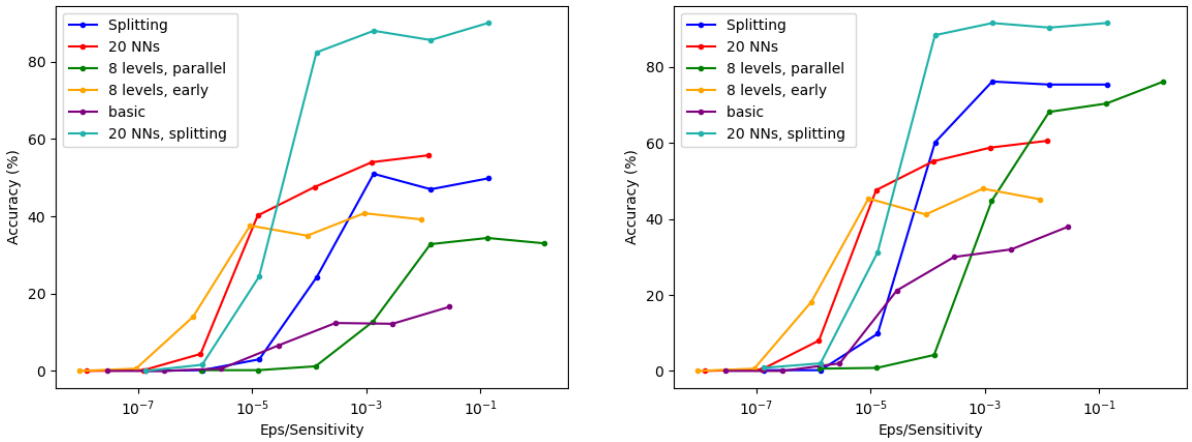


(a) Raw accuracy (b) Top 5 accuracy

Figure 6.14: Comparing different tree methods for database of size 100k in 1-dimension

Figures 6.14 and 6.15 show the comparison between the different methods explored. In 1-dimension, we note that early stopping 8 levels early (DP-TT-DIS E8) performs most optimally while parallel traversal (DP-TT-DIS P8) performs the worst. Looking up the neighborhood of found values (DP-TT-DIS N20) performs about the same whether splitting is implemented or not. We note that splitting at every level performs worse than releasing neighborhoods, suggesting that the benefit seen in the combined approach stems from the neighborhood and not the splitting.

In 2-dimensions, early stopping (DP-TT-DIS E8) and releasing a neighborhood (DP-



(a) Raw accuracy

(b) Top 5 accuracy

Figure 6.15: Comparing different tree methods for database of size 100k in 2-dimensions

TT-DIS N20) perform better at a higher privacy radius but are overtaken by the combined splitting and neighborhood approach (DP-TT-DIS GS-N). The basic method performs the worst in the long term but is better than the parallel traversal initially. Greedy splitting (DP-TT-DIS GS) performs badly at higher privacy radii, but overtakes neighborhoods in terms of top 5 accuracy. The combined splitting and neighborhood method achieves the highest accuracy of about 85% at the privacy radius of $r = 10,000$ which is about 0.001% of the global sensitivity given a privacy level $\epsilon^* = 1$. Here the combined method shows signs of benefiting from both aspects of splitting and neighborhoods.

It is important to note that each of these DP-TT variants result in a different amount of results released by the server. DP-TT-DIS E8 results in at most $2^8 = 256$ results. DP-TT-DIS GS and DP-TT-DIS N20 results in at most $(\lceil \log_2 N + 1 \rceil) * 20$ results, which in the case of a database of 100k is 360 results. Only splitting at every level results in $\lceil \log_2 N + 1 \rceil$ or 18 (for database of size 100k) results.

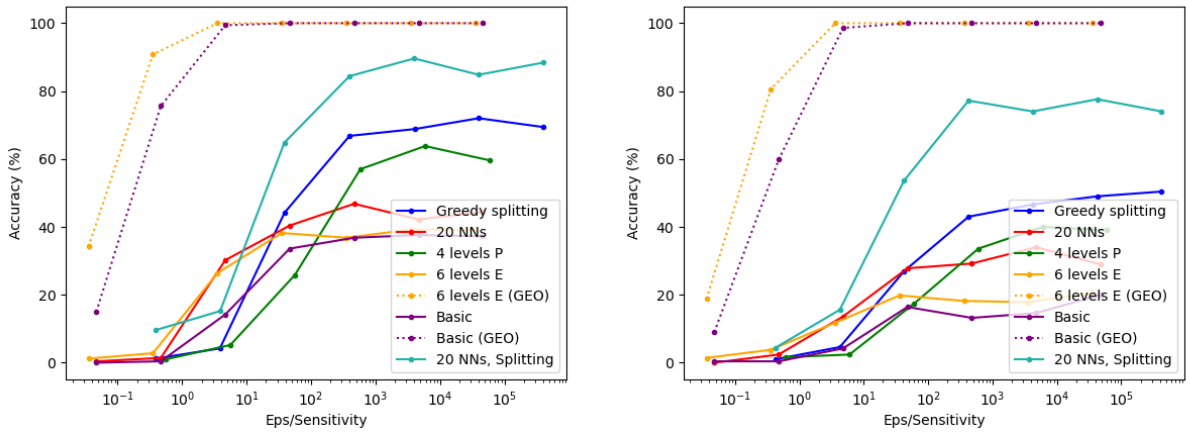
Applying methods to real world datasets

We run the experiments on the SF and Austin datasets and compare them.

Figure 6.16 shows the comparison of all the variants previously tested. They exhibit the same trends as the 2-dimensional dataset in that the combined greedy splitting and neighborhood method preforms best, along with the greedy splitting and parallel traversal.

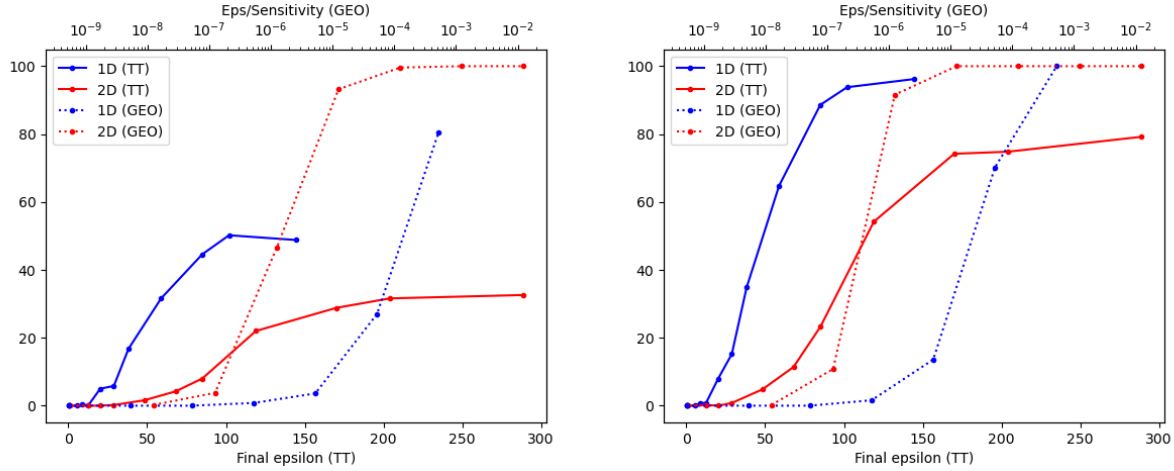
6.2.2 Comparison metric (CMP)

