Coresets for differentially private k-means clustering and applications to privacy in mobile sensor networks

Coresets for Differentially Private K-Means Clustering and Applications to Privacy in Mobile Sensor Networks

ABSTRACT

Mobile sensor networks are a great source of data. By collecting data with mobile sensor nodes from individuals in a user community, e.g. using their smartphones, we can learn global information such as traffic congestion patterns in the city, location of key community facilities, and locations of gathering places. Can we publish and run queries on mobile sensor network databases without disclosing information about individual nodes?

Differential privacy is a strong notion of privacy which guarantees that very little will be learned about individual records in the database, no matter what the attackers already know or wish to learn. Still, there is no practical system applying differential privacy algorithms for clustering points on real databases. This paper describes the construction of small coresets for computing k-means clustering of a set of points while preserving differential privacy. As a result, we give the first k-means clustering algorithm that is both differentially private, and has an approximation error that depends sub-linearly on the data's dimension d. Previous results introduced errors that are exponential in d.

We implemented this algorithm and used it to create differentially private location data from GPS tracks. Specifically our algorithm allows clustering GPS databases generated from mobile nodes, while letting the user control the introduced noise due to privacy. We provide experimental results for the system and algorithms, and compare them to existing techniques. To the best of our knowledge, this is the first practical system that enables differentially private clustering on real data.

1. INTRODUCTION

Today city planners, research institutes and individuals are analyzing large databases of GPS data generated from mobile devices such as smart phones. Our goal is to make the process private so that people can publish, research, and run queries on such databases without disclosing individuals' information. Consider the data generated by a collection of roving sensor nodes attached to human-driven cars, smart-phone users, or self-driving cars. The GPS information gathered collectively over time contains important information about the activity patterns of invidividuals (for example, where they live, where they work, how they spend their leisure time, etc) as well as city-scale patterns (for example, how does congestion vary during the course of a day, where are the hot spots in the city, what is the impact of events on traffic, what is the distribution of work places, etc.) Collecting global statistics requires large amounts of accurate data, yet individual data may be subject to privacy considerations. For example, individual users may be comfortable submitting their location data collected during commute to work but may not be comfortable submitting their location data associated with where they live or play. We wish to create a method that allows individual mobile nodes to tune the privacy level of their data collection and empower them to contribute their data for global statistics and information, without compromising privacy. In this paper we present an algorithm for differential privacy in support of this goal. This algorithm allows users to tune their privacy levels by associating privacy levels with location regions. An individual user may set high privacy levels around where they live and low privacy levels along the main highway they use to commute to work.

Specifically, we focus on k-means clustering queries, one of the main tasks of machine learning, and guarantee strong privacy properties known as *differential privacy* on the GPS databases. The technique we utilize is called *private coresets*.

1.1 Differential Privacy

Differential privacy [9] is a robust, precise and mathematically rigorous notion of privacy, and it has emerged in a recent line of work that seeks private data analysis (See [4, 5, 22, 23, 20, 6, 3, 21, 10, 12, 8]). Differential privacy promises that very little about any specific individual can be learned in the data analysis process irrespective of the attacker's prior knowledge, information source and other datasets. Algorithm $\mathcal{A}$ is ε-differentially private if the following holds. When $\mathcal{A}$ is applied on any two databases that differ only on the details of one individual, the probability distributions of the two outputs are very similar, i.e., the probability of outputs lying in any specific single fixed set differs by a multiplicative factor $\exp(\varepsilon)$. In other words, the output is insensitive to a specific individual and depends only on the statistics of the database.

Here is an example explaining why differential privacy is strong and useful in the scenario of a mobile sensor network.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Suppose that we have a database P of n location records from n roving sensor nodes. Each record consists of a data point of the corresponding node as a pair of real numbers (latitude, longitude), i.e, the (x, y) coordinates. We wish to apply an algorithm $\mathcal{A}$ on our database P that will have the following property: an attacker will have the the algorithm's output $\mathcal{A}(P)$, would almost not be able to learn anything about a single node's data (x, y). Unlike the annonimity framework, this property must hold independently of the existing knowledge of the attacker, e.g.: other data sources about the users in P, non-noisy subsets of users in P, etc.

For a non-private deterministic or random algorithm $\mathcal{A}$, a sample attack is as the following. Suppose that the attacker wishes to know whether a node p visited one of the hospitals in a city. He already knows all other nodes' records in the database P, and concludes that the database can be either P if the user indeed visited one of the hospitals, and P' otherwise. Now, suppose that the owner of the database publishes the output $\mathcal{A}(P)$ of an algorithm $\mathcal{A}$ that was applied on the database. If the distribution of $\mathcal{A}(P)$ is different from the distribution of $\mathcal{A}(P')$, the attacker can now tell that the original database is P and not P', and thus p actually visited the hospital.

We wish to develop differentially privacy algorithms for mobile sensor networks that not only have the strong *privacy* promise but also the *utility* property, i.e. we learned the desired statistics about the database. For example, if P is a location database recording tracks of sensor nodes in a city, we may learn by applying a clustering algorithm $\mathcal{A}$ on P that what are the typical tracks in the city. In this paper we use the relaxed version of pure differential privacy, known as (ε, δ)-differential privacy [8]. It is a relaxation of pure differential privacy where the privacy guarantee needs to be satisfied only for events whose probability is at least δ.

1.2 Private Coresets

Unlike k-anonymity, differential private algorithms usually do not output databases, but rather answer statistical queries about the data. In every query call that is answered via the algorithm (the "curator") there is some leakage of information. After a small number of queries, the allowed bound on the information leakage is reached and the curator refuses to answer additional queries. This makes differential private algorithms impractical for people trying to analyze the databases.

Coresets were introduced in [2]. For a set Q of possibly an infinite number of queries, and a database P, an α-coreset C is a database such that the answer to every query $q \in Q$ on P can be approximated up to an α-error, i.e., $(1-\alpha)\cdot q(P) \leq q(C) \leq (1+\alpha)\cdot q(P)$. If the coreset C is much smaller than P, we can solve optimization problems with respect to Q much faster, by running existing (possibly inefficient) algorithms on the reduced set C. Coresets for different types of queries have been the subject of many recent papers, such as k-medians ([7, 13, 15, 16]), k-means ([13, 15, 16]), k-centers ([18]), etc. Using the merge-and-reduce technique, it can also be shown that such an off-line coreset construction implies an on-line construction that can be computed using a small amount of memory and in one pass over the data ("streaming") [17, 14].

The private coresets scheme was defined in [12], as ε-differentially private algorithms that recieve a database P as the input and output an α-coreset C. This output is called ε-private α-coreset, or (α, ε)-*private coreset*, where α measures the utility of the output and ε denotes the privacy leakage. The user can then answer an unbounded number of queries on the released (sanitized) private coreset, and solve optimization problems by applying machine learning algorithms on the private coreset, without introducing additional privacy leakage and without an curator.

Intuitively, there should not be a connection between coresets and privacy. While the main objective in (non-private) coresets is minimizing the number of points in the coreset $C = A(P)$, this is not the main objective for private coresets — large coresets are OK as long as differential privacy is preserved. It is possible to construct a private coreset scheme with a number of points that is comparable to that of non-private schemes by first applying a *private* coreset scheme on the input, and then applying on its outcome a (regular, *non-private*) coreset scheme where the number of points in the output coreset is small. The challenge and quality measure of constructing private coresets is thus not the size of the output coreset, but rather the added noise to P that should be small enough to keep the approximation error α small and large enough to provide ε-privacy of the records in P.

The reason why coresets techniques are relevant for privacy is because they are both sensitive to outliers: in the context of coresets, outliers should be chosen with high probability as representatives. In the context of differential privacy, outliers require more additive noise to hide them, unlike large clusters of points with a mean that won't change significantly by moving of a single point. In both cases, the challenge is to measure how much each point is isolated from the other points and prove that the sum of this measure is small for every given input set.

1.3 Our Contributions

Our main contributions are:

1. The first (ε, δ)-differentially private algorithm that computes a provable approximation of the k-means clustering problem, where the multiplicative and additive errors are sub-linear in the dimension d of the input points. See Theorem 3.6 for details.

2. An (ε, δ)-differentially private coreset construction that approximates the sum of squared distances from the input points to any k centers in $\mathbb{R}^d$. An unbounded number of k-centers queries can be evaluated on this sanitized coreset, with no additional information leakage or privacy loss.

3. An implementation and experimental results for the (ε, δ)-differentially private k-means algorithm.

4. An application enabling mobile sensor network nodes and adminstrators control the privacy-utility trade-off while applying the differentially private k-means clustering algorithm to massive GPS databases.