We now evaluate DP-TT using the comparison metric (DP-TT-CMP). This metric is distance independent so comparing it to geo-indistinguishability is more difficult. We use the



(a) Austin dataset (b) San Francisco dataset

Figure 6.16: Comparison of DP-TT-DIS methods for real world dataset



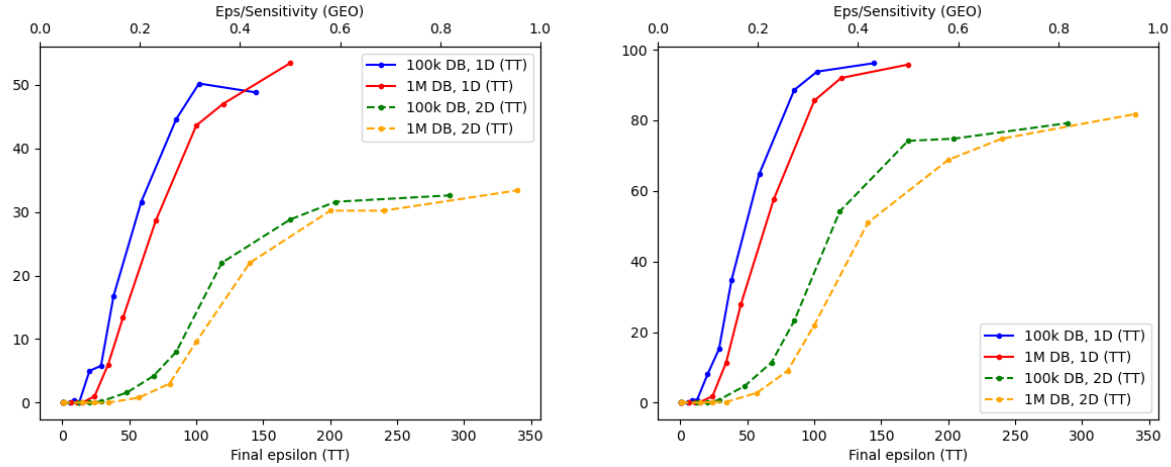
(a) Raw accuracy (b) Top 5 accuracy

Figure 6.17: Accuracy of DP-TT-CMP and GEO for database of size 100k in 1 and 2-dimensions

same parameters for GEO as previously stated. Comparison DP-TT relies solely on ϵ so we vary it with the following values $[0.01, 0.1, 1, 1.5, 2, 3, 4, 5, 7, 10, 12, 17]$ and graph the final ϵ value calculated with bounded range composition.

GEO performs much worse due to the fact that it is used with global sensitivity. In order to get a better comparison, we will plot GEO keeping ϵ constant and varying the bounded sensitivity (Figure 6.17) and present GEO's scale at the top of graphs.

Figure 6.18 shows that the accuracy degrades as the size of the database increases. The ϵ value necessary to achieve a reasonable accuracy for DP-TT-CMP is quite large. This is



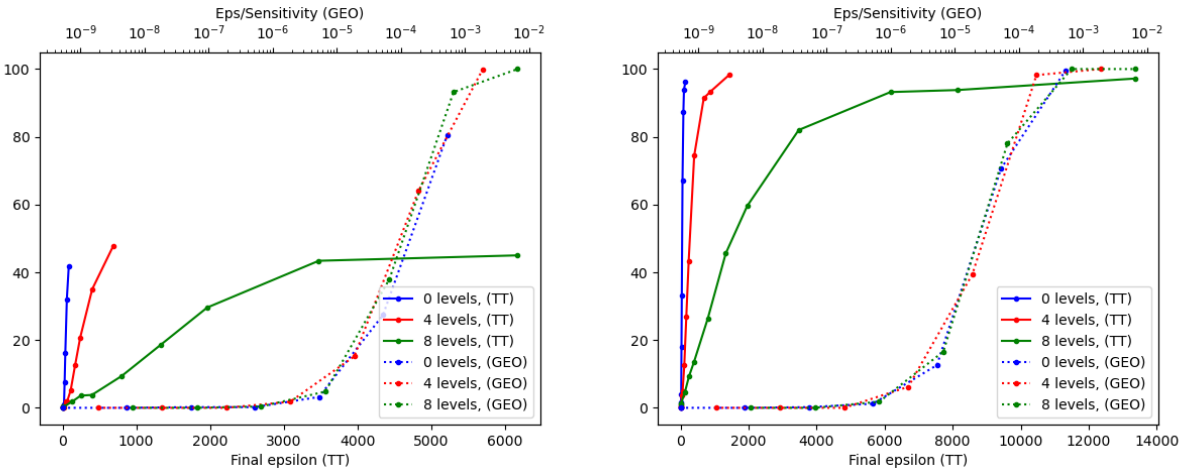
(a) Raw accuracy

(b) Top 5 accuracy

Figure 6.18: Accuracy of DP-TT-CMP for database of size 100k and 1M in 1 and 2-dimensions

not surprising as the ϵ used at every node is composed $\lceil \log_2(N) \rceil$ times. The 2-dimensional search does not achieve 100% accuracy and plateaus at 80% accuracy. The raw metric is half as accurate as the top 5 metric.