This paper is organized as follows. Section 2 describes briefly the related work. Section 3 presents our algorithm for private k-means clustering. Section 4 discusses implementation issues and presents performance results. Sections 5 and 6 show results from experiments with data collected by the NYC taxi cabs.

2. RELATED WORK

The algorithm in this paper builds on the work of [12]. They define the notion of *private coresets* and suggest such a coreset for the k-medians clustering problem in low-dimensional space. The advantage of the private coresets technique is that, unlike previous solutions to differential privacy, a sanitized database is published with some small information leakage ε (in the formal sense of ε-differential privacy). Then an unbounded number of queries can be applied on the published sanitized database with no additional information leakage. However, [12] suggests a coreset scheme with error exponentially in dimension d, which is very noisy.

Differential Privacy Much of the work on differential privacy is in an interactive scenario where a central authority (called a *curator*) answers a small number of specific queries. Since there is an ε-information leakage in *each* such query, after a small number of queries, no more queries are allowed and the curator rejects them without any answer. In our scenario, one would like to publish a "sanitized" version of the data, the publication of which preserves differential privacy. Such a database could be queried *infintely* without impacting privacy. A query to the sanitized database should return (approximately) the same answer as obtained for the same query on the *original* database.

Non-interactive differentially private sanitizations are studied in [10, 21, 3, 6]. The most related to our work is [6]. This paper proves the existence of differentially private "sanitized" databases for range queries, where the range space from which the queries are taken is of low VC dimension [24].

Differentially Private k-means Clustering The problem of differentially private k-means clustering was previously studied in [5, 22]. Their approaches are different and more restricted compared to our solution. [5] describes a private implementation of a specific heuristic, namely Lloyd's heuristic, which is not flexible enough to extend to other heuristics. The accuracy of the output is based on an assumption that the number of points in each cluster is large enough. [22] suggests an implementation of differentially private k-means clustering in the sample and aggregate framework. Meaningful results rely on repeated executions of the algorithm on random samples from the original database.

k-means Coresets Non-private k-means coresets have been well studied in [13, 15, 16]. While our major objective for private k-means coresets is to get better utility under the same privacy leakage ε, [13, 15] focus more on minimizing the coreset size. They show that a coreset of size that is polynomial in the number of clusters k and α gives an $(1\pm\alpha)$-approximation of the k-means clustering problem.

3. DIFFERENTIALLY PRIVATE K-MEANS CLUSTERING ALGORITHM

In this section we start with an informal overview of the algorithm (Section 3.1), then provide some background definitions and theorems (Section 3.2). and finally present the formal algorithm and its analysis (Section 3.3).

Throughout the paper, we use X^d to denote a discrete d-dimensional domain. The domain is a real unit cube quantized with grid size $1/(|X|-1)$. Formally speaking,

$$X^d = \{0, 1, 2, \cdots, |X|-1\}^d / (|X|-1)$$

3.1 Algorithm Overview

Here we give only an informal overview. The exact definitions and claims can be found in Section 3.3.

The input to the algorithm is a set P of n points in the d-dimensional domain X^d, an integer $k \geq 1$ for the number of clusters k and privacy parameters ε and δ. We note that it was proven in [12, Theorem 6] that private clustering of a general set of n points in $\mathbb{R}^d$ is impossible, even in low dimensional space, since the additive error depends on the diameter and number of possible distances between the points. Hence, the restriction of input points on a grid cannot be avoided. The output of the algorithm is a set C of k-centers that approximates the sum of squared distances to the optimal k-means of P with a multiplicative and additive error that depend sub-linearly in d.

In each iteration, the algorithm computes the center c of an approximately smallest ball that contains at least $3n/(8k)$ input points. The computation of c is done using a differentially private subroutine Small-Ball that is described in Theorem 3.5.

Next, we remove the $3n/(8k)$ points that are closest to the center c from the input. Intuitively, c represents these points and thus get a weight of $w = 3n/(8k)$. We then continue recursively on the remaining points until there are almost no input points. The last (small) set of remaining points is ignored. Note that the value of n is reduced in each iteration.

After $O(k \log n)$ iterations, we thus have $O(k \log n)$ weighted centers. This set of centers D is our private coreset. We then compute a (regular, non-private) k-mean approximation on D. That is, we compute a set C of k centers among the $O(k \log n)$ weighted points that minimizes the sum of squared distances to the points.

3.2 Background Definitions and Theorems

Differential privacy guarantees that no individual's record can be learnt from an outcome of an algorithm, independently of what a potential intruder already knows about.

Definition 3.1 (Differential Privacy [8]). *Databases $S_1 \in U^m$ and $S_2 \in U^m$ over a domain U are called* neighboring *if they differ in exactly one entry. A randomized algorithm $\mathcal{A}$ is (ε, δ)-differentially private if for all neighboring databases $S_1, S_2 \in U^m$, and for all sets F of outputs, the probability that $\mathcal{A}(S_1) \in F$ is at most*

$$\Pr[\mathcal{A}(S_1) \in F] \leq \exp\varepsilon \cdot \Pr[\mathcal{A}(S_2) \in F] + \delta.$$

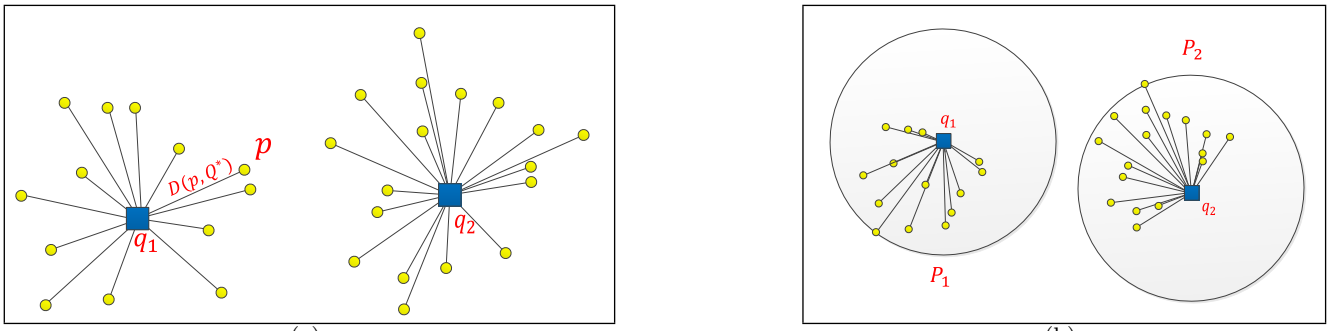
A differentially private algorithm can be constructed by other differentially private algorithms and non-private randomized computation. It is stated formally as the following Composition theorem.

Theorem 3.2 (Composition [11]). *Let $\mathcal{A}(\cdot)$ be any non-private randomized computation, let $\mathcal{A}_0(\cdot)$ be $(\varepsilon_0, \delta_0)$-differentially private, and let $\mathcal{A}_1(\cdot, \cdot)$ be a parameterized algorithm such that $A_1(C_0, \cdot)$ is $(\varepsilon_1, \delta_1)$-differentially private for all C_0. Then the following hold:*

(i) *Algorithm $\mathcal{B}$ that on input P, computes $C_0 \leftarrow \mathcal{A}_0(P)$, then $C \leftarrow \mathcal{A}(C_0)$ and outputs C is $(\varepsilon_0, \delta_0)$-differentially private.*

(ii) *Algorithm $\mathcal{B}$ that on input P, computes $C_0 \leftarrow \mathcal{A}_0(P)$, then $C_1 \leftarrow \mathcal{A}_1(C_0, P)$ and outputs (C_0, C_1) is $(\varepsilon_0 + \varepsilon_1, \delta_0 + \delta_1)$-differentially private.*

For a set of centers $X \subseteq \mathbb{R}^d$ and a point $p \in \mathbb{R}^d$ we denote $\text{dist}(p, X) = \min_{x \in X} |p - x|$. For a finite set $Q \subseteq \mathbb{R}^d$, the



(a) (b)

Figure 1: (a) A set P of points (in yellow) and its 2-means $Q^* = \{q_1, q_2\}$ (blue squares). The distance between the point p and its nearest center in Q^* is denoted by $D(p, Q^*)$. (b) A 10-approximation $\tilde{Q} = \{q_1, q_2\}$ (blue squares) of the 2-means of P with additive error 5. That is, $\text{cost}(P, \tilde{Q}) \leq 10\text{cost}(P, Q^*) + 5$. The sets P_1 and P_2 are the corresponding partition of P into two clusters: P_1 contains the points that are closest to q_1, and P_2 contains the points that are closest to q_2.

sum of squared distances is defined as

$$\text{cost}(Q, X) := \sum_{p \in Q} \text{dist}^2(p, X).$$

If $|X| = \{x\}$ we denote $\text{cost}(Q, x) = \text{cost}(Q, \{x\})$ for simplicity.