Parallel traversal

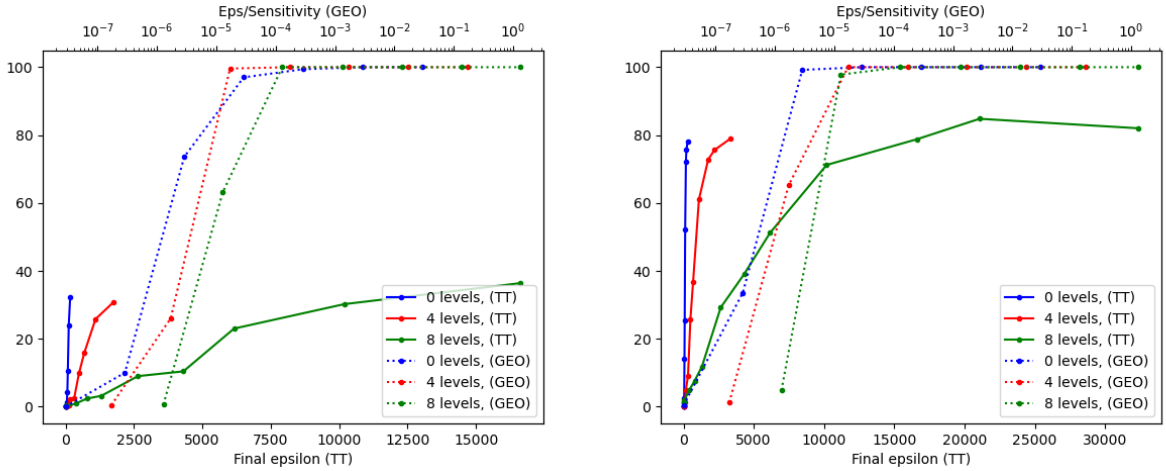


(a) Raw accuracy

(b) Top 5 accuracy

Figure 6.19: Accuracy of DP-TT-CMP P for database of size 100k in 1-dimension

As Figures 6.19 and 6.20 show, parallel traversal does not provide any benefits. In fact, the privacy budget used by DP-TT-CMP P8 is so large that the P0 and P4 lines

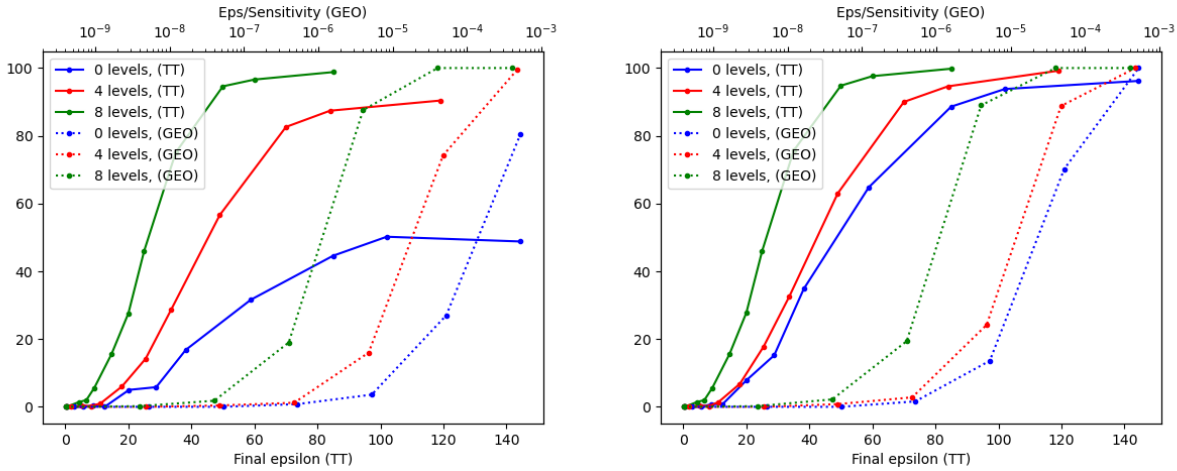


(a) Raw accuracy (b) Top 5 accuracy

Figure 6.20: Accuracy of DP-TT-CMP P for database of size 100k in 2-dimensions

are flattened to near vertical when they actually plateau when graphed on an appropriate scale.

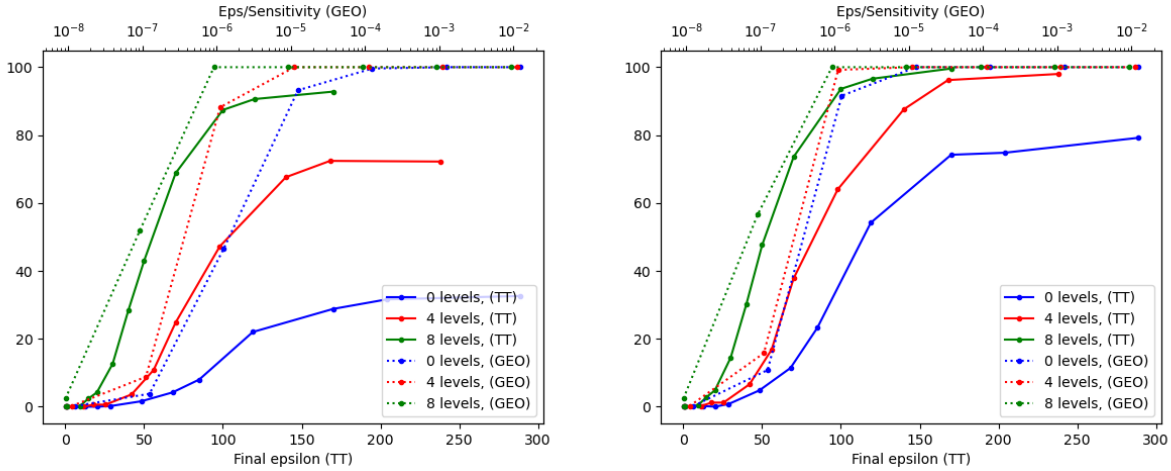
Early stopping



(a) Raw accuracy (b) Top 5 accuracy

Figure 6.21: Accuracy of DP-TT-CMP E for database of size 100k in 1-dimension

We also look at stopping the search thread early at the bottom (DP-TT-CMP E). In order to fairly compare this method to GEO, we include the nearest neighbors to the found value in GEO such that as many values are included in GEO as output by DP-TT-CMP E.

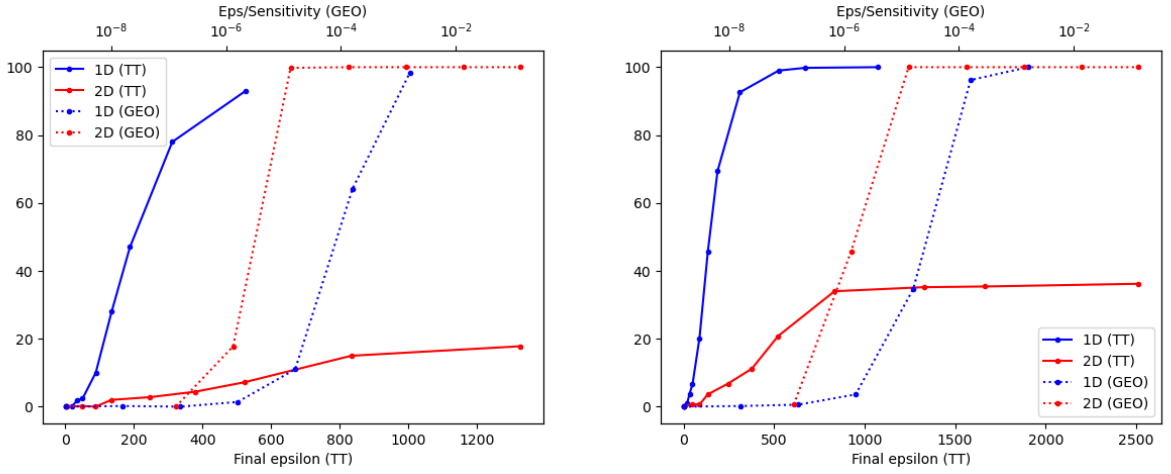


(a) Raw accuracy (b) Top 5 accuracy

Figure 6.22: Accuracy of DP-TT-CMP E for database of size 100k in 2-dimensions

Figures 6.21 and 6.22 show that the earlier the thread stops and splits, the higher the accuracy. This also means that more values are outputted by DP-TT-CMP E (2^s values with s the number of splits). In 2-dimensions, stopping 8 levels early allows the traversal to achieve 100% accuracy for a final $\epsilon = 100$.

Greedy splitting



(a) Raw accuracy (b) Top 5

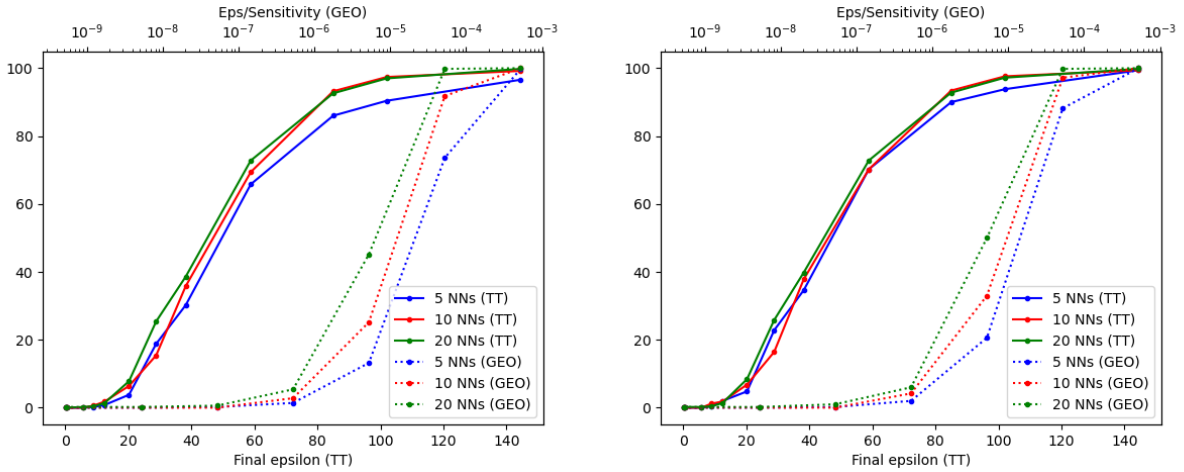
Figure 6.23: Accuracy of DP-TT-CMP GS for database of size 100k in 1 and 2-dimensions

The final method we explore is greedy splitting at every level (DP-TT-CMP GS).

Figure 6.23 shows that the 1-dimension mechanism achieves high accuracy while the 2-dimension mechanism achieves a maximum of 40% top 5 accuracy.

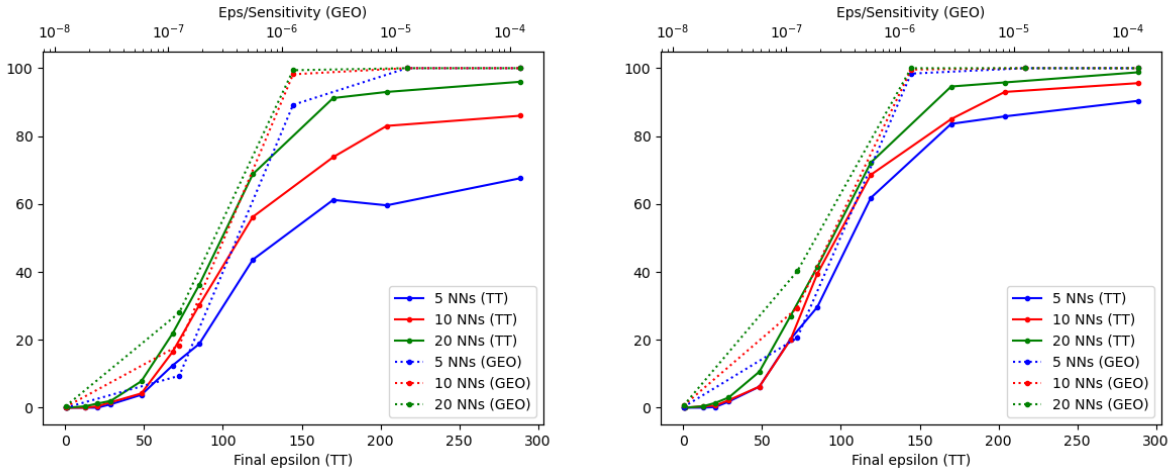
Neighborhoods

We also looked at releasing the neighborhoods of the found value (DP-TT-CMP N).



(a) Raw accuracy (b) Top 5 accuracy

Figure 6.24: Accuracy of DP-TT-CMP N for database of size 100k in 1-dimension



(a) Raw accuracy (b) Top 5 accuracy

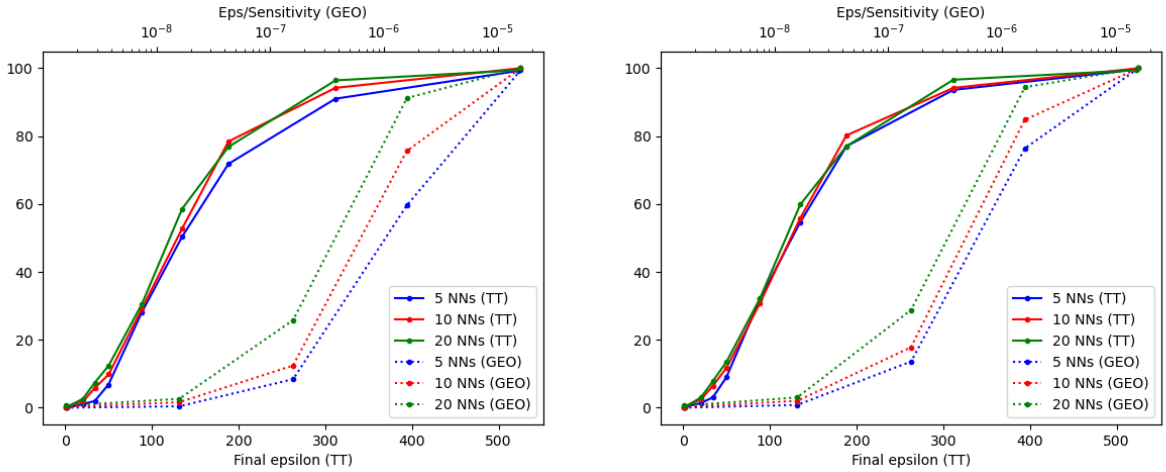
Figure 6.25: Accuracy of DP-TT-CMP N for database of size 100k in 2-dimensions

Figures 6.24 shows that in 1-dimension the size of the neighborhood makes little to no difference in the accuracy of the mechanism. Figure 6.25 shows that the size of the

neighborhood improves the raw accuracy but has a less important impact when it comes to top 5 accuracy.