For a *weighted set* $D = \{(p_1, w_1), \cdots, (p_n, w_n)\} \subseteq \mathbb{R}^d \times \mathbb{R}^+ \cup \{0\}$, the weighted cost is

$$\text{cost}(D, X) := \sum_{(p,w) \in D} w\text{dist}^2(p, X).$$

Definition 3.3 (Approximated k-Means)**.** *Let P be a finite set in $\mathbb{R}^d$. A set C^* of k centers (points) in $\mathbb{R}^d$ is called the k-means of P if it minimizes $\text{cost}(P, C')$ over every such set C'. A set $C \subseteq \mathbb{R}^d$ of k centers is a γ-approximation with an η additive error for the k-means of P if*

$$\text{cost}(P, C) = \gamma \cdot \text{cost}(P, C^*) + \eta.$$

An illustration of the approximated k-means location data is shown in Figure 1.

In our algorithm, we calculate the approximate k-means (Definition 3.3) from k-means private coresets, which is defined in Definition 3.4 adapted from [12].

Definition 3.4 (k-Means Private Coreset)**.** *Let P be a finite set in $\mathbb{R}^d$. Let C^* be the k-means of P. A $(\gamma, \eta, \varepsilon, \delta)$-private coreset scheme for k-means clustering is an algorithm $\mathcal{A}$ satisfying:*

- *Algorithm $\mathcal{A}$ preserves (ε, δ)-differentially privacy.*
- *Let $C_{\mathcal{A}}$ be the k-means of the coreset $\mathcal{A}(P)$. Then $C_{\mathcal{A}}$ is a γ-approximation with η additive error for k-means of P.*

Our algorithm is based on a subroutine that given a database $P \in X^d$ and an integer t, approximately returns the center of the smallest ball that contains at least t points from P. We use a private approximation algorithm to this problem, that was suggested recently by K. Nissim et al. [23] as the following.

Algorithm 1: PRIVATE-K-MEAN$(P, k, \varepsilon, \delta, \beta)$

Input: A set P of n points in the d-dimensional domain X^d, a constant integer $k \geq 1$, privacy parameters ε, δ, and error parameter β.

Output: A set C of k centers

$D \leftarrow \emptyset, \varepsilon' \leftarrow \frac{3\varepsilon}{8k \log n}, \delta' \leftarrow \frac{3\delta}{8k \log n}, \beta' \leftarrow \frac{3\beta}{8k \log n}$

$t_{\min} \leftarrow O(\frac{k\sqrt{d}}{\varepsilon'} \log(\frac{1}{\beta'}) \log(\frac{nd}{\beta'\varepsilon'\delta'}) \sqrt{\log(\frac{1}{\beta'\delta'})} \cdot 9^{\log^*(2|X|\sqrt{d})})$

while $|P| > t_{\min}$ **do**

  $n \leftarrow |P|$

  $\Delta \leftarrow O(\frac{1}{\varepsilon'} \log(\frac{1}{\beta'}) \log(\frac{n}{\beta'\epsilon'\delta'}) \cdot 9^{\log^*(2|X|\sqrt{d})})$

  $w \leftarrow \frac{3n}{8k}$

  $r_{\text{opt}} \leftarrow$ the radius of the smallest ball that contains $\frac{3n}{8k} + \Delta$ points from P

  $c \leftarrow$ the center of a ball that contains at least $\frac{3n}{8k}$ points from P, whose radius is $r \leq \alpha r_{\text{opt}}$, where $\alpha = \sqrt{\log(n)}$. The step is calculated by the (ε', δ')-private SMALL-BALL subroutine with $t = \frac{3n}{8k} + \Delta$. See Theorem 3.5.

  $D \leftarrow D \cup \{(c, w)\}$

  $G \leftarrow$ the union of the closest w points to c in P

  $P \leftarrow P \setminus G$

$C \leftarrow$ an $O(1)$-approximation to the k-means of $\bigcup_{(c,w) \in D} c$ with no additive error; See Definition 3.3

return C

Theorem 3.5 (Private Center [23])**.** *Let P be a set of n points in the d-dimensional domain X^d. Let $n, t, \beta, \varepsilon, \delta$ be s.t.*

$$t \geq O(\frac{\sqrt{d}}{\varepsilon} \log(\frac{1}{\beta}) \log(\frac{nd}{\beta\varepsilon\delta}) \sqrt{\log(\frac{1}{\beta\delta})} \cdot 9^{\log^*(2|X|\sqrt{d})})^1$$

Denote r_{opt} as the radius of the smallest ball in X^d containing at least t points from P. Then there is an (ε, δ)-private

[1] Given n, $\log^*(n)$ denotes the number of times that the function $\log(x)$ must be iteratively applied before the result is less or equal to 1, i.e., $\log^*(n) = 1 + \log^*(\log n)$

SMALL-BALL *algorithm that runs in polynomial time and returns a center c such that, with probability at least $1-\beta$, there is a ball of radius $r \le \alpha r_{\text{opt}}$ that is centered at c and contains at least $t-\Delta$ points from P, where $\alpha = O(\sqrt{\log n})$ and*

$$\Delta = O(\frac{1}{\varepsilon}\log(\frac{1}{\beta})\log(\frac{n}{\beta\epsilon\delta})\cdot 9^{\log^*(2|X|\sqrt{d})})$$

3.3 Problem Statement and Proofs

In this section we give the algorithm's pseudocode, state the properties of the algorithm in the form of a theorem, and then give the proofs for the correctness and privacy guarantees of our algorithm. Algorithm 1 is the pseudocode of PRIVATE-K-MEAN.

Algorithm 1 is structured as the following. We initialize the private coreset D to be empty (Line 1). Then we divide input parameters $\beta, \varepsilon, \delta$ by the number of iterations to calculate failure parameter β', privacy parameters ε' and δ' for a single iteration (Line 2). After that, we calculate a minimum threshold of database size $t_{\min}$ determining when to stop iterating (Line 3). In every iteration (Line 4 - 11), we call the SMALL-BALL subroutine (Line 9) with inputs calculated from Line 5-7, and output a new data (c, w) to be added to the private coreset D (Line 10). We then remove the $3n/(8k)$ points that are closest to c from the database (Line 11- 12). After all the iterations, we can calculate the approximate k-means from the private coreset D (Line 13).

Our main result is stated in the following theorem.

Theorem 3.6. *Let P be a set of n points in the d-dimensional domain X^d. Let $n, k, \beta, \varepsilon, \delta$ be s.t. $n > t_{\min}$ where $\varepsilon' = \frac{3\varepsilon}{8k\log n}, \delta' = \frac{3\delta}{8k\log n}, \beta' = \frac{3\beta}{8k\log n}$, and*

$$t_{\min} = O(\frac{k\sqrt{d}}{\varepsilon'}\log(\frac{1}{\beta'})\log(\frac{nd}{\beta'\varepsilon'\delta'})\sqrt{\log(\frac{1}{\beta'\delta'})}\cdot 9^{\log^*(2|X|\sqrt{d})}).$$

Let C be the output of a call to PRIVATE-K-MEAN$(P, k, \varepsilon, \delta, \beta)$. *Then the following hold.*

- ***(i)*** *The algorithm* PRIVATE-K-MEAN *is (ε, δ)-differentially private; See Definition 3.1.*
- ***(ii)*** *With probability at least $1-\beta$, the set C is a γ-approximation for the k-means of P with additive error η, where $\gamma = O(k\log n)$, and $\eta = t_{\min}$. See Definition 3.3.*

The errors γ and η in Theorem 3.6 depend sub-linearly on the data's dimension d, which improves from exponential on d in previous literature [12]. We provide a brief explanation here. First, the multiplicative error γ is independent of d. As for the additive error η, only considering parameter d, we have $\eta = O(C_0\sqrt{d}\log(C_1 d)\cdot 9^{\log^*(C_2\sqrt{d})})$ where C_0, C_1, C_2 are expressions of other parameters. When d is sufficiently large,

$$\begin{aligned}\log^*(C_2\sqrt{d}) &= 1+\log^*(\log(C_2\sqrt{d})) \\ &= 2+\log^*(\log\log(C_2\sqrt{d})) \\ &< 2+\log\log(C_2\sqrt{d})\end{aligned} \tag{1}$$

Therefore,

$$\begin{aligned}\eta &= O(C_0\sqrt{d}\log(C_1 d)\cdot 9^{\log^*(C_2\sqrt{d})}) \\ &= o(C_0\sqrt{d}\log(C_1 d)\log(C_2\sqrt{d})) \\ &= o(d).\end{aligned} \tag{2}$$

Proof. ***(i, Privacy Guarantee)*** Since $|P|$ is reduced by a factor of $(1-\frac{3}{8k})$ in each "while" iteration, there are at most $\frac{8k\log n}{3}$ iterations. Every iteration outputs a new pair (c, w) to compose the private coreset D. Because the value of w in each iteration is the same for each database of size n, we only need to consider the value of c, which is the output of the SMALL-BALL subroutine.