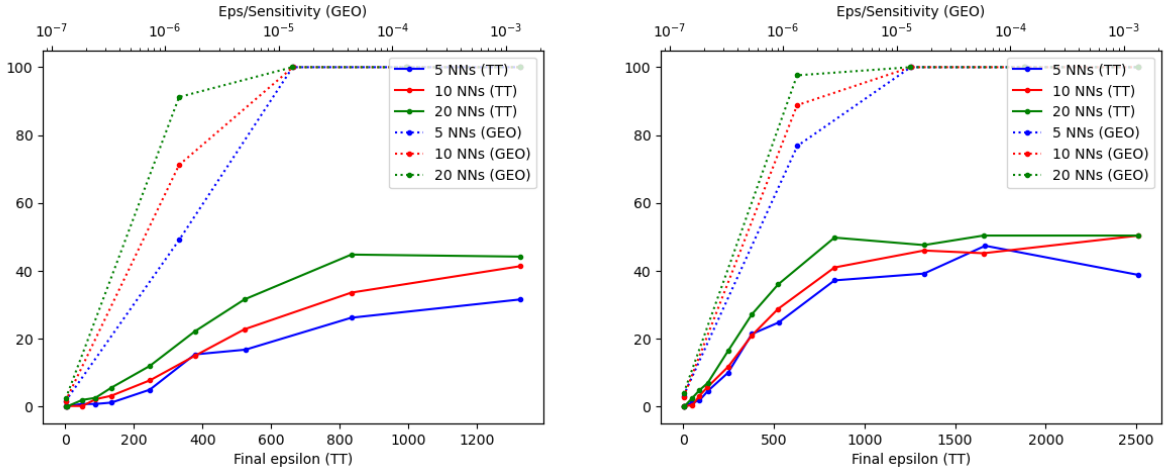
Combining neighborhoods with greedy splitting

Finally, we looked at combining both greedy splitting and neighborhoods (DP-TT-CMP GS-N).



(a) Raw accuracy (b) Top 5 accuracy

Figure 6.26: Accuracy of DP-TT-CMP GS-N for database of size 100k in 1-dimension



(a) Raw accuracy (b) Top 5 accuracy

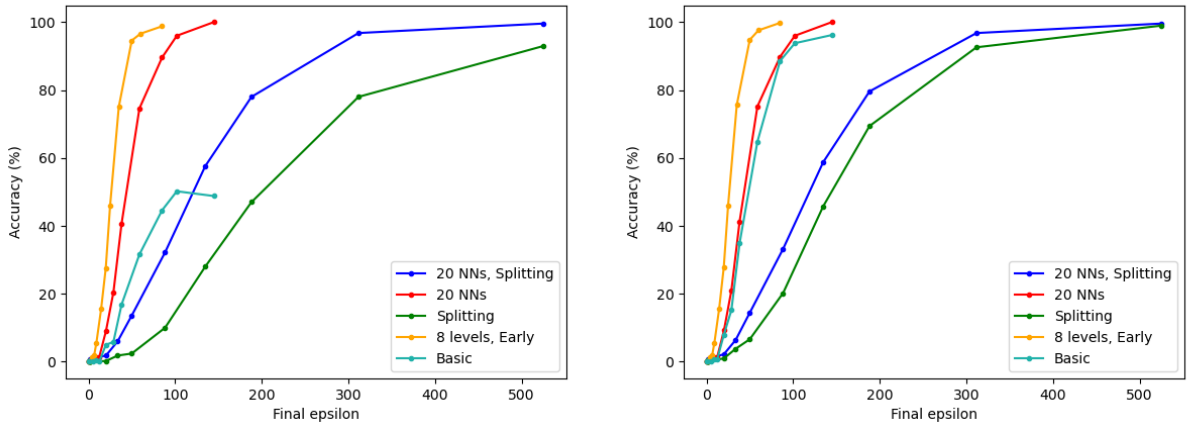
Figure 6.27: Accuracy of DP-TT-CMP GS-N for database of size 100k in 2-dimensions

Figure 6.26 shows that adding neighborhoods to splitting at every level does not change the accuracy based on neighborhood size. Figure 6.27 shows that the size of the neighbor-

hood improves the raw accuracy but has a less important impact when it comes to top 5 accuracy.

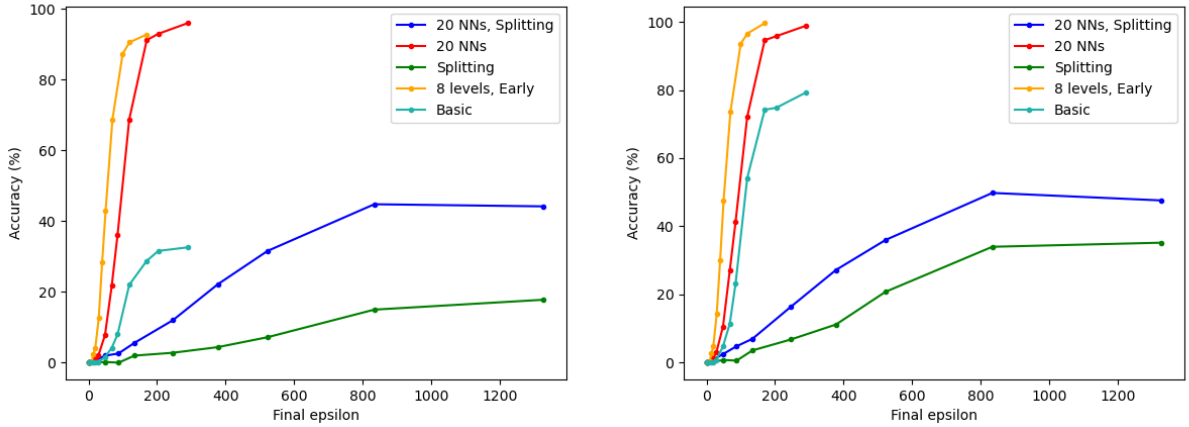
Comparing methods

We now compare all methods. We exclude the parallel traversal mechanism as it performs worse than basic DP-TT-CMP.