Every iteration except the first one can be regarded as a parameterized algorithm, which takes the original database, and all the previous pairs of (c, w) as inputs. Notice that in each iteration, the selected removed points only depend on the choice of c. If two neighboring database P and P' differ by a single entry, after one iteration, they will differ by at most a single entry. By induction, before every iteration, P and P' are neighbors of each other. Therefore we can claim every iteration is (ε', δ')-differentially private. Then by composition theorem (Theorem 3.2(ii)), the algorithm for outputting D (up to Line 12 of Algorithm 1) is $(\varepsilon', \delta')\times\frac{8k\log n}{3} = (\varepsilon, \delta)$-private.

Since the returned set C is computed on the "sanitized" private coreset D. According to the composition theorem (Theorem 3.2(i)), the entire Algoithm 1 is (ε, δ)-private.

(ii, Utility Guarantee) Table 1 is a table of notations that are useful in the proof.

Table 1: Notations in the proof

X^*	A set of k centers that minimizes $\text{cost}(P, X)$ of the original database P over every such X.
P_i, c_i, w_i, G_i	The value of P, c, w and G during the ith iteration.
P_i^*	The $\lceil 3\lvert P_i\rvert/4\rceil$ points $p \in P_i$ with the smallest value $\text{dist}(p, X^*)$.
P_{e}	All the eliminated points $\bigcup G_i$.
P_{r}	All the remaining points in P after the last iteration, i.e., $P \setminus P_{\text{e}}$.

We give the entire proof step by step by proving the following facts.

- Step 1: The entire algorithm has the desired output with probability $1-\beta$.
- Step 2: $\text{cost}(G_i, c_i) \le \alpha^2\text{cost}(P_i^*, X^*)$.
- Step 3: $\sum_{i=1}^{|D|}\text{cost}(G_i, c_i) \le O(\alpha^2 k)\cdot\text{cost}(P, X^*)$.
- Step 4: $\text{cost}(P, C) \le O(\alpha^2 k)\cdot\text{cost}(P, X^*) + O(t_{\min})$.

Step 1 Notice that every iteration satisfies the precondition $n > t_{\min}$. Therefore, we have

$$\begin{aligned}\frac{3n}{8k}+\Delta &> \frac{3t_{\min}}{8k} \\ &= O(\frac{\sqrt{d}}{\varepsilon'}\log(\frac{1}{\beta'})\log(\frac{nd}{\beta'\varepsilon'\delta'})\sqrt{\log(\frac{1}{\beta'\delta'})}\cdot 9^{\log^*(2|X|\sqrt{d})})\end{aligned} \tag{3}$$

This exactly matches the SMALL-BALL subroutine's precondition with inputs $\varepsilon', \delta', \beta'$. See Theorem 3.5. As a result, with probability $1-\beta'$, every iteration will output the desired c. Because there are at most $\frac{8k\log n}{3}$ iterations, Algorithm 1 has the desired output with probability $1-\frac{8k\log n}{3}\beta' = 1-\beta$.

Step 2 By the pigeonhole principle there must be a center $x \in X^*$ that serves $m \geq |P_i^*|/k$ points in P_i^*, i.e., at least $1/k$ fraction of the points in P_i^* have x as their closest center in X^*. Using Markov inequality, half of them have distance of at most $\sqrt{\frac{2\text{cost}(P_i^*, X^*)}{m}}$ to x. We conclude that there is a ball of radius $\sqrt{\frac{2\text{cost}(P_i^*, X^*)}{m}}$ that contains at least $m/2 \geq |P_i^*|/2k = 3|P_i|/8k$ points from P_i. By definition of r_{opt} in Line 7 of Algorithm 1,

$$r_{\text{opt}} \leq \sqrt{\frac{2\text{cost}(P_i^*, X^*)}{m}} \tag{4}$$

$$\frac{m}{2} r_{\text{opt}}^2 \leq \text{cost}(P_i^*, X^*) \tag{5}$$

Also we have

$$|G_i| = \frac{3|P_i|}{8k} = \frac{|P_i^*|}{2k} \leq \frac{m}{2} \tag{6}$$

Combining the result of (5) and (6),

$$\text{cost}(G_i, c_i) \leq |G_i| \cdot (\alpha r_{\text{opt}})^2 \leq \alpha^2 \text{cost}(P_i^*, X^*) \tag{7}$$

Step 3 Because each iteration contributes to one pair of (c, w) in D, $|D| = O(k \log n)$.

We order the points in P by $P = \{p_1, \cdots, p_n\}$, such that $\text{dist}(p_a, X^*) \leq \text{dist}(p_b, X^*)$ for every $1 \leq a < b \leq n$, where ties are broken arbitrarily. Let

$$\begin{aligned} U_i &= \left\{p_1, \cdots, p_{n-|P_i|}\right\}, \\ V_i &= \left\{p_{n-|P_i|+1}, \cdots, p_{n-|P_i|+|P_i^*|}\right\} \end{aligned} \tag{8}$$

During the first $(i-1)$ "while" iterations, an overall of $n-|P_i|$ points were removed from P. Hence,

$$\begin{aligned} |(U_i \cup V_i) \cap P_i| &\geq |U_i| + |V_i| - (n - |P_i|) \\ &= |V_i| = |P_i^*|. \end{aligned} \tag{9}$$

We thus have $U_i \cup V_i \supseteq P_i^*$. The set V_i contains the $|V_i| = |P_i^*|$ points $p \in U_i \cup V_i$ with the largest values $\text{dist}(p, X^*)$. Hence, $\text{cost}(P_i^*, X^*) \leq \text{cost}(V_i, X^*)$. Combining (7) with (9) yields

$$\text{cost}(G_i, c_i) \leq \alpha^2 \text{cost}(P_i^*, X^*) \leq \alpha^2 \text{cost}(V_i, X^*) \tag{10}$$

Summing (10) over all the $O(k \log n)$ iterations, we obtain

$$\sum_{i=1}^{|D|} \text{cost}(G_i, c_i) \leq \alpha^2 \sum_{i=1}^{|D|} \text{cost}(V_i, X^*) \tag{11}$$

The last point index in the set V_i is $n - |P_i| + |P_i^*| = n - |P_i|/4$, and the first index in V_{i+4k} is $n - |P_{i+4k}| + 1$. Since

$$\begin{aligned} n - |P_{i+4k}| + 1 &\geq n - |P_i| \cdot (1 - \frac{3}{8k})^{4k} \\ &\geq n - |P_i|/4 \text{ (when } k \geq 1) \end{aligned} \tag{12}$$

every point appears in $O(k)$ sets in the sequence $V_1, V_2, \cdots$. Hence,

$$\sum_{i=1}^{|D|} \text{cost}(V_i, X^*) \leq O(k) \cdot \sum_{i=1}^{|D|} \text{cost}(P, X^*). \tag{13}$$

Plugging (13) in (11) yields

$$\sum_{i=1}^{|D|} \text{cost}(G_i, c_i) \leq O(\alpha^2 k) \cdot \text{cost}(P, X^*) \tag{14}$$

Step 4 We use the weak triangle inequality [14, Lemma 7.1] stating that for every $p, q \in \mathbb{R}^d$ and a closed set $C \subseteq \mathbb{R}^d$ we have

$$|\text{dist}^2(p, C) - \text{dist}^2(q, C)| \leq \frac{12|p-q|_2^2}{\lambda} + \frac{\lambda}{2} \text{dist}^2(p, C). \tag{15}$$

In our case, by letting p' denote the associated c_i to $p \in P_e$ when p is eliminated, and using $\lambda = 1/2$, this implies

$$\begin{aligned} &|\text{cost}(P_e, C) - \text{cost}(D, C)| \\ &= |\sum_{p \in P_e} \text{dist}^2(p, C) - \sum_{p \in P_e} \text{dist}^2(p', C)| \\ &\leq \sum_{p \in P_e} (\frac{12|p-p'|_2^2}{\lambda} + \frac{\lambda}{2} \text{dist}^2(p, C)) \\ &= 24 \sum_{i=1}^{|B|} \text{cost}(G_i, c_i) + \frac{1}{4} \text{cost}(P_e, C). \end{aligned} \tag{16}$$

Hence,

$$\frac{3\text{cost}(P_e, C)}{4} \leq \text{cost}(D, C) + 24 \sum_{i=1}^{|D|} \text{cost}(G_i, c_i). \tag{17}$$