(a) Raw accuracy (b) Top 5 accuracy

Figure 6.28: Comparison of DP-TT variants for database of size 100k in 1-dimension



(a) Raw accuracy (b) Top 5 accuracy

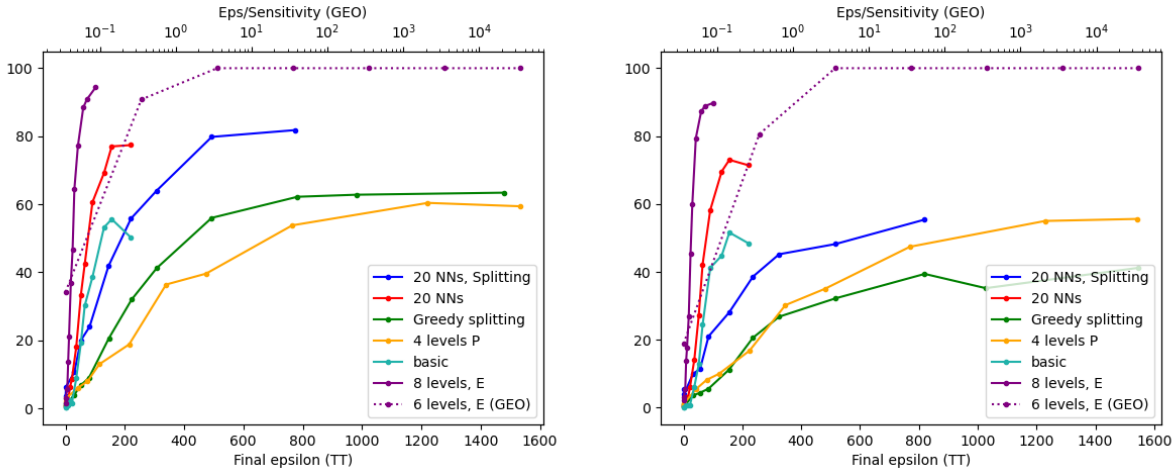
Figure 6.29: Comparison of DP-TT variants for database of size 100k in 2-dimensions

Figures 6.28 and 6.29 show the comparison of the different DP-TT-CMP variants. We note that early stopping provides the most benefit for the smallest privacy budget for both

dimensions. The raw accuracy for basic DP-TT-CMP is about half as accurate as the top 5 metric. However all other methods have comparable top 5 and raw accuracies.

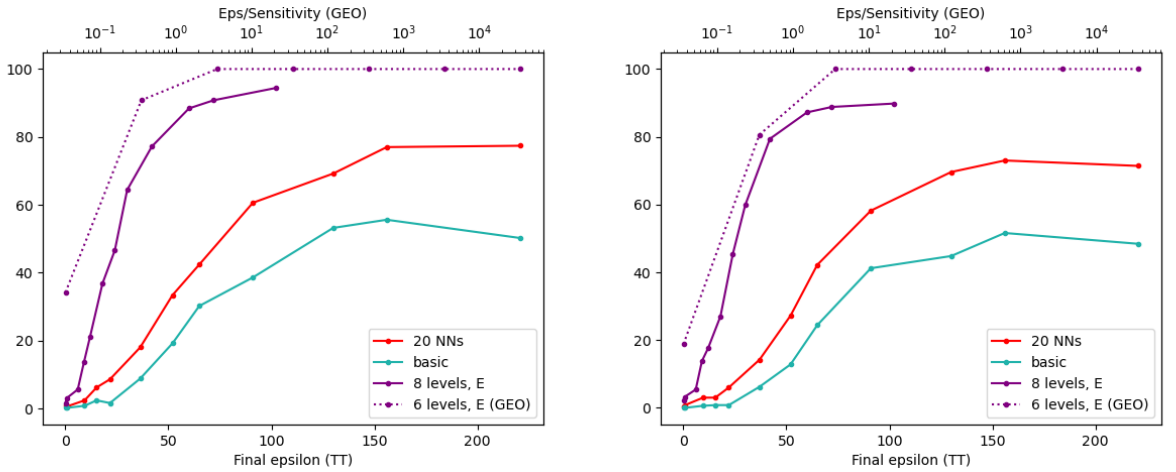
Applying methods to real world datasets

We run the experiments on the SF and Austin datasets and compare them.



(a) Austin dataset (b) San Francisco dataset

Figure 6.30: Comparison of DP-TT-CMP methods for real world dataset



(a) Austin dataset (b) San Francisco dataset

Figure 6.31: Comparison of best DP-TT-CMP methods for real world dataset

Figure 6.30 shows the comparison of all the variants previously tested. They exhibit the same trends as the 2-dimensional dataset in that the early stopping and neighborhood

methods perform best. We remove the worst performing methods to more clearly see the behavior of the best methods in Figure 6.31.

6.2.3 Analysis

In this section we evaluated geo-indistinguishability relative to the tree traversal methods we propose using two different metrics, distance and comparison. Geo-indistinguishability and the distance based DP-TT are more easily comparable as they both use distance metrics to randomize results. For both metrics, the parallel traversal method performs worse than basic DP-TT.

Distance metric In 1-dimension, the distance metric achieves usable accuracy, but at a lower privacy radius than geo-indistinguishability does. Of all DP-TT-DISs we tried (Figure 6.14), early stopping (E) was the most effective. This is due to more values being released for a lower privacy budget. However, releasing neighborhoods (N), which achieves a similar guarantee of releasing more values at no cost to the privacy budget does not perform as well. In terms of raw accuracy, basic DP-TT-DIS and parallel traversal (P) perform the worst of the compared methods.

Unlike in 1-dimension, in 2-dimensions, greedy splitting with neighborhoods performs the best overall. We attribute this to the fact that the 1-dimensional traversal of a tree does not require backtracking to find the correct value, but in higher dimensions backtracking is necessary. In 1-dimension the benefits of splitting are outweighed by the extra privacy cost it incurs. In 2-dimensions, splitting, which simulates backtracking, demonstrates its benefits by performing relatively well. Neighborhoods also performs reasonably well and the combination of splitting and neighborhoods performs the best.

Comparison metric The comparison metric achieves usable accuracy as well, but at the cost of a high privacy budget. In 1-dimension, the best method identified, early stopping, achieves a 95% accuracy for a database of size 100k at a privacy budget of $\epsilon = 50$. To achieve a similar accuracy, GEO requires a $\epsilon^*/\Delta f$ value of about 10^{-5}. If we assume an $\epsilon^* = 50$ for GEO, the sensitivity would need to be $5,000,000$ which is 0.5% of the global sensitivity. So to achieve a 95% accuracy for a protection radius of $5,000,000$, the privacy parameter is $\epsilon = 50/5,000,000 = 10^{-5}$.

Unlike with the distance metric, there is no difference between which variant works best for 1-dimension versus 2-dimensions. We attribute this to the way that the privacy budget is calculated. For the distance metric the privacy level of ϵ^* makes little difference until an order of magnitude of change occurs since the privacy parameter is calculated from ϵ^* and the radius. For example, $\epsilon = \epsilon^*/\Delta = 10^{-4}$ versus $\epsilon = 3 * 10^{-4}$ makes comparably little difference relative to $\epsilon = 10^{-4}$ and $\epsilon = 10^{-5}$. So spreading out the privacy budget among several threads does not impact the final epsilon budget by a lot relatively. For the

comparison metric, our results show that the methods that spend all their privacy budget on one thread (early stopping, basic, and neighborhoods) perform the best compared to the methods that spend more on overall.

Real world data We tested our methods on two real world datasets and note that the behavior seen with our artificial data carries over to these datasets.

Takeaways As is typical of differentially private mechanisms, privacy and utility compete against one another. We propose two methods which achieve usable utility for different privacy budgets. DP-TT-CMP provides a method that does not require any bounding of a distance metric. DP-TT-DIS provides a method that does require bounding of a distance metric but is not better than geo-indistinguishability.

For the comparison metric, the poor performance of the greedy splitting method confirms that backtracking would not be beneficial as the addition to the privacy budget outweighs any increase in utility. However for the distance metric, greedy splitting might have an advantage in 2-dimensions. While greedy splitting simulates the behavior of backtracking, it notably does not add to the privacy cost (as it does not require additional uses of the exponential mechanism) unlike backtracking. From our evaluation, we conclude that distance is not a good choice to help us choose an optimal tree traversal, as the privacy cost outweighs the utility benefits.

6.3 Conclusion

In this research we propose a new method for differentially private nearest neighbor search relative to the search value. Previous work focused on preserving the privacy of the database being searched rather than the searched value. DP nearest neighbor search is a common application for location privacy with the most common implementation being geo-indistinguishability.

We identify two potential metrics for our method, one using distance and the other a simple comparison. We evaluate these metrics along with several variations on a DP tree traversal method and identify a few variations with better accuracy. One metric (distance) does not achieve as good accuracy as geo-indistinguishability for a similar privacy level. The other metric (comparison) is distance-independent, and is thus not tied to any notion of privacy levels for location privacy. However to achieve high accuracy, the privacy budget is quite high.

The comparison metric is advantageous as it does not rely on the notion of extended DP or relaxed local DP and to achieve similar accuracy geo-indistinguishability requires a relatively small privacy level.

In future work, more complex tree traversal algorithms could be explored in order to maximize utility and minimize the privacy budget. More complex nearest neighbor algorithms and datastructures could be explored as well.

References

[1] Ritesh Ahuja, Gabriel Ghinita, and Cyrus Shahabi. A utility-preserving and scalable technique for protecting location data with geo-indistinguishability. In *EDBT*, 2019.

[2] Miguel E. Andrés, Nicolás Emilio Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Geo-indistinguishability: Differential privacy for location-based systems. *CoRR*, abs/1212.1984, 2012.

[3] Jie Bao, Chi-Yin Chow, Mohamed F. Mokbel, and Wei-Shinn Ku. Efficient evaluation of k-range nearest neighbor queries in road networks. In *2010 Eleventh International Conference on Mobile Data Management*, pages 115–124, 2010.

[4] Gilles Barthe, George Danezis, Benjamin Grégoire, César Kunz, and Santiago Zanella-Béguelin. Verified computational differential privacy with applications to smart metering. In *2013 IEEE 26th Computer Security Foundations Symposium*, pages 287–301, 2013.

[5] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, sep 1975.

[6] Nitin Bhatia and Vandana. Survey of nearest neighbor techniques. *International Journal of Computer Science and Information Security*, 8, July 2010.

[7] Nicolás E. Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Optimal geo-indistinguishable mechanisms for location privacy. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, page 251–262, New York, NY, USA, 2014. Association for Computing Machinery.

[8] Ta-Chien Chan and Chwan-Chuen King. *Surveillance and Epidemiology of Infectious Diseases using Spatial and Temporal Lustering Methods*, volume 27, pages 207–234. 10 2011.

[9] Konstantinos Chatzikokolakis, Miguel E. Andrés, Nicolás Emilio Bordenabe, and Catuscia Palamidessi. Broadening the scope of differential privacy using metrics. In Emiliano De Cristofaro and Matthew Wright, editors, *Privacy Enhancing Technologies*, pages 82–102, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[10] Kostas Chatzikokolakis, Ehab Elsalamouny, and Catuscia Palamidessi. Efficient utility improvement for location privacy. *Proceedings on Privacy Enhancing Technologies*, 2017, 10 2017.

[11] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. 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*, KDD '11, page 1082–1090, New York, NY, USA, 2011. Association for Computing Machinery.

[12] Chi-Yin Chow, Mohamed F. Mokbel, and Walid G. Aref. Casper*: Query processing for location services without compromising privacy. *ACM Trans. Database Syst.*, 34(4), dec 2009.

[13] Graham Cormode, Somesh Jha, Tejas Kulkarni, Ninghui Li, Divesh Srivastava, and Tianhao Wang. Privacy at scale: Local differential privacy in practice. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 1655–1658, New York, NY, USA, 2018. Association for Computing Machinery.

[14] Fatemeh Deldar and Mahdi Abadi. A differentially private location generalization approach to guarantee non-uniform privacy in moving objects databases. *Knowledge-Based Systems*, 225:107084, 2021.

[15] Rinku Dewri. Local differential perturbations: Location privacy under approximate knowledge attackers. *IEEE Transactions on Mobile Computing*, 12(12):2360–2372, 2013.

[16] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. Collecting telemetry data privately. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3574–3583, Red Hook, NY, USA, 2017. Curran Associates Inc.

[17] Jinshuo Dong, David Durfee, and Ryan Rogers. Optimal differential privacy composition for exponential mechanisms. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2597–2606. PMLR, 13–18 Jul 2020.

[18] David Durfee and Ryan Rogers. *Practical Differentially Private Top-k Selection with Pay-What-You-Get Composition*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[19] Cynthia Dwork. Differential privacy. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming*, pages 1–12, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[20] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography*, TCC'06, page 265–284, Berlin, Heidelberg, 2006. Springer-Verlag.