By the optimality of C,

$$\text{cost}(D, C) \leq \text{cost}(D, X^*). \tag{18}$$

Similarly to (16),

$$|\text{cost}(P_e, X^*) - \text{cost}(D, X^*)| \leq 24 \sum_{i=1}^{|D|} \text{cost}(G_i, c_i) + \frac{1}{4} \text{cost}(P_e, X^*). \tag{19}$$

Hence,

$$\text{cost}(D, X^*) \leq \frac{5}{4} \text{cost}(P_e, X^*) + 24 \sum_{i=1}^{|B|} \text{cost}(G_i, c_i) \tag{20}$$

Combining (14), (17), (18) and (20), we get

$$\begin{aligned} \text{cost}(P_e, C) &\leq \frac{4}{3} (\text{cost}(D, C) + 24 \sum_{i=1}^{|B|} \text{cost}(G_i, c_i)) \\ &\leq \frac{4}{3} \text{cost}(D, X^*) + 32 \sum_{i=1}^{|B|} \text{cost}(G_i, c_i) \\ &\leq \frac{5}{3} \text{cost}(P_e, X^*) + 64 \sum_{i=1}^{|B|} \text{cost}(G_i, c_i) \\ &\leq O(1) \text{cost}(P_e, X^*) + O(\alpha^2 k) \cdot \text{cost}(P, X^*) \\ &= O(\alpha^2 k) \cdot \text{cost}(P, X^*) \end{aligned} \tag{21}$$

Notice that $\text{cost}(P, C) = \text{cost}(P_e, C) + \text{cost}(P_r, C)$, and X^d is bounded by the unit cube with diameter 1, we have $\text{cost}(P_r, C) = O(t_{\min})$. Thus combining with (21),

$$\text{cost}(P, C) \leq O(\alpha^2 k) \cdot \text{cost}(P, X^*) + O(t_{\min}) \tag{22}$$

The multiplicative error is $O(\alpha^2 k) = O(k \log n)$. The additive error is

$$O(t_{\min}) = O(\frac{k\sqrt{d}}{\varepsilon'} \log(\frac{1}{\beta'}) \log(\frac{nd}{\beta'\varepsilon'\delta'}) \sqrt{\log(\frac{1}{\beta'\delta'})} \cdot 9^{\log^*(2|X|\sqrt{d})})$$

□

4. IMPLEMENTATION AND PERFORMANCE EVALUATION

We implement the PRIVATE-K-MEAN algorithm in Python 2.7, and utilize third-party libraries including Numpy, Scipy, Pandas, and Scikit-learn. We first briefly introduce the SMALL-BALL subroutine that we use iteratively in the main algorithm. Then we discuss our attempts of implementation and a performance optimization for $d = 2$ situation. On one hand, this optimization is useful for mobile sensor network with location information and more generally GPS databases, because GPS data with form (latitude, longitude) are two dimensional. On the other hand, we expect our algorithm to significantly outperform the private coreset algorithm in [12], because our error is nearly-linear to $1/\varepsilon$ (See Section 3.3), while their error is linear to $1/\varepsilon^d$, i.e., quadratic to $1/\varepsilon$ when $d = 2$. Indeed, we can see the improvements shown in the experiment results (See Section 5.5). Finally, we analyze the time complexity of the PRIVATE-K-MEAN algorithm. All the time complexity expressions do not contain the data dimension d because our implementation is based on $d = 2$.

4.1 SMALL-BALL Subroutine

Our main PRIVATE-K-MEAN algorithm calls the SMALL-BALL subroutine introduced by Nissim et al. [23] iteratively, see Line 8 of the Algorithm 1. Given a set of n points and a target number t, the goal of SMALL-BALL algorithm is to approximately find a ball of minimal radius r_{opt} that contains at least t input points. The algorithm contains two steps. The first step GOOD-RADIUS finds $r = O(r_{\text{opt}})$ such that there is a ball of radius r that contains at least t input points. The second step GOOD-CENTER locates the ball given the radius r [23].

Nissim et al. proves the theoretical guarantees of SMALL-BALL algorithm in their paper, but does not provide runtime analysis nor implementation suggestion. Therefore, we implement the subroutine from scratch. We identify the bottleneck in the first step GOOD-RADIUS. Algorithm 2 provides the pseudocode of GOOD-RADIUS. Intuitively, for every radius r and every data x, we calculate how many points in P are contained in a ball of radius r around x (Line 1). Then for every r, we approximately calculate the maximum number of points contained in a ball of radius r (Line 2). After that, we reduce it to a quasi-concave problem, and privately find the smallest r such that there exists a ball of radius r containing at least t points (Line 3-4). More details can be found in [23].

4.2 Brute Force Implementation

The most time consuming part of Algorithm 2 is specifically line 1 and 2. First we describe a straightforward brute force implementation. The goal here is to calculate $L(r, P)$ for every single r. The implementation details are the following.

1. For every single point $x_i \in P$, and every radius $r \in [1, |X|]$, we count how many other points in P are within a ball of radius r around x_i. Bounding the result by t, we get $\overline{B}_r(x_i, P)$. Calculating a single $\overline{B}_r(x_i, P)$ takes $O(n)$ time, where n is the size of P. Therefore, calculating for all values of x_i and r takes $O(n) \times n \times |X| = O(n^2|X|)$ time.

2. For every single r, we choose t biggest $\overline{B}_r(x_i, P)$ among all $x_i \in P$, and take the average, which is $L(r, P)$. Utilizing the order statistics algorithm, this step takes $O(n|X|)$ time.

To sum up, Algorithm 2 (GOOD-RADIUS$(P, t, \beta, \varepsilon, \delta)$) takes $O(n^2|X|) + O(n|X|) = O(n^2|X|)$ time in total.

Algorithm 2: GOOD-RADIUS$(P, t, \beta, \varepsilon, \delta)$[23]

Input: A set P of n points in the d-dimensional domain X^d, parameter t, failure parameter β, and privacy parameter ε, δ.

1 For $x \in X^d$ and $0 \le r \in \mathbb{R}$ let $B_r(x, P)$ denote the number of input points contained in a ball of radius r around x, and let $\overline{B}_r(x, P) = \min\{B_r(x, P), t\}$. For $r < 0$ we say that $B_r(x, P) = \overline{B}_r(x, P) = 0$.

2 For $r \in \mathbb{R}$ define $L(r, P) = \max_{\text{distinct } x_1, \cdots, x_t \in P} \{\frac{\overline{B}_r(x_1, P) + \cdots + \overline{B}_r(x_t, P)}{t}\}$.

3 Denote $\Delta = 8^{\log^*(|X|\sqrt{d})} \frac{144 \log^*(|X|\sqrt{d})}{\varepsilon} \log(\frac{12 \log^*(|X|\sqrt{d})}{\beta\delta})$, and define the quality function $Q(r, P) = \frac{1}{2} \min\{t - L(r-1, P), L(r, P) - t + \Delta\}$.

4 Choose and return $z \in \{0, 1, 2, \cdots, \lceil |X|\sqrt{d} \rceil\}$ using algorithm RecConcave [4] with the quality function Q, quality promise $\frac{\Delta}{4}$, and approximation parameter $\frac{1}{2}$.

4.3 Performance Optimization When $d = 2$

Calculating $\overline{B}_r(x_i, P)$ for all pairs of (x_i, r) is time consuming. In order to improve the performance, we utilize faster approximation algorithms that count the number of neighbors around a point. Notice that before applying an algorithm, we need to ensure that it does not break the contract of differential privacy, i.e., the privacy guarantee in Theorem 3.6 still holds. This criteria eliminates some approximation techniques, such as ε-net. ε-net is $O(\frac{1}{\varepsilon})$ points random sampled from the original database, and has the following property: with high probability, any ball that covers at least α fraction of input points must also cover at least α fraction of the sample. [19] Therefore, the number of neighbors in ε-net can be used to approximate the number of neighbors in the original database. However, we can not use ε-net in our implementation because random sampling does not preserve differential privacy. Fortunately, we find another faster implementation for $d = 2$ case. In this section, we will talk about its details and prove that it preserves differential privacy.

The idea is to use L_∞ distance, or Chebyshev distance to approximate the Euclidean distance between two points. For any two points (x_1, y_1) and (x_2, y_2) in the two dimensional space, the L_∞ distance between them is $D_\infty = max(|x_2 - x_1|, |y_2 - y_1|)$, and the Euclidean distance between them is $D_{\text{Euclidean}} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. It is easy to see $D_\infty \le D_{\text{Euclidean}} \le \sqrt{2} D_\infty$ so this is a constant factor ap-

proximation. Calculating the L_∞ distance is faster than calculating the Euclidean distance.

The implementation steps of Line 1 and 2 of GOOD-RADIUS pseudocode are as the following:

1. Create a matrix $C_{|X|\times|X|}$, in which $C(i,j)$ is the number of times (i,j) appear in the database P. This takes $O(n)$ time.

2. Calculate an aggregation matrix $A_{|X|\times|X|}$, in which $A(i,j) = \sum_{a=1}^{i}\sum_{b=1}^{j} C(a,b)$. One single element $A(i,j)$ can be calculated in constant time by a recursion $A(i,j) = A(i-1,j)+A(i,j-1)-A(i-1,j-1)+C(i,j)$. Therefore, this step takes $O(|X| \times |X|) = O(|X|^2)$ time.

3. The aggregation matrix calculated in the previous step can help us calculate the sum of entries in any submatrix of C in constant time, as illustrated in Figure 2. Notice that the number of neighbors around a point (i,j) within L_∞ distance d can be regarded as the sum of C's $(2d+1)\times(2d+1)$ submatrix centered at (i,j). In this way, we can calculate $\overline{B}_r(x_i,S)$ for a specific pair of (x_i,r) in constant time. Therefore, calculating for all values of x_i and r takes $O(1)\times n\times|X| = O(n|X|)$ time.

4. The same as step 2 in the brute force algorithm. For every single r, we choose t biggest $\overline{B}_r(x_i,P)$ among all $x_i \in P$, and take the average, which is $L(r,S)$. This step takes $O(n|X|)$ time.

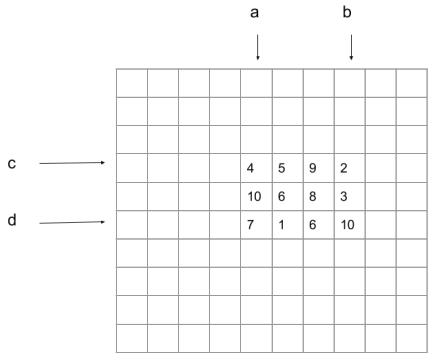


Figure 2: The sum of the submatrix can be calculated as $A(d,b)-A(d,a-1)-A(c-1,b)+A(c-1,a-1)$. It is a constant time computation.

To sum up, Algorithm 2 takes $O(n)+O(|X|^2)+O(n|X|) = O(|X|^2+n|X|)$ time in total. When we run experiments, we choose $|X|$ such that the total number of positions in the domain X^d approximately equals the size of database n. Hence, in the two dimensional case, $|X|^2 \approx n$. Compared to the brute force algorithm in Section 4.2, we improve from $O(n^{2.5})$ to $O(n^{1.5})$.

Note that the above technique maintains differential privacy. Different from ε-net, everything we calculate so far is deterministic. The only random bits in the GOOD-RADIUS algorithm is still the RecConcave algorithm in Line 4. Therefore, the privacy guarantee still holds.

4.4 Runtime Analysis

We identified the bottleneck of Algorithm 1 (PRIVATE-K-MEAN) in each iteration as the GOOD-RADIUS algorithm, and Section 4.3 have shown that its running time is $O(n^{1.5})$. According to Section 3.3, there are at most $O(k\log n)$ iterations, so the totally running time of calculating the private coreset D (up to Line 21 of Algorithm 1) is $O(k\log n)\times O(n^{1.5}) = O(kn^{1.5}\cdot\log n)$. The result only holds for two dimensional case.

5. EXPERIMENTS

We conduct experiments with a roving mobile sensor network's data with publicly available date: the New York City cab data from January 2015 [1]. In this section, we discuss how we set up the experiments, give the running time plots, the coreset size and utility results, and finally compare them to the literature.

5.1 Experiment Setup

The database records all yellow taxi trips (pickup location and destination) in New York City from January of 2015. It contains 12,748,986 entries. Each entry records the details about a single trip. Among all the fields, we are interested in *pickup_longitude* and *pickup_latitude* columns, which show where the taxis are picking up passengers, for two dimensional settings. We further add the *dropoff_longitude* and *dropoff_latitude* columns to see the running time of our algorithm in high dimensional settings. We assume that each pick up/drop off location is independent of other pick up/drop off locations, i.e., attackers cannot infer the information about one single trip from other trips. Under this assumption, it makes sense to apply differentially private algorithm on this GPS dataset, because differential privacy is useful to protect single entry.

The database is noisy, and some GPS coordinates do not lie in the New York City area or are even invalid. Therefore, we filter the database by letting the latitude be between 73.9°W and 74.1°W, the longitude be between 40.7°N and 40.9°N, which approximately outlines the area of New York City. After that, there are 11,840,734 remaining entries.

As suggested in Section 3, our PRIVATE-K-MEAN algorithm takes input from a discrete domain, so we need to discretize the data. Specifically, we map the 2D geographic coordinate space into an $|X|\times|X|$ grid, where $|X|\times|X|$ approximates the number of the locations in the database, Every GPS point is rounded to its nearest vertex in the grid.

The PRIVATE-K-MEAN algorithm's inputs include the database P with size n, number of clusters k, and privacy parameters ε,δ. As Dwork and Roth [11] suggests, it is dangerous to choose δ on the order of $1/n$. Therefore, we always choose δ to be much less than $1/n$ in our experiments. We adjust the database size n by taking the first n entries out of the original database. For every single combination of (n,k,ε), we run the algorithm 100 times and take the average in order to smooth out the noise created by randomness.

5.2 Running Time

We measure the algorithm's running time on a virtual machine in our OpenStack cloud. The virtual machine has 12 cores, 12GB of RAMs, and 16GB of root disk. The result for two dimensions is shown in Figure 3. Our algorithm processes 10^6 data points in about 40 seconds, and 10^7 data

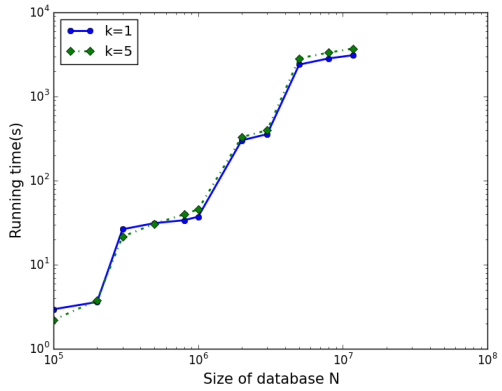


Figure 3: Running time vs. database size N under different number of clusters k in two dimensions. The privacy parameter $\varepsilon = 0.8$.

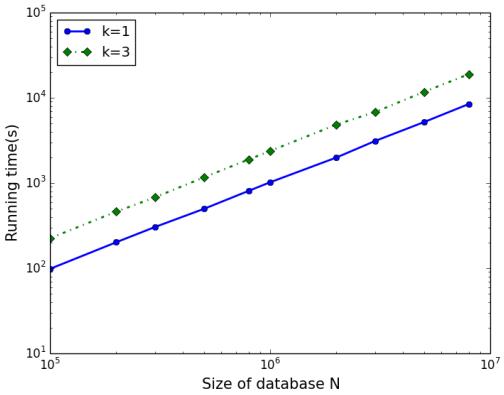


Figure 4: Running time vs. database size N under different number of clusters k in four dimensions. The privacy parameter $\varepsilon = 1$.

points in about one hour. The log of running time is approximately linear to the log of database size. By linear regression, the slope is 1.6 for $k = 1$ case, and 1.7 for $k = 5$ case. Therefore, the running time $t \propto n^{1.6-1.7}$. Another fact from the plot is that the running time for $k = 5$ case is larger than the $k = 1$ case, especially when database size n gets larger and there are more iterations. The results match the theoretical estimation of $t = O(kn^{1.5} \cdot \log n)$ in Section 4.4.

We conducted experiments for higher dimensional datasets. The results for four-dimensional data is shown in Figure 4. For high dimensional datasets, the optimization framework proposed in Section 4 is no longer applicable. We thus make use of a divide-and-conquer style approach by splitting the entire dataset into smaller sub-datasets, and running our algo- rithm on each sub-dataset. More specifi- cally, we set the privacy parameter $\epsilon = 1$, and for a dataset of size N, we divide it into a total of $\frac{N}{10000}$ sub-datasets of size 10000, and take the union of the outputs generated by each of the sub-datasets. Figure 4 shows that the log-scale plot of the running time is approximately linear in the log of database size, and the running time for $k = 1$ is comparable with the one in two dimensional case.

In Section 5.1, we discretize the data into an $|X| \times |X|$ grid. For convenience of implementation, we choose $|X|$ to be the minimum power of 2 such that $|X| \times |X| > n$. Therefore, $|X|$ will stay constant while n increases and this results in a relatively slow increase of the overall running time. However when n reaches a threshold that requires a larger $|X|$, the overall running time sees a jump. This phenomenon is illustrated in Figure 3.