[21] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3–4):211–407, aug 2014.

[22] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, page 1054–1067, New York, NY, USA, 2014. Association for Computing Machinery.

[23] Natasha Fernandes, Mark Dras, and Annabelle McIver. Generalised differential privacy for text document processing. In Flemming Nielson and David Sands, editors, *Principles of Security and Trust*, pages 123–148, Cham, 2019. Springer International Publishing.

[24] Natasha Fernandes, Yusuke Kawamoto, and Takao Murakami. Locality sensitive hashing with extended differential privacy. In Elisa Bertino, Haya Shulman, and Michael Waidner, editors, *Computer Security – ESORICS 2021*, pages 563–583, Cham, 2021. Springer International Publishing.

[25] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, sep 1977.

[26] Jacob E. Goodman, Joseph O'Rourke, and Piotr Indyk. Handbook of discrete and computational geometry (2nd ed.). chapter 39: Nearest neighbours in high-dimensional spaces. CRC Press, 2004.

[27] Mehmet Gursoy, Ali Inan, Mehmet Nergiz, and Yucel Saygin. Differentially private nearest neighbor classification. *Data Mining and Knowledge Discovery*, 31, 09 2017.

[28] Michael Hay, Vibhor Rastogi, Gerome Miklau, and Dan Suciu. Boosting the accuracy of differentially private histograms through consistency. *Proc. VLDB Endow.*, 3(1–2):1021–1032, sep 2010.

[29] Xi He, Ashwin Machanavajjhala, Cheryl J. Flynn, and Divesh Srivastava. Composing differential privacy and secure computation: A case study on scaling private record linkage. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1389–1406. ACM, 2017.

[30] Haryati Jaafar, Nordiana Mukahar, and Dzati Ramli. A methodology of nearest neighbor: Design and comparison of biometric image database. 12 2016.

[31] Parameswaran Kamalaruban, Victor Perrier, Hassan Jameel Asghar, and Mohamed Ali Kaafar. Not all attributes are created equal: -private mechanisms for linear queries. *Proceedings on Privacy Enhancing Technologies*, 2020(1):103–125, 2020.

[32] Jong Seon Kim and Yon Chung. Differentially private and skew-aware spatial decompositions for mobile crowdsensing. *Sensors*, 18:3696, 10 2018.

[33] Jong Wook Kim, Kennedy Edemacu, Jong Seon Kim, Yon Dohn Chung, and Beakcheol Jang. A survey of differential privacy-based techniques and their applicability to location-based services. *Computers Security*, 111:102464, 2021.

[34] Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., USA, 1998.

[35] Yuka Komai, Yuya Sasaki, Takahiro Hara, and Shojiro Nishio. k nearest neighbor search for location-dependent sensor data in manets. *IEEE Access*, 3:942–954, 2015.

[36] Ashwin Machanavajjhala, Daniel Kifer, John Abowd, Johannes Gehrke, and Lars Vilhuber. Privacy: Theory meets practice on the map. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, page 277–286, USA, 2008. IEEE Computer Society.

[37] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '07, page 94–103, USA, 2007. IEEE Computer Society.

[38] Ilya Mironov, Omkant Pandey, Omer Reingold, and Salil Vadhan. Computational differential privacy. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009*, pages 126–142, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[39] Ben Niu, Qinghua Li, Xiaoyan Zhu, Guohong Cao, and Hui Li. Achieving k-anonymity in privacy-aware location-based services. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 754–762, 2014.

[40] D.J. O'Neil. *Nearest Neighbors Problem*, pages 783–787. Springer US, Boston, MA, 2008.

[41] Nicolas Papernot, Martín Abadi, Úlfar Erlingsson, Ian Goodfellow, and Kunal Talwar. Semi-supervised knowledge transfer for deep learning from private training data, 2016.

[42] Aniket Pingley, Wei Yu, Nan Zhang, Xinwen Fu, and Wei Zhao. Cap: A context-aware privacy protection system for location-based services. *2009 29th IEEE International Conference on Distributed Computing Systems*, pages 49–57, 2009.

[43] Vincent Primault, Sonia Ben Mokhtar, Cedric Lauradoux, and Lionel Brunie. Differentially private location privacy in practice, 2014.

[44] C. Procopiuc, T. Yu, E. Shen, D. Srivastava, and G. Cormode. Differentially private spatial decompositions. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 20–31, Los Alamitos, CA, USA, apr 2012. IEEE Computer Society.

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

[46] Jens Rauch, Iyiola E. Olatunji, and Megha Khosla. Achieving differential privacy for k-nearest neighbors based outlier detection by data partitioning. *CoRR*, abs/2104.07938, 2021.

[47] Robert Sedgewick and Kevin Wayne. *Algorithms, 4th Edition.* Addison-Wesley, 2011.

[48] Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk. Nearest-neighbor methods in learning and vision. *IEEE Trans. Neural Networks*, 19(2):377, 2008.

[49] Pravin Shankar, Vinod Ganapathy, and Liviu Iftode. Privately querying location-based services with sybilquery. In *Proceedings of the 11th International Conference on Ubiquitous Computing*, UbiComp '09, page 31–40, New York, NY, USA, 2009. Association for Computing Machinery.

[50] Manolis Terrovitis. Privacy preservation in the dissemination of location data. *SIGKDD Explorations*, 13:6–18, 08 2011.

[51] Hien To, Liyue Fan, and Cyrus Shahabi. Differentially private h-tree. In *Proceedings of the 2nd Workshop on Privacy in Geographic Information Collection and Analysis*, GeoPrivacy'15, New York, NY, USA, 2015. Association for Computing Machinery.

[52] Benjamin Weggenmann and Florian Kerschbaum. Differential privacy for directional data. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 1205–1222, New York, NY, USA, 2021. Association for Computing Machinery.

[53] Zhuolun Xiang, Bolin Ding, Xi He, and Jingren Zhou. Linear and range counting under metric-based local differential privacy. In *2020 IEEE International Symposium on Information Theory (ISIT)*, page 908–913. IEEE Press, 2020.

[54] Yonghui Xiao, Li Xiong, and Chun Yuan. Differentially private data release through multidimensional partitioning. In *Secure Data Management*, 2010.

[55] Dejun Yang, Xi Fang, and Guoliang Xue. Truthful incentive mechanisms for k-anonymity location privacy. In *2013 Proceedings IEEE INFOCOM*, pages 2994–3002, 2013.

[56] Yuqing Zhu, Xiang Yu, Manmohan Chandraker, and Yu-Xiang Wang. Private-knn: Practical differential privacy for computer vision. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11851–11859, 2020.