5.3 Coreset Size

The relationship between the coreset size and the privacy parameter ε is shown in the right side of Figure 5. The coreset size is positively correlated to database size n, the privacy parameter ε and the number of clusters k. Also, under the same database and k, the coreset size has a linear relationship with respective to $\log \varepsilon$.

There is an interesting *staging phenomenon* in the plot, i.e., as ε increases, the coreset size sometimes remains the same for a while before starting to increase again. This is especially clear in the $k = 1$ plot. The result matches our Private-k-Mean pseudocode. According to line 9 of Algorithm 1, each iteration contributes to one point to the coreset. Hence the coreset size equals the number of iterations. If k is smaller, during each iteration, a larger portion of points are removed from the database (Line 6). As a result, when ε changes slightly (which causes the threshold $t_{\min}$ to change slightly (Line 2)), the number of iterations for the database size to reach that threshold is not affected.

5.4 Utility

In Theorem 3.6, we claim that the returned set C has both multiplicative and additive error. However, in practical setting, it is infeasible to measure both of them concurrently. Using the notation in Definition 3.3, we treat the difference between the cost of private clustering $\text{cost}(P, C)$ and the minimum cost $\text{cost}(P, C^*)$ as a multiplicative error, which is calculated by

$$\text{error} = \frac{\text{cost}(P, C) - \text{cost}(P, C^*)}{\text{cost}(P, C^*)}.$$

Because finding the absolute minimum cost, $\text{cost}(P, C^*)$, is an NP-hard problem, we use the k-means clustering module in the Scikit-learn library to approximately calculate it.

The experiment results are shown in the left side of Figure 5. For fixed values of N and k, the error becomes smaller when ε increases, which demonstrates the tradeoff between utility and privacy. However in the $k = 1$ plot, the negatively correlated relationship does not always hold in every small neighborhood. The error is the same or might fluctuate a bit when ε increases. One cause is that coreset size stays the same while ε increases (known as *staging phenomenon* mentioned in Section 5.3). The other reason is that when $k = 1$, the coreset size is very small (less than 10), which introduces large randomness.

For different combination of (k, N), we pick different ε to start with. Those are not arbitrary options. In fact, when ε is below a threshold, the code does not output meaningful results. We approximately pick those thresholds as the lowest values to start the experiments. Here are the problems when ε is too small:

- The size of the original database is already below $t_{\min}$, so there will be no iterations. See Line 4 of Algorithm 1 (Private-k-Mean).

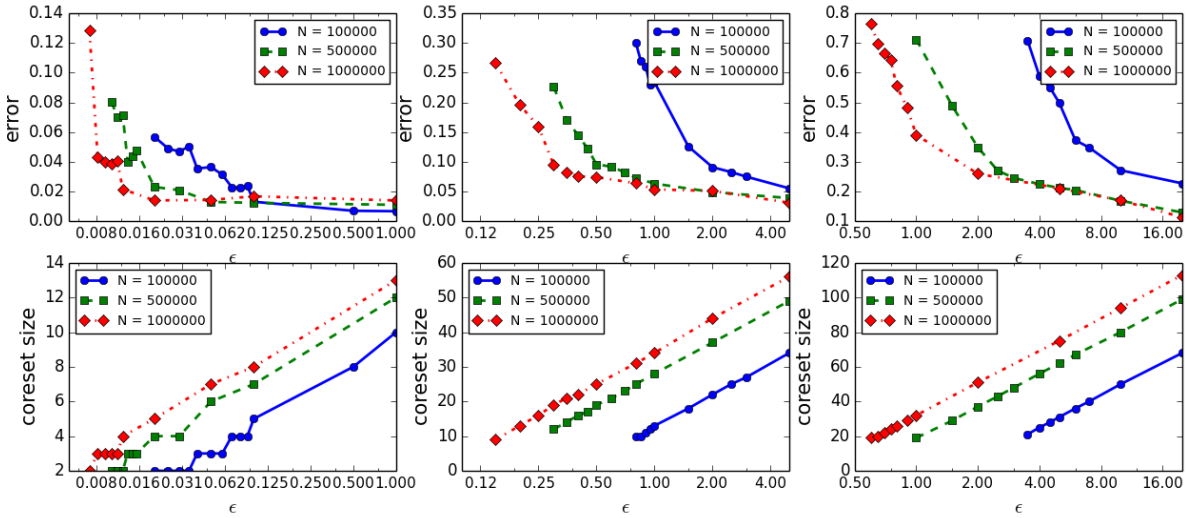


Figure 5: The top three subplots show the relationship between error and privacy parameter ε. The bottom three subplots show the relationships between coreset size and ε. The three rows represent 1-center, 5-centers and 10-centers clustering respectively. In every subplot, different lines represent different database sizes. Notice that the x-axes are log scale.

- RecConcave algorithm in Line 4 of the Algorithm 2 (GOOD-RADIUS) halts because the noise is too big [4].

Table 2: Minimum feasible ε for different (k, N)

k \ N	100,000	500,000	1,000,000
1	0.02	0.01	0.007
5	0.8	0.3	0.15
10	3.5	1.0	0.6

Table 2 shows the minimum feasible ε for different combinations of (k, N). It suggests that in practice, we prefer large database size N and a small number of clusters k because in that case, the mimimum ε is smaller, and there is less information leakage. However, our code will barely protect privacy if N is too small and k is too large. For example, when $(k, N) = (10, 10^5)$, the minimum ε equals 3.5, which is too large.

Figure 6 shows the relationship between error and the number of clusters k. As there are more clusters, the error becomes larger. Also, the increasing rate of error is larger for smaller privacy parameter ε.

5.5 Comparison with Previous Literature

There is no practical evaluation or benchmark on a differentially private k-means clustering algorithm in the academic literature. Gilad from University of Haifa provided us an implementation of Feldman et al.'s k-medians clustering private coreset algorithm [12]. It is not a perfectly fair comparison because their main objective is k-medians clustering instead of k-means clustering. However, k-medians clustering approximates k-means clustering in the lower dimensional case, which justifies the comparison.

We pick the same privacy parameter ε and number of clusters k to do the comparison. The result is shown in

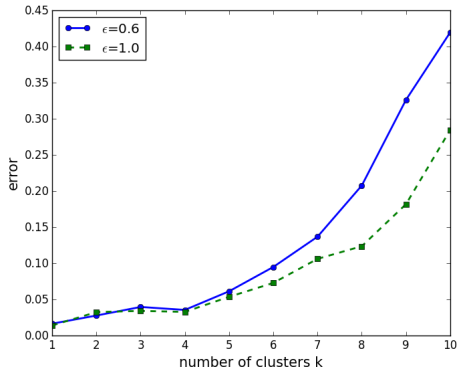


Figure 6: Error vs. the number of clusters k under different privacy parameter ε. The database size n is 1,000,000.

Figure 7. The error of our code is clearly smaller, which means we improve the utility under the same amount of information leakage. As for coreset sizes, the algorithm of [12] produces coresets that are 20× to 40× larger than the coresets computed in this paper. Furthermore, some of the coresets in [12] include negative weights; our coresets have only positive weights. Negative weights could be confusing to application end users.

6. DIFFERENTIALLY PRIVATE LOCATION FROM MOBILE SENSOR NETWORKS

We use our PRIVATE-K-MEAN algorithm in the context of mobile/roving sensor networks to hide the location of every single sensor node. Suppose a raw GPS database is constructed at the base station by collecting GPS tracks from every sensor node, and the administrator wants to gener-

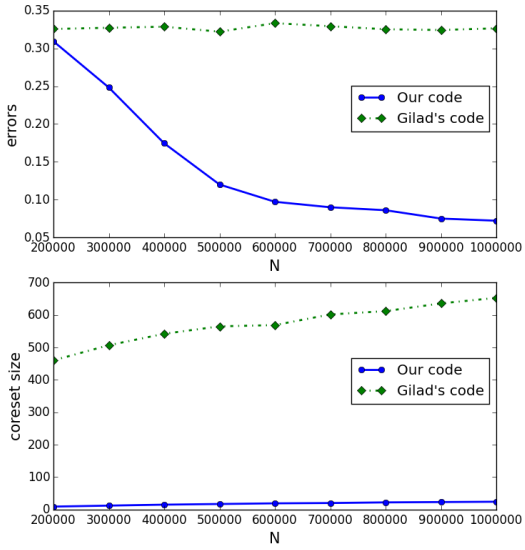


Figure 7: Comparison between our Private-k-Mean implementation and Gilad's implementation of Feldman et al's STOC'09 paper[12]. In the experiments, the number of clusters $k = 5$, and the privacy parameter $\varepsilon = 0.5$. The first graph shows the relationship between error and database size N, and the second graph shows the relationship between coreset size and database size N.

ate the k-means private coreset as a useful private sanitized database from original raw database. The objective is to allow the administrator to set the privacy parameters. Low privacy parameters result in more accurate date sanitization while higher privacy parameters result in more uncertainty in the sanitized data. The administrator would adaptively select the privacy level.

We implement the idea using the New York city yellow taxi data from January 2015 [1]. It visualizes the privacy and utility trade-off to non-technical end users to help them make decisions on which privacy level works the best for them. Some screenshots of the application applied to New York City cab data are shown in Figure 8.

Since it is hard for the administrator to get the intuition behind the ε and δ in the (ε, δ)-privacy guarantees, we do not expose those privacy parameters to them. Instead, we create discrete privacy levels corresponding to different information leakage parameters ε. Users use a scroll bar to choose between those different levels. The left most level is the most dangerous and has the most information leakage, and the right most level has the least information leakage. The GPS points in the resulting private coresets, private k-means cluster centers and non-private k-means cluster centers are shown on a map. When the privacy level increases, we can explicitly see that the coreset contains less points, and some information is hidden. For example, in Figure 8, in the lowest level, there are GPS points outside of Manhattan island in the coreset, while in the highest level, all GPS points in the coreset are inside the Manhattan island. This idea can be used by base station administrator to choose the desired privacy level based on what location data should be hidden. After that, they can publicize the private sanitized database (k-means private coreset) for further publicizations and investigations.

The application can also easily be extended to let a single sensor node manage the privacy of its own data. For example, say a sensor node (a person, a robot, or a vehicle) logs continuously its GPS tracks for further processing and historical logging, but there is a particular region where the activity of the node should be subjected to strong privacy considerations. Our algorithm can be triggered for data collection and display in that area, resulting in a similar private feedback as shown in Figure 8.

7. CONCLUSION

In this paper we provide an efficient algorithm for k-means clustering with privacy guarantees that are based on differential privacy. Our algorithm constructs sanitized databases of statistics for such problems while preserving the privacy of the individuals whose data generated them. These databases called private coresets provide: (i) approximation for the sum of squared distances of k-means to the original data. (ii) differential privacy. We designed and implemented the algorithm in two dimensional case and describe the experimental results. Finally, we present an application that is based on this algorithm that allows the roving mobile sensor nodes and sensor network administrators to collectively select the trade-off between privacy and utility/accuracy of the data.

We believe this work provides first steps towards the exiciting vision of protecting every individual's privacy in real life data analysis. Our goal for the near future is to apply the differentially private k-means algorithm to higher dimensional database and implement private coresets for other types of query classes.

8. REFERENCES

[1] New york city cab data. http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml. Online; accessed: 6-April-2016.

[2] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *Journal of the ACM*, 51(4):606–635, 2004.

[3] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar. Privacy, accuracy, and consistency too: a holistic solution to contingency table release. In *Proc. 26th ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems (PODS)*, pages 273–282, 2007.

[4] A. Beimel, K. Nissim, and U. Stemmer. Private learning and sanitization: Pure vs. approximate differential privacy. *CoRR*, abs/1407.2674, 2014.

[5] A. Blum, C. Dwork, F. McSherry, and K. Nissim. Practical privacy: the sulq framework. In *Proceedings of the Twenty-fourth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 13-15, 2005, Baltimore, Maryland, USA*, pages 128–138, 2005.

[6] A. Blum, K. Ligett, and A. Roth. A learning theory approach to non-interactive database privacy. In *Proc. 40th Annu. ACM Symp. on Theory of Computing (STOC)*, pages 609–618, 2008.

[7] K. Chen. On k-median clustering in high dimensions. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006,*

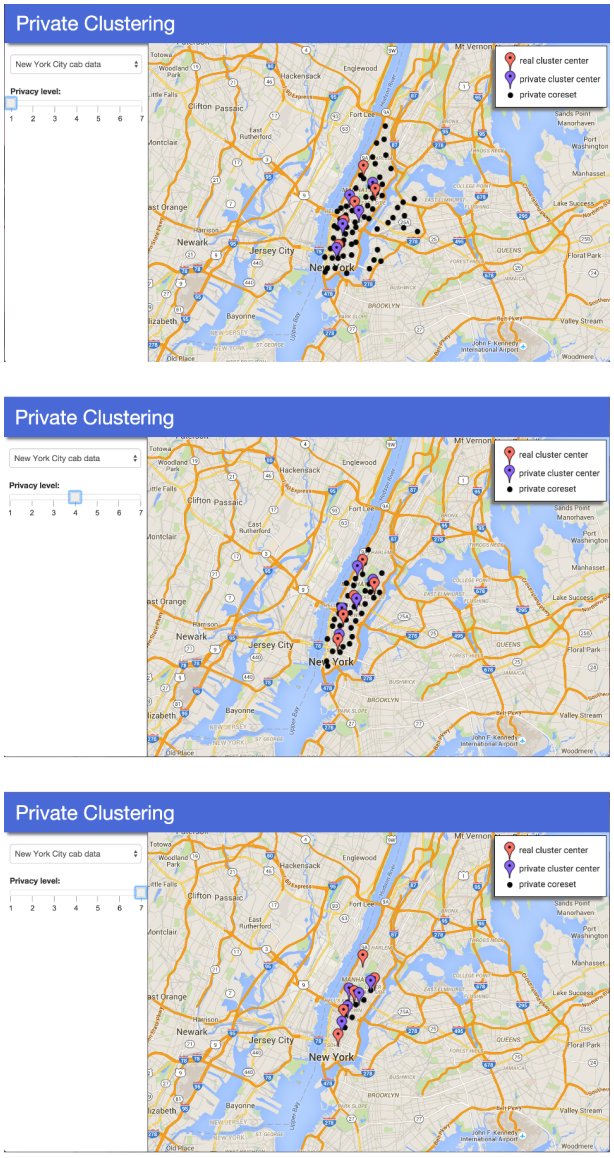


Figure 8: Application screenshots for New York City cab data on different privacy levels. The top one has the lowest privacy, and the bottom one has the highest privacy. Black dots represent the points in the private coresets. Red markers represent the non-private k-means clustering centers. Blue markers represent the private k-means clustering centers. In the screenshots, $k = 5$.

Miami, Florida, USA, January 22-26, 2006, pages 1177–1185, 2006.

[8] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology-EUROCRYPT 2006*, pages 486–503. Springer, 2006.

[9] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Proc. 3rd Theory of Cryptography Conf. (TCC)*, pages 265–284, 2006.

[10] C. Dwork and K. Nissim. Privacy-preserving datamining on vertically partitioned databases. In *Proc. 24th Annu. Int. Cryptology Conf. (CRYPTO)*, volume 3152 of *Lecture Notes in Computer Science*, pages 528–544. Springer, 2004.

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

[12] D. Feldman, A. Fiat, H. Kaplan, and K. Nissim. Private coresets. In *Proc. 41st Annu. ACM Symp. on Theory of Computing (STOC)*, pages 361–370, 2009.

[13] D. Feldman, M. Monemizadeh, and C. Sohler. A PTAS for k-means clustering based on weak coresets. In *Proceedings of the 23rd ACM Symposium on Computational Geometry, Gyeongju, South Korea, June 6-8, 2007*, pages 11–18, 2007.

[14] D. Feldman, M. Schmidt, and C. Sohler. Turning big data into tiny data: Constant-size coresets for k-means, PCA and projective clustering. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1434–1453, 2013.

[15] S. Har-Peled and A. Kushal. Smaller coresets for k-median and k-means clustering. *Discrete & Computational Geometry*, 37(1):3–19, 2007.

[16] S. Har-Peled and S. Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 291–300, 2004.

[17] S. Har-Peled and S. Mazumdar. On coresets for k-means and k-median clustering. In *Proc. 36th Annu. ACM Symp. on Theory of Computing (STOC)*, pages 291–300, 2004.

[18] S. Har-Peled and K. R. Varadarajan. High-dimensional shape fitting in linear time. *Discrete & Computational Geometry*, 32(2):269–288, 2004.

[19] D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. In *Proceedings of the Second Annual ACM SIGACT/SIGGRAPH Symposium on Computational Geometry, Yorktown Heights, NY, USA, June 2-4, 1986*, pages 61–71, 1986.

[20] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *Proc. 48th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 94–103, 2007.

[21] N. Mishra and M. Sandler. Privacy via pseudorandom sketches. In *Proc. 25th ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems (PODS)*, pages 143–152, 2006.

[22] K. Nissim, S. Raskhodnikova, and A. Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 75–84, 2007.

[23] K. Nissim, U. Stemmer, and S. Vadhan. Locating a small cluster privately. 2016. To appear.

[24] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory Prob. Appl.*, 16:264–280, 1971.